Ersatz

Ersatz User Guide

version 1.35

Jan J Barendregt

Ersatz

Web: www.epigear.com
Email: info@epigear.com

Ersatz

Contents

Introduction

Monte Carlo Simulation in Excel

Ersatz is a tool for Monte Carlo simulation in Microsoft Excel. It extends Excel with a range of functions that offer statistical distributions, the ability to draw randomly from these distributions, and repeat this a large number of times while gathering results in output functions. Applications include uncertainty analysis (aka 'risk analysis', and 'probabilistic sensitivity analysis'), microsimulation, bootstrapping, and probabilistic bias quantification. Ersatz is a generic tool, but offers some special functions that reveal its origin in health economic modelling. A list of the available Ersatz functions is in the *Ersatz Function Overview* document.

Installation

Ersatz 1.3 has been tested with Excel 2000, 2003, 2007, 2010, 2013, and 2016[1]. Installation is usually straightforward: make sure Excel is not running, and run the ErsatzSetup executable. If all goes well, the functions will be visible in the Excel Function Wizard (in the function category 'Ersatz') and you can open the example spreadsheets without seeing any "#NAME?" errors. See the section on 'Trouble shooting' when this is not the case.
The installation program installs an Excel add-in (called 'ersatzdll.xll') and an executable (called 'Ersatz.exe'), together with some documentation files and a few example spreadsheets. These items are accessible through the Windows Start menu, together with an Uninstall option. For a detailed discussion of the installation issues, see the Technical Appendix below.

Registration and license

In most cases you will need to obtain a license number and release code to be able to run Ersatz[2]. When you try to run Ersatz for the first time after installation it will show its Registration form, where you must enter a valid license number and a machine specific release code.
A license number will be issued to you after payment has been received, see the website (www.epigear.com) for payment options. Generally, the license is a personal one: you are granted the right to use Ersatz. When you use more than one PC, you have the right to run Ersatz on all of them. However, for each PC you will need to apply for a different release code. To obtain a release code, send your license number and the machine ID reported by Ersatz to ErsatzReg@epigear.com (please use copy & paste for the machine ID to avoid typos), we will respond as soon as possible.
Please contact Epigear (info@epigear.com) if you need a different license arrangement.

Features

Ersatz has a number of special features:

1. Multiple run mode.
2. An option to execute Excel macros before and after each iteration and run.

[1] Ersatz 1.3 actively supports the new features of Excel 2007 and higher. Ersatz supports both the 32-bit and 64-bit versions of Excel 2010 and higher. Please note that Ersatz is not compatible with the cloud-based Office 365 service.
[2] A license number and release code are not needed for the Trial and Workshop versions of Ersatz.

3. Conditional firing of the Ersatz random functions.
4. Several options for sensitivity analysis.
5. Special functions for microsimulation, in particular discrete event simulation.
6. A distribution viewer.
7. A choice of seven different random number generators.
8. Correlated random numbers, including for the multivariate Dirichlet and multinomial distributions.
9. Four different optimization algorithms, two deterministic and two stochastic.

These features are discussed in sections below.

Design issues

As mentioned above, Ersatz originates from a background of health economic modelling. It has been developed with academic research in mind. This implies that 1) transparency of the methods used is deemed important, and 2) rigour in the application of those methods is also important.

The issue of transparency hinges on the documentation. I've aimed to reference all the algorithms used, but it is very well possible that some are not referenced yet: the documentation tends to lag behind the implementation. Please bear with me.

As for rigour: Ersatz has been designed to be rather demanding in its inputs. For example, when the parameters of a distribution are outside their defined range, the function will return #NUM, and no summary statistics such as mean and standard deviation will be calculated, even if this happens in only a single instance of many iterations. The justification of that is that if such a thing happens, there is a design flaw in the modelling, and it is better to be confronted with that than to rely unwittingly on outcomes that may be flawed.

Another design issue is the interface. Ersatz does not attempt to melt into Excel, but is clearly a stand-alone application. In addition, the interface has been kept very simple (and therefore, hopefully, easy to navigate). These choices reflect my own preferences, which other people may not share.

Operation

Running Ersatz is done using the Ersatz executable (and not from within Excel). To operate, start Excel, and populate the spreadsheet with at least one input random function, and at least one ErOutput function (see the *Ersatz Function Overview* document for the parameters these functions take, or use the Excel Function Wizard). When done, start Ersatz, which will automatically connect to the running Excel spreadsheet (it will show the name of the spreadsheet in the titlebar). Then just press the 'Calculate' button for a default of 1000 iterations. When Ersatz has finished, additional tabs will become visible with 'Summary output' in table form, and histograms.

Clicking the check boxes in the Ersatz area of the Setting tab with 'Complete output' etc will reveal additional tabs with the corresponding data in tables. Results can be saved to file by choosing 'File|Save results', and can be saved to the clipboard (and then pasted into, for example, Excel) by choosing 'Options|Copy to clipboard', right clicking and choosing 'Copy', or simply by using the short-cut Ctrl C on the currently visible table or graph.

Iterations and runs

Ersatz distinguishes ‘iterations’ and ‘runs’. An iteration is a single recalculation of the Excel spreadsheet, using randomly drawn values from the input distributions. A run consists of a large number of iterations (typically at least 2000), and Ersatz collects all outcomes during the run and calculates for each outcome summary statistics such as mean, standard deviation, and CI. When the ‘Multiple runs’ option is chosen, this process is repeated as many times as the user specifies. For a detailed discussion of this option and its uses see the topic on ‘Multiple runs’.

This User Guide

Although in the remainder I will touch briefly upon the concepts of uncertainty analysis, bootstrapping, and microsimulation, this User Guide is certainly not meant as an introduction to these issues. The reader is kindly referred to the literature, see the references for some examples.
The aim of this document is to make the user familiar with Ersatz and its interface, possibilities, and quirks.

Technical note

Please consult the Technical Appendix below for statistical and other technical details.

Interface and features

Introduction

In this section I will describe the Ersatz interface, and simultaneously explain the various features the interface refers to. Controlling Ersatz happens through three interface elements: the ‘Settings tab’, the ‘Calculation panel’, and the ‘Main menu’. I will describe these elements in turn.

Settings tab

The Settings tab has two main areas: an Excel and an Ersatz area. In the Excel area the following check boxes are placed:

1. **No screen updates while running**. If checked this suppresses the Excel screen updates after each recalculation. This greatly speeds up the calculation. Default value is checked.
2. **Show mean values while not running**. If checked Excel will show the mean values of the Ersatz distribution functions in the spreadsheet, if unchecked the functions will return a random value drawn from the distributions (and you can draw new values manually by letting Excel recalculate the sheet through pressing F9). Default value is checked.
3. **Check for errors while running**. If checked Ersatz will for each iteration test whether the input parameters of the random functions comply with requirements, if not the function will produce #NUM and a non-fatal error message will be posted to the Message window. If not checked, this test will not be performed, and if the parameters are outside the required range it is unpredictable what the function will return. It is therefore strongly recommended to have this option on at all times, the performance penalty is not noticable. Default value is checked.

In the Ersatz area the following items are visible:

1. **Confidence intervals (%)**. You can choose confidence interval values between 50 and 99%. Default is 95%.
2. **Number of decimals in tables**. This controls the number of decimals in the output tables. You can set values from 0 to 10. Default is 3. Please note that this setting does not affect the number decimals in tables saved or pasted: in that case a high precision is maintained.
3. **Show scattergrams.** If checked Ersatz will display scattergrams of inputs and outputs on the ‘Scattergrams’ tab. Default value is unchecked.
4. **Show complete output.** If checked Ersatz will display the values of each output function for each iteration in a table on the ‘Complete output’ tab. Default value is unchecked.
5. **Show sorted output**. If checked Ersatz will display all values of each output function sorted from lowest to highest value in a table on the ‘Sorted output’ tab. Default value is unchecked.
6. **Show input**. If checked Ersatz will display the values of each input function for each iteration in a table on the ‘Input’ tab. Default value is unchecked.
7. **Show convergence graphs**. If checked Ersatz will display the running mean value of each output function for each iteration ‘Convergence’ tab. Default value is unchecked.

8. **Multiple runs**. If checked on the calculation panel an additional box is visible that allows to set the number of runs. At the same time the options 'Show complete output', 'Show sorted output', and 'Show input' become unavailable, while the 'Multiple run output mode' checkbox becomes enabled. The output and input options are disabled because they can easily become very big indeed with multiple runs. But complete input and output can be saved to file, and therefore is still available. For a detailed discussion of this option and its uses see the topic on 'Multiple runs'. Default value is unchecked.
9. **Multiple run output mode**. This checkbox is enabled only when the 'Multiple runs' box is checked. For a detailed discussion of this option and its uses see the topic on 'Multiple runs'. Default value is unchecked.
10. **Optimization**. If checked, this will open the Optimization tab, which allows access to Ersatz's four optimization methods and their options. For a detailed discussion of optimization, the four methods and their options, see the section on 'Optimization' below. Default value is unchecked.

This concludes the 'Settings tab' section.

Calculation panel

The calculation panel is the panel on the left of the main window, displaying the Ersatz logo. In addition it contains usually two, and occasionally three items, depending on whether the 'Multiple runs' option is chosen.
The calculation panel allows to specify the number of iterations and runs, and to start the calculation:

1. **Calculation**. This button starts the calculation, and allows to cancel it when it has started.
2. **Number of iterations**. This box allows to specify the number of iterations in each run. The default value is 1000 (but this is not the recommended value for uncertainty analysis, see the topic 'How much is enough?'). The box also contains an indicator that allows to monitor calculation progress.
3. **Number of runs**. This box is visible only when the 'Multiple runs' option on the Settings tab is chosen. Otherwise it is analogous to the 'Number of iterations' box, including a progress indicator.

This concludes the 'Calculation panel' section.

The Main menu

The Main menu is located just below the title bar of the main Ersatz window. It has four items (File, Options, View, and Help), each with several sub-items. First the 'File' item:

1. **File|Connect to Excel**. This item is disabled when Ersatz is connected to an Excel workbook. When the connection is lost, for example because you closed the workbook, it will become enabled and lets you connect again.
2. **File|Disconnect Excel**. This item is enabled only when Ersatz is connected to an Excel workbook. When chosen, Ersatz will disconnect and disable many options such as the Calculation button.
3. **File|Save results**. This item is enabled only when results are available after the calculation. When chosen it opens the 'Save results' window, which allows you to save complete inputs and outputs, and summary outputs, each in

a separate file. Check the items you want to save, specify valid file names (use the Browse buttons for that), and click the Save button when done.
Note that when multiple runs have been calculated each iteration will be prefixed with the run number. The file format is comma-delimited (with extension 'csv'): this format can be read by Excel and most statistical packages.
4. **File|Exit**. This exits Ersatz.

The following sub-items are located in the 'Options' item:

1. **Options|RNG**. RNG stands for 'random number generator'. Ersatz uses its own RNGs, and offers the user a choice of 7 different generators, which are shown in the 'RNG options' window. For details, see the topic on Random numbers.
2. **Options|Macros**. Ersatz allows to run Excel macros during execution. Please note that this will, as a rule, considerably slow down the calculation speed. Choosing this option will open the 'Macros' window for the specification of the macros and further options. See the topic on Macros for details.
3. **Options|Miscellaneous**. Here you can set various options:
   a) The maximum number of times that the ErTruncate function will resample the embedded random function to obtain a number that falls within the limits. The default number is 10.
   b) Whether or not Ersatz should use short cell addresses in its Sensitivity graphs. The default is yes.
   c) Whether or not Ersatz should use the ErSetItno function. Default is yes.
   See the *Ersatz Function Overview* for details on these functions.
4. **Options|Conditional firing**. Normally Ersatz functions are triggered on every iteration. Choosing this option will open the 'Condional firing' window, where you can change the default behaviour by input function. Note that for this condition to take hold the 'Conditional firing on' checkbox has to be checked. For details, see the topic on Conditional firing.
5. **Options|Copy to clipboard**. This option is enabled when one of the output or input grids or graphs is visible. Choosing this option will copy the grid or graph to the clipboard, from where it can be pasted into other applications, such as Excel and Word. It is also available by right-clicking the grid or graph.
6. **Options|Graph options** . This option allows to change the appearance of the current graph. It is also available by right-clicking the graph.
7. **Options|Correlation**. Chosing this item will open the 'Correlation' window. See the topic on 'Correlated random deviates' for details.

The 'Sensitivity' item contains two sub-items:

1. **Sensitivity|Univariate**. This opens the univariate sensitivity window. Please note that results from a run will be destroyed by this action.
2. **Sensitivity|Multivariate**. This opens the multivariate sensitivity window. Please note that results from a run have to be available for this option to be enabled.

See the topic on 'Sensitivity and uncertainty analysis' for details on these sensitivity options.

The 'View' item contains the following sub-items:

1. **View|Distributions**. This item opens the Ersatz Distribution viewer, a window where the user can choose from the list of distributions implemented in Ersatz functions, get a graphical representation, and change parameters to examine the effect on the distribution. The window also displays the corresponding mean and standard deviation.
2. **View|Messages**. This opens the Messages window, which displays some run statistics, such as numbers of inputs and outputs, length of calculation time, etc. It also displays error messages.

The ‘Help’ item has the following sub-items:

1. **Help|Ersatz User Guide**. Displays this document in a window.
2. **Help|Ersatz Function Overview**. Displays the *Ersatz Function Overview* document, which lists all Ersatz functions and details the parameters.
3. **Help|Example spreadsheets.** Displays and allows to open the example spreadsheets that come with the installation.
4. **Help|About**. Displays the About box, with details of the Ersatz version and serial number (if applicable).

This concludes the ‘Main menu’ section.

Special topics

Introduction

In this Special Topics section I will briefly touch on some general concepts in the field of modelling and uncertainty analysis, but focus on the features of Ersatz that relate to these concepts. Subjects are:

1. Bootstrapping and Monte Carlo Simulation
2. Microsimulation
3. Random numbers
4. Uncertainty and sensitivity
5. Good modelling practice
6. Multiple runs
7. Macros
8. Conditional firing
9. Optimization

The aim of this section is to give some background to the various features of Ersatz, including references to the literature, and help the user to make full and appropriate use of these features.

Bootstrapping and Monte Carlo Simulation

Numerical methods

Uncertainty analysis aims to quantify the uncertainty around a central (or point) result by, for example, giving a confidence interval around that central result. For instance, an incremental cost-effectiveness ratio (ICER) from an economic evaluation study might be assessed at \$34,000 per quality adjusted life year (QALY), with a 95% confidence interval (CI_{95}) of \$28,000 to \$41,500. The CI_{95} expresses that, given the data, we expect 95% of all possible outcomes to fall within this range[3].
A caveat is due here: with an uncertainty analysis we usually limit ourselves to the uncertainty due to the sampling error of the input data. Other sources of uncertainty are explored, if at all, using different methods.
For parametric distributions it can be quite easy to calculate a CI. For example, the CI_{95} of a variable with a Normal distribution with parameters μ and σ is given by $\mu \pm 1.96\ \sigma$. However, an ICER has several sources of sampling uncertainty: the uncertainty around the effect size of the intervention, uncertainty around the costs, and perhaps several more, and it then becomes impossible to find an analytical expression to calculate a CI.
Enter Monte Carlo simulation and bootstrapping. These methods use numerical (as opposed to analytical) methods to obtain an estimate of the uncertainty. They rely on re-sampling to obtain a distribution of outcomes, from which the standard deviation and a CI are derived. These two main variants of numerical methods are both supported by Ersatz.

Non-parametric bootstrapping

Non-parametric bootstrapping requires a dataset on the individual record level. From this dataset a bootstrap replicate of the same size is constructed by randomly drawing records from the dataset with replacement. This implies that some records will be present more than once, while others will not be present at all. From this bootstrap replicate the outcome of interest is calculated. This procedure is repeated many times, resulting in a distribution of outcomes.
Non-parametric bootstrapping has the big advantage that no assumptions on the distribution of the data have to be made. This makes it particularly useful when such assumptions become rather heroic. For large datasets the central limit theorem assures that the assumption of normally distributed statistics like the mean is close enough, but for small datasets this assumption often stretches credibility.
Ersatz has implemented non-parametric bootstrapping through the ErNonparam and ErNonparamCom functions, see the *Ersatz Function Overview* document for details.

Monte Carlo Simulation

Monte Carlo simulation (sometimes also called 'parametric bootstrapping') does not require individual record level data, but it does require you to make assumptions on the distributions of the variables in the model. Once all appropriate variables have been replaced by suitable distribution functions (including suitable parameters), the

[3] I know, from a frequentist's point of view this is not correct. However, uncertainty analysis is in many respects closer to the Bayesian approach to uncertainty, and within the Bayesian framework it is a valid statement.

model is re-calculated many times, each time drawing a random value from each of the distribution functions, again resulting in a distribution of outcomes.
The main issue with Monte Carlo Simulation is the choice of suitable distributions and parameters. Sometimes the choice is obvious (e.g. average height from a large population survey has a Normal distribution), sometimes almost by definition (e.g. Binomial and Multinomial and their conjugate Beta and Dirichlet distributions for categorical variables), but inevitably sometimes a rather subjective choice must be made. For a discussion on the choice of distributions and the estimation of parameters in the context of health economic evaluation, see the section on Good Modelling Practice below, and the book by Briggs et al (Briggs, Sculpher et al. 2006).
Ersatz has a whole range of distributions for Monte Carlo Simulation, see the *Ersatz Function Overview* document for the list of functions.

Combined parametric and non-parametric models

Ersatz allows mixed models, with some parameters described by parametric distribution functions, while others get their value from a non-parametric bootstrap.

How much is enough?

How many iterations should you use? A good question, and the answer is ‘it depends’. It depends on your model, the outcome statistic you are interested in, and on your own preferences. Generally speaking, the number of iterations is enough when repeated runs of your model with that particular number of iterations generate results that are sufficiently similar.
Note the subjectivity in ‘sufficiently similar’. When you are interested in central statistics such as the mean, you can for most models get away with less than 1000 iterations. However, when you are interested in confidence intervals, you will generally need more than that, because confidence intervals are derived from the infrequent samples from the extremes of the distributions.
Ersatz offers a convergence graph to let you inspect the stability of the outcome variables. This graph shows the running mean by iteration. Typically at the lower end of the number of iterations this running mean shows large volatility, that decreases towards the higher iterations end of the graph. Your model outcomes have converged on their mean value when the right part of the convergence graph is virtually a straight horizontal line.
As a general rule I recommend not to skimp on the number of iterations. Of course calculation time can become an issue, but for most models on today’s PCs the issue is minor.

Microsimulation

Introduction

In the discussion of bootstrapping above it was implicitly assumed that the model describes a population. Parameters, such as the risk of disease, are assumed to be equally applicable to all members of the population. When this assumption is obviously not met, the standard response is to subdivide the population, for example by age and sex, such that within the subpopulations the assumption of homogeneity is reasonable.

Subdividing the population to accommodate heterogeneity works, up to a point. It may be self-defeating when the number of subdivisions is so large that the model becomes unwieldy. This can easily happen with models describing various risk factors with several levels of exposure and time-dependent risks, for instance.

Another example of an area of research where subdividing the population is only a partial solution is that of cancer screening models, where much of the evaluation of a screening program is determined by the degree of heterogeneity that the model allows. In such cases, microsimulation may be the best option.

Individual level models

Microsimulation is a technique where the unit of simulation is the individual. The model describes an individual life history using probabilistic functions. An instance of a life history is created by drawing randomly from these distributions. Population level outcomes are obtained by creating many of these life histories, and deriving the desired outcomes, such as disease prevalence, from the individual level simulated data.

It will be clear that accommodating heterogeneity should be no problem in such individual level modelling. What may not be obvious is that microsimulation usually is very computationally intensive. To obtain reasonably stable population level outputs often a very large number of individual life histories needs to be created. In addition, microsimulation is generally more demanding in data requirements: to model heterogeneity requires data on heterogeneity. These drawbacks make microsimulation an option that needs careful consideration.

Microsimulation and Excel

Given the tendency of microsimulation to be computationally intensive, Excel is hardly the ideal environment for this technique. Microsimulation models are often written in general purpose compiled programming languages, such as Object Pascal and C++, that produce optimised executables. However, programming in such languages requires special tools, specific expertise, and usually a large effort. Therefore microsimulation in Excel may be an attractive alternative.

When you go down that road, model efficiency should be foremost in your mind. A well designed model may run several times faster than a poorly designed one, and that becomes important when run times are counted in hours rather than seconds.

An example will illustrate this. One of the returning issues in disease modelling is to assess an age at death, given age-specific conditional mortality probabilities such as routinely published by national statistical bureaus. In a discrete single-year time/age approach this can be done by drawing for each age a uniformly distributed number between 0 and 1, and assess whether it is smaller than or equal to the mortality probability of that age. The youngest age for which this is true then becomes the age at death. If your model considers ages from 0-100, this requires 101 random draws and comparisons (see example workbook ‘survival123’, spreadsheet ‘survival1’). While this works, it is not efficient. An about twice as fast way of obtaining the same result requires the mortality probabilities to be converted to a survival curve, and then use a single random uniformly distributed number between 0 and 1 together with the Excel INDEX and MATCH functions to look up this value in the survival curve (see example workbook ‘survival123’, spreadsheet ‘survival2’). The same technique can be used to determine age at disease incidence and other empirical data driven time-to-state-transition model variables.

Microsimulation and Ersatz

Using survival curves improves performance, but the implementation in ‘survival2’ is still not ideal. The main problem is that time is treated as a discrete variable, which of course it is not. In ‘survival2’ (as in ‘survival1’) it is assumed that death occurs half way through the age interval (the ‘0.5’ in the formulas), which is reasonable enough on the population, but less so on the individual level. Moreover, it is not clear what to do when more than one cause of death can occur in the age interval (say, a disease specific and an ‘all other’ causes death). This so-called ‘competing risks’ problem is introduced by treating time/age as a discrete variable.

A better design choice is therefore to use a continuous time, discrete event approach instead of the perhaps more intuitive discrete time one. Briefly, in continuous time, discrete event modelling everything that happens is called an ‘event’, and each event has an exact time linked to it. The model proceeds from one event to the next (see for example (Law and Kelton 2000) for an introduction to discrete event modelling). This is elegant because it solves the competing risks problem and is efficient at the same time.

Ersatz has of course the continuous parametric distributions that are well suited to this kind of modelling, but it also implements a continuous empirical survival function, ErSurvival. This function takes conditional failure probabilities by discrete time as parameters, and returns a continuous survival time (see the *Ersatz Function Overview* for details). ErSurvival also takes a parameter that allows it to return survival time, conditional on having survived so far. The ‘survival3’ sheet in workbook ‘survival123’ has an implementation of ErSurvival that is equivalent to the survival1 & 2 worksheets, but then in continuous time.

Other Ersatz features that come in handy for microsimulation are ‘Multiple runs’ and the ErRunOutput, ErIteration, and ErConditional functions. When doing population level modelling an iteration stands for a recalculation of the entire population, given a random draw from uncertain parameters such as effect size. A ‘run’ consists of a large, say 2000, number of iterations, with the effect of uncertainty in the parameters reflected in the distribution of outcomes.

When doing individual level modelling (microsimulation) an iteration stands for an individual, and a ‘run’ is the equivalent of doing a single iteration in a population level model. Consequently, to do an assessment equivalent to a population level analysis with 2000 iterations, in microsimulation you will have to do 2000 runs with each a number of iterations equal to the population size you deem proper (here is where microsimulation truly becomes computationally intensive).

And with a microsimulation run equivalent to a population level iteration, it is also clear that you want a single draw from uncertain parameters such as effect size to apply all individuals in your population (= a microsimulation run). In other words, in

microsimulation you have to do uncertainty analysis by doing multiple runs, with some individual level functions (such as ErSurvival) drawing a random value at each iteration, while population level functions (such as ErRelativeRisk for effect size) draw a random value only once for each run.
Ersatz offers the tools to implement this. See the topic on *Useful functions for multiple runs* in the section on *Multiple runs* below and the *Ersatz Function Overview* for details.

Combating randomness in microsimulation

Health economic evaluation is all about comparing a specific intervention with a comparator that is in all respects the same except for that intervention. As such it is using the same design as a randomised controlled trial (RCT), the only difference being that in addition to measuring the health outcomes in each arm it is also measuring the costs.
In an RCT an important issue is to make the subjects in the intervention and control arms as comparable as possible. To that end a study population is selected that is similar in all attributes that can be measured, and randomisation is used to control for chance (or, equivalently, for attributes that cannot be measured). Ideally you would like to have each subject to act as its own control, but apart from the limited applicability of cross-over designs this ideal cannot be reached in practice (and even then!).
A microsimulation model mimicking an RCT that compares disease survival with and without some drug, for example, would simulate life histories in both arms by randomly drawing ages at incidence, deaths from all other causes, and disease survival. The difference between the arms would originate from drug induced systematic differences in the survival, and all chance differences due to the random draws.
We are, however, trying to evaluate the drug induced systematic difference only, all the other random differences are just muddling the picture (or, to put it more precisely, increasing variance).
While this is inevitable in RCTs, it is avoidable in microsimulation modelling. In microsimulation it is possible to make life histories the same between arms in every respect except the survival. A single draw determines age at incidence, and another single draw age at death from all other causes in both arms. To avoid random difference in the survival Ersatz implements a variant of the ErSurvival function (called ErSurvival2) that takes as an additional parameter a uniformly distributed number between 0 and 1. This allows to apply the same random draw to two different survival functions.
Such a set up makes, at least in the microsimulation model, subjects act as their own controls, thereby greatly reducing variance (and increasing precision). In the example workbook BreastCa the two cases are compared. This workbook implements the survival effect of trastuzumab (Herceptin™) for her2-positive women as compared to standard therapy survival.
In worksheet Breastca1 the two arms have independently drawn ages at incidence, all other causes death, and survival after breast cancer incidence. In the BreastCa2 worksheet the two arms are identical in all these respects except for the survival effect of tratuzumab. The standard deviation of the survival difference is reduced from about 13.5 to 1, the 95% uncertainty interval from about –26..+30 to 0..3. A huge reduction in uncertainty.

Microsimulation is a very powerful technique but is has clear drawbacks as well, and the inherent (so-called "first order") randomness is one of them. Clever design, however, can greatly reduce this first order randomness. An additional technique to reduce it, so-called "common random numbers" is discussed in the next section on random numbers.

Random numbers

Introduction

Bootstrapping and microsimulation depend on random numbers: the model uses distributions to describe crucial variables, and a particular outcome is calculated using specific values for these variables that are drawn randomly from the distributions. This implies, by the way, that the outcome of the model is a random variable as well. While this is the purpose of an uncertainty analysis, because it allows you to quantify the uncertainty of that outcome, it can also be a real nuisance.
When, for example, you assess the health and cost outcomes of a particular medical intervention by comparing model outcomes with and without the intervention, you don't want this comparison to be confounded by the randomness of that outcome. And fitting a random model to observations can be a pain too.
The usual response to combat the randomness of the model outcome is to increase the number of iterations, but this has of course the drawback of increasing calculation time. Another response is to use 'common random numbers', see the section on this topic below.
Computer programs can produce random numbers using algorithms called 'random number generators' (RNGs), and Ersatz offers the choice of seven different ones.

Random number generators

An 'algorithm that produces random numbers' is of course an oxymoron: a deterministic process (an algorithm) cannot produce random numbers. The outcome of a RNG is therefore more correctly described as 'pseudo random numbers': something that looks like random numbers, but isn't. An important property of a RNG is how well the pseudo random numbers resemble the real thing.
The basic operation of a RNG is as follows. Each RNG produces a large but fixed number of fixed 'random' numbers in a fixed sequence. The large number is called 'the cycle length' or 'period'. A 'seed number' is used to determine where in this cycle the RNG will start to generate a stream of 'random' numbers. When the requested number of 'random' numbers is larger than the cycle length the RNG simply restarts at its entry point.
From this discussion it will be clear that a RNG produces anything but random numbers. It will also be clear that the cycle length is an important property of a RNG: longer cycle lengths are preferred.
So there are two main characteristics that determine the quality of a RNG: its statistical properties (how well do the numbers resemble random numbers) and its cycle length. High quality RNGs have good statistical properties and a long cycle. Unfortunately, there is a trade-off: high quality RNGs require as a rule more computational effort than low quality ones. The build-in RNG of Excel (accessible through the RAND() function) prior to version 2003 was not very good; with version 2003 and later a better algorithm (based on Wichman & Hill) has been implemented. Ersatz uses its own RNGs and therefore can offer the user a choice how to trade off speed against quality.

The Ersatz RNGs

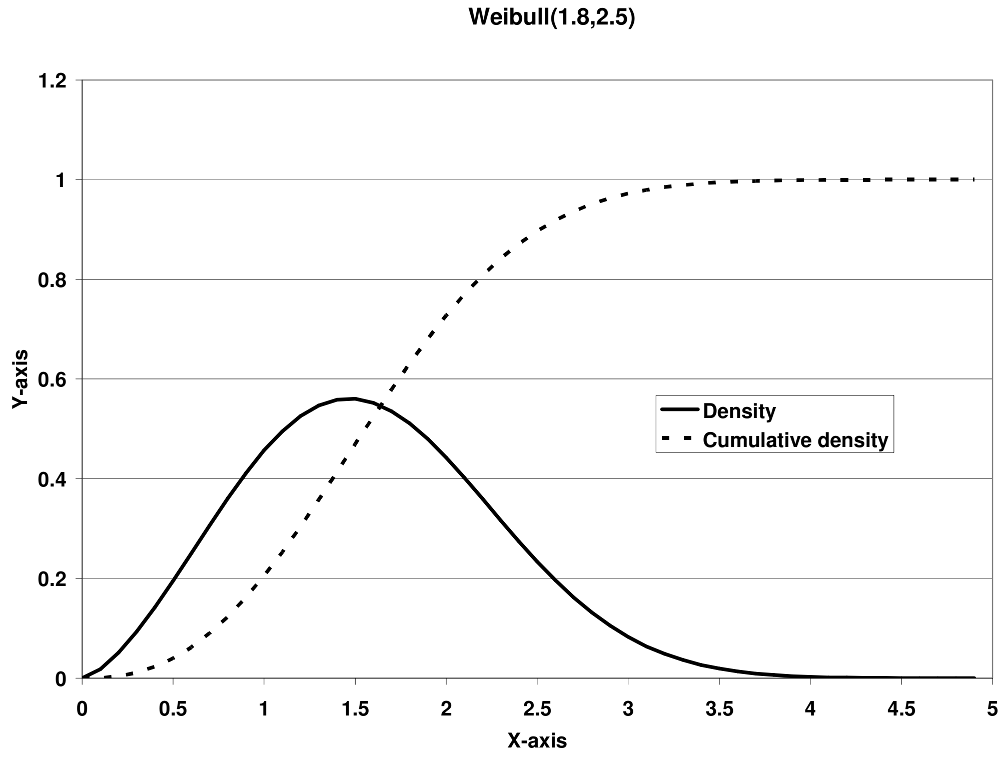
Ersatz has seven build-in RNGs, mostly based on the Ultimate Random Number Suite as programmed by Peter N Roth and Stefan Hoffmeister (and the following text is partly based on their release notes). The RNGs are, by and large in order of increasing quality:

1. Quick and dirty
2. Park & Miller minimal standard congruential generator
3. Park & Miller with a Bays & Durham shuffle
4. L'Ecuyer's two-series combo plus a shuffle
5. Mersenne twister
6. Fast Marsaglia
7. Top-quality Marsaglia

The RNGs number 1-4 are from *Numerical Recipes* (Press, Teukolsky et al.). The Mersenne twister is based on the algorithm developed by Makoto Matsumoto and Takuji Nishimura (see http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html) and has excellent statistical properties. The Marsaglia generators (6 & 7) provide an extremely long cycle (more than 1.0e356) and also have excellent statistical properties (Marsaglia and Zaman).
The default RNG in Ersatz is the Fast Marsaglia. For many applications this is an excellent choice, if probably quality overkill. On the other hand, in my experience there is very little speed advantage to be had from moving to a lower quality, but faster RNG: the computational bottlenecks are more likely to be in Excel than Ersatz. But the user may want to experiment.

Random deviates

RNGs produce uniformly distributed random numbers between 0 and 1 (an art that, as the reader will have deduced by now, comprises a whole research field of its own). The only Ersatz function that reproduces the output of the RNG directly is ErUniform01(), the function that returns a uniformly distributed random number between 0 and 1. Other Ersatz distribution functions return random deviates from other distributions, such as Normal, Gamma, etc. How are these obtained?
Getting random deviates from a specific distribution, given a random number between 0 and 1, can be very simple indeed. All that is needed is the inverse of the cumulative distribution function (CDF).
The CDF(x) is the integral (continuous distributions) or the sum (discrete distributions) of the density function from the lowest value of the range the function is defined for to x. While the density function returns the probability mass at a specific point x, the CDF(x) returns the probability mass from the lowest defined value up to and including x. The CDF is therefore a non-decreasing function between 0 and 1. The figure shows a Weibull(1.8,2.5) density function, and the corresponding CDF. For each value on the x-axis you can read from the y-axis how much probability mass is lying to the left of it.



The inverse of the CDF does something similar but the other way around: pick a value from the y-axis, and read off the corresponding value of the x-axis. If you pick a random value between 0 and 1 from the y-axis, you can thus read off a randomly chosen value from the Weibull(1.8,2.5) distribution. The CDF of the Weibull distribution is:

$$F(x) = 1 - \exp\left(-\left(\frac{x}{\alpha}\right)^{\beta}\right)$$

The inverse of this function can be written as:

$$F^{-1}(x) = \alpha(-\ln(1-x))^{\frac{1}{\beta}}$$

To generate randomly drawn numbers from the Weibull(1.8,2.5) distribution in Excel you type into a cell "=1.8*(-LN(RAND())^(1/2.5))" (1-*x* is equivalent to *x* when *x* is a uniformly distributed random number between 0 and 1), and recalculate (press F9). Too easy. The problem is, however, that for only very few distributions, such as the Weibull and the Exponential, an analytical expression for the inverse CDF exists (none exists for the Normal, for example). So how does Ersatz (and other software) obtain random deviates from distributions without known inverse CDFs?
As with the generation of uniformly distributed random numbers between 0 and 1, this comprises a whole field of research in its own. A range of algorithms, often very ingenious but hardly intuitive, has been developed. Some examples (with code) appear in *Numerical Recipes* (Press, Teukolsky et al.), Law and Kelton have a more elaborate selection (with pseudo code) (Law and Kelton 2000), but the canonical book

in this area (also with pseudo code) is *Non-Uniform Random Variate Generation* by Luc Devroye[4] (Devroye). Few people will ever be in the situation to need a comprehensive overview of algorithms to generate random deviates, but if you happen to be one of those, this book is strongly recommended.

Common random numbers

As mentioned in the introduction of this section, one of the problems of random numbers is that they are, well, random. This makes the outcome of the model random as well, and makes it impossible to assess whether outcome differences are due to real differences or to randomness. In addition, it confuses optimization routines used to fit the model to observed data. Under these circumstances you would like to have the same random numbers over and over again, a randomness reduction technique that goes by the name of 'common random numbers'.
From the discussion above on RNGs, you can conclude that by providing the same seed number to the RNG, you get the same stream of quasi random numbers each time. And indeed, that is a solution many programs use to produce common random numbers. However, in combination with Excel this technique is highly unreliable. Excel determines the order in which the functions in your workbook are evaluated, and this order may change with even minor changes to the workbook. So if you have more than one Ersatz function in your workbook, the fixed stream of random numbers may end up in a different order at the Ersatz functions, thereby breaking common random numbers. It is for this reason that Ersatz does not offer the option to set the seed number of the RNG.
You can achieve common random numbers in Ersatz/Excel by creating streams of random numbers for each of your input variables, save these to Excel, and reuse them as many times you want. This is the way to do this:

1. Run your model with Ersatz set to as many iterations as you will need.
2. When Ersatz in done, check the 'Show input' box, and click the input tab. This will show a table with your input variables and the values generated for each iteration.
3. Copy this table (Ctrl-C), and paste it into Excel.
4. Replace your input Ersatz functions with 'ErFixed(range,ErIteration())', where 'range' is the Excel range which contains the random numbers of the corresponding input variable. For details on the ErFixed and ErIteration functions, see the *Ersatz Function Overview* document.

Note that if you have more than 1 input variable, as is likely, it is a good idea to have a single ErIteration function in your spreadsheet, with all the ErFixed functions linked to it (see the Common random numbers example spreadsheet). Also note that it is a good idea to have several instances of streams of common random numbers and to change around the one in use once in a while: avoid relying on a single one on the off-chance that it produces a very exceptional result.

[4] This book from 1986 has been out of print for a long time, with the publisher refusing to re-print. This has enraged the author such that he had the whole book scanned and put up his website, where it is available for free (http://cg.scs.carleton.ca/~luc/rnbookindex.html). It's a big download, but worth every bit of it.

Correlated random deviates

It may be that your model needs random draws from different distribution functions to be correlated. For example, you may want the prevalence of diabetes in the model population to be positively correlated with the randomly drawn average body mass index (BMI).

The best described method to obtain correlated random deviates is for the Normal distribution. This method uses the covariance matrix of *N* Normally distributed variables, such as you would obtain from a statistical package. The so-called Cholesky decomposition of this matrix provides factors that allow to obtain randomly drawn numbers from those *N* Normal distributions with the required correlation structure (see (Briggs, Sculpher et al. 2006) for an example for *N*=2). This method is implemented in the Ersatz ErCorrNormal and ErCorrNormalCom functions, see the *Ersatz Function Overview* document for details.

For other than Normal distributions the picture is less clear. There are a number of analytical approaches for bivariate distributions (such as the Gamma), but very little for more than two, and I am not aware of any analytical solutions for correlated random draws from different distributions.

To provide correlated random draws from *N* arbitrary distributions Ersatz therefore implements a rank correlation solution. The algorithm is as follows:

1. Starting point is a user provided correlation matrix.
2. Using the Cholesky decomposition, *N* correlated N(0,1) random deviates are drawn for each iteration, and subsequently ranked.
3. For each of the *N* arbitrary distributions random deviates are drawn for each iteration, and subsequently sorted.
4. At each iteration the rank numbers of the correlated Normals are used to obtain the correspondingly ranked random deviate from the arbitrary distributions.

This method is implemented in the Ersatz ErRankCorr and ErRankCorrCom functions, see the *Ersatz Function Overview* document for details. The nature of the algorithm requires all the activity to be performed at the start of the run, at each iteration the ErRankCorrCom functions simply provide a number from the pre-calculated list.

While rank correlation is not the same as Pearson product moment correlation, in many cases the results are pretty close, see the RankCorr example spreadsheet for an example of rank correlated draws from a Gamma, a Weibull, and a Poisson distribution.

Correlated multinomial and Dirichlet distributions

The rank correlation algorithm described in the previous section work for univariate distributions only. But, as has happened with several other functions available in Ersatz, I needed correlated multivariate, in particular Dirichlet, distributions for my own research, so there.

The area of correlated multinomial and Dirichlet distributions seems an untrodden field: I’m not aware of any previous work. First I will define what exactly is meant here by correlated random draws from these multivariate distributions. It is defined here as a number of multivariate distributions of equal dimensionality, with all dimensions having the same correlation between distributions. So there is just a single correlation matrix, and each category has the correlation specified in the matrix with its corresponding category in the other distributions.

In the remainder of this section, I will discuss the algorithm for the Dirichlet distribution only, but the same algorithm is used for the closely related multinomial distribution. The algorithm is based on rank correlation.
The challenge is that a Dirichlet distribution has the properties that the sum over the categories always equals 1, and that the categories have a negative covariance. Of course, neither of these properties should be compromised by the imposition of a correlation structure with other Dirichlet distributions.
The algorithm to generate Dirichlet distributed random numbers, based on (Devroye 1986), uses for each category a random draw from a Gamma distribution with the category parameter as the first parameter for the Gamma and 1 as the second. The outcomes are summed over the categories, and each category outcome is then divided by the sum to obtain the desired proportions for the Dirichlet.
Given that the draws from the Gamma distribution are independent, this leads to the following rank correlation algorithm to obtain correlated Dirichlet distributions:

1. For each category of the Dirichlet distributions for each iteration draw correlated random numbers from a standard Normal distribution using the Cholesky decomposition of the correlation matrix.
2. Rank the numbers for each category.
3. Draw numbers for each category and iteration from the Gamma distribution, and sort for each category.
4. For each category and iteration, use the ranks of the correlated Normals to obtain numbers from the Gamma outcomes.
5. For each iteration, sum over the categories and divide each category number by the sum.

Again, rank correlation is not the same as linear correlation, and in the case of correlated Dirichlet distributions the achieved linear correlation tends to be a bit closer to zero than the one specified in the correlation matrix.
The algorithm for the stand-alone multinomial distribution in Ersatz is based for each category on a draw from a binomial distribution, conditional on the draws from previous categories (Devroye). This makes it incompatible with a rank correlation algorithm as described above. The correlated multinomial function is therefore based on the Dirichlet algorithm, with the outcomes obtained by a rounded multiplication of the Dirichlet proportion with the sum over the category numbers, except for the last category which is assigned the remaining number. This gives a very close approximation of the conditional binomial approach.
See the *Ersatz Function Overview* document for details on the implementation of these functions.

Valid correlation matrix

A correlation (or covariance) matrix for 3 or more distributions can be invalid, or, to put it mathematically, a correlation matrix needs to be positive (semi) definite to be valid. This means that no eigenvalue must be negative and at least one positive. When a covariance matrix is obtained from a statistical package, this requirement will be satisfied (but beware of rounding error!), but when assembling a correlation matrix of 3 or more distributions from pairwise observations, it is easily violated. Ersatz checks the provided correlation matrix, and it is a fatal error when it is invalid.
Ersatz therefore offers the option to check your correlation or covariance matrix for validity. Choose ‘Options|Correlation’, paste your matrix into the grid (it will

automatically resize to fit), and click the ‘Check’ button. If ‘Matrix valid’ appears, you’re on your way.
If the matrix proves to be invalid, Ersatz can suggest an alternative but valid matrix, as close as possible to the original one. Ersatz calculates a valid correlation matrix by shifting the original correlation matrix by its lowest eigenvalue. If you provided a covariance matrix, this valid correlation matrix is used to calculate a valid covariance matrix.

Sensitivity and uncertainty analysis

Introduction

Uncertainty and sensitivity analysis mean different things to different people. In this section I will describe my understanding of uncertainty and sensitivity analysis, and how Ersatz can deal with them.
There are several kinds of uncertainty in a health economic evaluation, Briggs et al for example distinguish four, among them uncertainty relating to analytical methods and extrapolation (Briggs, Sculpher et al. 1994). The present discussion is limited, however, to outcome uncertainty as a consequence of uncertainty in the values of the input parameters of the model. This limitation therefore excludes, for example, the so-called 'structural uncertainty' that follows from the fact that there is more than one way to model a certain health intervention (actually, an unlimited number of ways).

Sensitivity analysis

Sensitivity analysis I define as the quantification of the effect of variation in *specific* input parameters on the outcome. The function of sensitivity analysis is to show which variables have the largest impact on the outcome. For the researcher this has the specific benefit of pointing out which input variables need to be estimated with high precision in order to reduce the uncertainty in the outcome. Ersatz implements two types of sensitivity analysis.

Univariate deterministic sensitivity analysis

In this type of sensitivity analysis the values of the input parameters are varied one by one, and for each variation the output variables are calculated. The variation in the input can be either plus and minus one standard deviation, or plus and minus 10% of the mean input value. Output is as tornado graphs and a table.

Multivariate probabilistic sensitivity analysis

This type of sensitivity analysis builds on the results of an uncertainty analysis, and therefore requires that first an uncertainty analysis is run (see below). Given all the values of all the input and output variables, the probabilistic senstivity analysis then calculates the correlation coefficients between each input and output pair of variables. Ersatz implements a number of methods to calculate these correlation coefficients (for details on these methods, see for example (Conover 1999)):

1. Pearson's Product Moment Correlation Coefficient.
   This method assumes that there are no correlations between the input variables and that there exists a linear correlation between the input and output variable pair. In particular the linearity assumption will often be violated, so this might not be a good choice.
2. Spearman's Rank Correlation Coefficient.
   Unlike Pearson's PMCC this method does not assume linear correlations, however it does assume that there are no correlations between the input variables.
3. Kendall's tau.
   Kendall's tau uses a different method, but is for all practical purposes equivalent to Spearman's RCC.

4. Spearman's input-input Rank Correlation Coefficient.
   This option allows to investigate whether there are correlations between the input variables. Note that the scattergrams allow a visual inspection of correlations between model variables.
5. Partial Rank Correlation Coefficient.
   The Partial RCC is provided in case the input variables are correlated. It is, however, a rather computationally involved method, in particular when the number of variables is large.

When offered several options, the question becomes which to use. The default method is Spearman’s RCC, and this method is usually adequate. Depending on taste the user might prefer Kendall’s tau. But in both cases it is wise to check the input variables for strong correlations, and if they exist to use the Partial RCC instead.

Uncertainty analysis

Uncertainty analysis I define as the quantification of the simultaneous and combined effect of the uncertainty in the input variables on the outcome of interest. Rather implicitly the uncertainty in the input variables that is meant here is uncertainty due to sampling error. For the uncertainty analysis we replace in our model fixed parameter values by distributions[5] with a mean and standard deviation, and repeatedly recalculate the model with values sampled from those distributions. This makes sense only for parameters that can be meaningfully described by a distribution, and these are in practice parameters that have been (or could have been) estimated from sample data.
Not all inputs with uncertain values confirm to this criterion. A classical example in health economic evaluation is the uncertainty of the discount rate, the degree of time preference included in the analysis. Health economists have endlessly debated what the discount rate should be, but no consensus has been reached. So while the value of the discount rate is definitely uncertain, it is not uncertainty due to sampling error. Its effect is therefore best explored by univariate sensitivity analysis: simply recalculate the model outcome using a range of values for the discount rate.

Conclusion

So what should you do, sensitivity or uncertainty analysis? The answer is an emphatically “Both!”. Sensitivity and uncertainty analysis both examine the uncertainty in the outcome of your analysis, but they answer to different questions. Moreover, they give only partial answers: only uncertainty due to uncertain values for input parameters is explored. Methods to include quantified uncertainty due to other sources, such as model structure, are currently not available, and may very well never become so.

[5] For guidance on which distributions to use, see the section on Good Modelling Practice below.

Good modelling practice

Introduction

Good modelling practice (GMP) is not something that is set in stone, but rather a collection of rules and norms that, when followed, will make your model qualify. Some of these rules are hard ones, but most are not, and in the end there is also an aesthetic dimension. This section is not meant as a general introduction to GMP. It will touch only briefly on most aspects, and then concentrate on an aspect very pertinent to modelling with Ersatz: the choice of appropriate distribution functions for given model variables.

Rules and norms

Formal validity

An important rule, and the only really hard one, is that of formal validity: the model should calculate what it is supposed to do. In other words, no bugs.
As such this seems obvious, in practice it is not. The power of Excel as a modelling environment is its flexibility, but this is its main weakness as well. Because anything goes, it is quite easy to make a royal mess. It requires serious self-discipline to avoid this. In addition to self-discipline, techniques such as stress-testing (putting variables to their extreme values) and check-sums (making sure all items in your model are accounted for: none are generated or disappear miraculously) are recommended.

Simplicity

Given the research question, a model should always be the simplest one that can anwer it. This rule is basically the application of *Ockham's razor* to modelling, and therefore goes a long way back. It is an important principle, and a pre-requisite for the next rule.

Transparency

'Lack of transparency' is probably the most used argument to challenge results from a modelling exercise. While this argument may sometimes be borne out of laziness or even bad faith, it is all too often justified. You cannot expect people to believe your model's results while the model itself is a black box to them. Your duty as a modeller is to reveal how your model works, and to explain the results, all in a language that is understandable to people not familiar with modelling, but likely to be experts in the field your model pertains to. If you fail to convince them, your work will have been in vain (even if you were right).

Elegance

This is probably the hardest criterion of all to define. However, when a model is valid, simple, and transparent, it already comes a long way to being elegant as well. But elegance requires something extra, a semblance of effortlessness. Good writers achieve elegant prose by developing a clear structure to their argument, and then by putting the right words in the right places (needless to say this requires hard work: the effortlessness is a semblance, and the benefit is to the reader).
In much the same way elegance of a model rests on a clear structure (simplicity and transparency), and of the choice of the right distributions in the right places. Choosing the right distributions is the subject of the next section.

Choosing appropriate distributions

Introduction

When replacing a model variable by a distribution function and its parameters, the modeller in many cases has to rely on the reported standard deviation or confidence intervals of that particular variable. The confidence intervals are typically calculated using the assumption of a Normal distribution, even when this leads to impossible results. For example, the lower limit of a confidence interval of a proportion may end up in negative territory, or the upper limit is greater than one, by definition impossible values.

When reporting confidence intervals no real harm is done, it only looks a bit awkward. However, when randomly sampling from such a Normal distribution, and recalculating the model with an impossible value, the results are not just awkward, but are actually invalid.

A ‘solution’ is then to truncate the Normal at 0 and 1. However, this is a bad solution, for three reasons. It is ugly, because it causes discontinuities at 0 and 1. It is invalid in the sense that the resulting distribution will not have the intended mean and standard deviation. And it is bad modelling practice, because a perfectly good solution exists.

When a randomly drawn value is to be constrained between 0 and 1, the natural candidate distribution is the Beta, a distribution that under no circumstances will produce values outside the 0..1 range. The Beta distribution has two parameters, by convention called α1 and α2. To calculate the parameters of the Beta distribution that has the reported mean and standard deviation, you set the equations for the mean and standard deviation of the Beta equal to μ and σ, the mean and standard deviation of the Normal:

$$\mu = \frac{\alpha 1}{\alpha 1 + \alpha 2}$$

$$\sigma = \sqrt{\frac{\alpha 1 \alpha 2}{(\alpha 1 + \alpha 2)^2 (\alpha 1 + \alpha 2 + 1)}}$$

Re-arranging these equations gives the values of α1 and α2 as functions of μ and σ:

$$\alpha_1 = \frac{\mu(\mu - \mu^2 - \sigma^2)}{\sigma^2}$$

$$\alpha_2 = \frac{\alpha_1 (1 - \mu)}{\mu}$$

The resulting Beta distribution has the intended mean and standard deviation, shows no discontinuities, and stays within the 0..1 range by definition.

When no standard deviation is reported, but a confidence interval is, it is easy to calculate the standard deviation *s* from the confidence interval:

$$s = \frac{H_{CI} - L_{CI}}{2z}$$

with H_{CI} and L_{CI} the upper and lower limit of the confidence interval respectively, and z the factor from the standard Normal that applies to the confidence interval (e.g 1.96 for the 95% confidence interval).
To summarise, the resampling technique used in uncertainty analysis requires to make sure that sampling from the distributions used never produces values that are outside the valid range of that variable. You can achieve that by using the ErTruncate function, but it is much more elegant to use distributions that will by definition not produce values that should not occur. A number of suggestions are in the remainder of this section.

Prevalence and probability: the 0..1 range

With a two category prevalence (e.g. diseased and non-diseased) the Binomial distribution is, almost by definition, the correct choice. In that case the Binomial is defined as the number of diseased in a population of size *N*. When you want to describe the prevalence as a proportion, you can use a random draw from the Binomial, divided by its *N* parameter. The resulting number will never get outside the 0..1 range.
The next step is to get the correct parameters. The Binomial has two parameters: *N* and *p*. The *p* parameter is simply the observed prevalence (expressed as a proportion). The standard deviation *s* is:

$$s = \sqrt{Np(1-p)},$$

again divided by *N* when expressed as a proportion. Given a standard deviation of the prevalence *s* you can therefore calculate the *N* that goes with the *p* and the standard deviation by:

$$N = \frac{s(1-s)}{p^2}.$$

However, the Binomial is a discrete distribution. When *N* is big, the resulting prevalence random draws will be nearly continuous, but for small *N* this may not be the case. To avoid the discontinuities that follow from the discrete nature of the Binomial, you can use the Beta distribution as an approximation. The Beta is, in Bayesian circles, often referred to as a 'conjugate' distribution of the Binomial (Gelman, Carlin et al.). The parameters of the Beta that approximates the Binomial (with parameters *N* and *p*) are

$\alpha 1=Np$, and

$\alpha 2=N(1-p)$.

Because prevalence proportions and probabilities share the same range of 0..1 values, the discussion above applies equally to transition probabilities in your model.

Multiple categories prevalence and transition probabilities

The discussion above is pertinent to two category prevalence, but there exist generalisations to multiple categories. The Multinomial distribution is the generalisation of the Binomial, and the Dirichlet distribution the generalisation of the Beta. Both are implemented in Ersatz as ‘component’ random functions, see the *Ersatz Function Overview* for details.

Two-sided arbitrary limits
What if you need two-sided limits on the values from a random function other than 0 and 1? Again the Beta distribution is useful here. If you need random draws from a distribution *f* between the values *min* and *max*, you can use

$$f(\alpha 1, \alpha 2, \min \max) = \min + (\max - \min)\mathrm{Beta}(\alpha 1, \alpha 2)$$

The mean of this function will be:

$$m = \min + (\max - \min) m_\beta = \min + (\max - \min)\frac{\alpha 1}{\alpha 1 + \alpha 2},$$

and the standard deviation:

$$s = (\max - \min) s_\beta = (\max - \min)\sqrt{\frac{\alpha 1 \alpha 2}{(\alpha 1 + \alpha 2)^2 (\alpha 1 + \alpha 2 + 1)}}.$$

In other words, you can simply shift and scale the distribution because the mean and standard deviation are measured in the same units. For the same reason, the derivation of the parameters of the Beta is straightforward: calculate the mean and standard deviation of the Beta as

$$m_\beta = \frac{m - \min}{\max - \min}, \text{ and}$$

$$s_\beta = \frac{s}{\max - \min},$$

and then use the equations

$$\alpha_1 = \frac{m_\beta \left(m_\beta - m_\beta{}^2 - s_\beta{}^2\right)}{s_\beta{}^2}, \text{ and}$$

$$\alpha_2 = \frac{\alpha_1 (1 - m_\beta)}{m_\beta}$$

from above to calculate the parameters of the Beta. The equations above are useful to calculate the parameters of the ErBeta4 function, which implements a rescaled Beta function along the lines described here.

One-sided limits
One-sided limits are best modelled using one of the distributions that have a 0..∞ range: Lognormal, Gamma, Weibull, and Exponential for continuous distributions, and Binomial, Geometric, Negative Binomial, and Poisson for discrete distributions. To obtain random numbers with a lower limit *A* simply use *A+dist*, where *dist* stands for one of the continuous or discrete functions mentioned above. Similarly, for random numbers with a upper limit *A* use *A-dist*.

Relative risk
For the relative risk (RR) the standard assumption is that ln(RR) has a Normal distribution with parameters ln(RR) and its standard error SE[ln(RR)]. More formally:

$$RR \sim \exp(N(\ln(RR), SE[\ln(RR)])).$$

This amounts to saying that RR has a Lognormal distribution.
To obtain an estimate of the SE[ln(RR)], consider the following two-by-two table:

	Exposed	Unexposed
Cases	a	b
People at risk	N1	N0

For the rate ratio the SE is obtained by:

$$\mathrm{SE}[\ln(RR)] = \sqrt{\frac{1}{a} + \frac{1}{b}}$$

For the risk ratio the following equation holds:

$$\mathrm{SE}[\ln(RR)] = \sqrt{\frac{1}{a} - \frac{1}{N_1} + \frac{1}{b} - \frac{1}{N_0}}$$

If no two-by-two table is available but the confidence interval is known, then the SE[ln(RR)] can be obtained using the following equations (assuming a 95% confidence interval):

$$\begin{aligned}\ln R_{95} &= \ln\left(CI_{95}^{+}\right) - \ln\left(CI_{95}^{-}\right)\\ &= \ln(RR) + 1.96SE[\ln(RR)] - \ln(RR) + 1.96SE[\ln(RR)]\end{aligned}$$

$$\mathrm{SE}[\ln(RR)] = \frac{\ln R_{95}}{2 \times 1.96} = \frac{\ln\left(CI_{95}^{+}\right) - \ln\left(CI_{95}^{-}\right)}{3.92}$$

The drawback of the assumption that ln(RR) has a Normal distribution is that the mean of the sampled values from this distribution is somewhat higher than RR because of the skewed Lognormal distribution. Ersatz therefore has an ErRelativeRisk function with an adjustment such that the mean of the sampled values equals RR. The drawback of the correction is that the resulting uncertainty interval is shifted somewhat than otherwise would occur. The user will have to decide which is the lesser of these two evils.

See the *Ersatz Function Overview* for details on the ErRelativeRisk function, and (Barendregt 2010) for details on the correction.

Some more general guidance
Apart from the issue of the valid range of a model variable there are some more general considerations to choose specific distributions for certain types of variables. Below is a list of suggestions.

- In many cases the Normal distribution is an obvious choice: by virtue of the central limit theorem the mean of a sample of sufficient size will have a normal distribution, irrespective of the underlying distribution. Moreover, 'sufficient size' does not mean huge: this property of the mean emerges for surprisingly modest sample sizes.
- As discussed above, prevalence and probability are best modelled using Binomial or Multinomial distributions, and their continuous approximating distributions, the Beta and the Dirichlet respectively.
- A constant rate[6] in an age or time interval is often assumed to have a Poisson distribution with the number of cases as its parameter. As a continuous approximation of the Poisson distribution the Gamma distribution can be used. The Gamma that approximates a Poisson(ν) distribution is Gamma(ν,1). When not the number of cases but the mean (denoted by *m*) and the standard deviation (*s*) of the rate are given, the parameters of the approximating Gamma(α,β) are:

  $$\alpha = \frac{m^2}{s^2}, \text{ and}$$

  $$\beta = \frac{s^2}{m}.$$

- Survival can be modelled either non-parametric (i.e. empirical) or parametric. For empirical survival Ersatz provides the ErSurvival function, see the *Ersatz Function Overview* for details. Parametric survival is often modelled using the Weibull distribution ("time to failure" in engineering cycles), but the Lognormal and Gamma distributions mostly give very similar results.
- Costs are usually attached to units of resources: number of drug prescriptions, hospital days, etc. Such discrete count data are modelled using the Poisson distribution with the number of units as the parameter. Again the Gamma distribution can be used as a continuous approximation, see the discussion of rates above.
- Utility and disability weights are usually confined to the 0..1 range, which makes the Beta the distribution of choice. Sometimes health states worse than death are allowed, the corresponding utility can be modelled as 1- a skewed

[6] Unfortunately, epidemiologists are often very sloppy in the use of technical terms. It is fairly standard to speak of the 'prevalence rate', while prevalence is clearly a proportion, for example. Rate is here properly defined as the number of cases in a time interval, divided by the person-years at risk (or any approximation thereof).

distribution such as the Lognormal, see the discussion on one-sided limits above.

The take-home message from this section is that choosing distributions for the variables in your model is not an arbitrary process: a careful choice of distributions avoids problems, makes the model more elegant, and in some cases, such as prevalence, relative risk, and utilities, aligns the model with existing theory and practice in epidemiology and health economics.
Readers will have noticed that I haven't recommended the use of the Triangular distribution in the guidance above, despite this distribution being rather popular. In fact, I agree with Briggs *et al* that the use of the Triangular distribution should be avoided (Briggs, Sculpher et al. 2006). It is an ugly distribution because of the discontinuities, and it lacks a grounding in statistical theory.
Nevertheless there may be circumstances where you have indeed not more than the opinion of a single expert about a minimum, a maximum, and a most likely value for one of your variables[7]. That is a rather desperate situation, but even then you are probably better off using the Pert instead of the Triangular distribution.
The Pert distribution (see the *Ersatz Function Overview* for details) takes the same parameters as the Triangular (minimum, mode, maximum), but it is in fact a re-scaled and re-parametrised Beta distribution. It therefore is firmly grounded in statistical theory, and, while serving the same purpose, does not have the drawbacks of the Triangular distribution. Ersatz provides the Triangular distribution, but I most emphatically do not recommend its use.

[7] When you have several expert opinions, non-parametric bootstrapping is a good option.

Multiple runs

Introduction

Ersatz offers the option of doing multiple subsequent runs. When checking the ‘Multiple runs’ box on the ‘Settings tab’, an additional box on the ‘Calculation panel’ becomes visible, allowing to specify the number of runs Ersatz will perform consecutively.
The reader may wonder why this is useful. If you do, say, ten runs in a row, each run will supersede the results from the previous one, and at the end you will simply have the results from the last run, with the previous ones only having wasted your and the computer’s time.
When used like this, the multiple runs option indeed makes no sense. But there are circumstances where it does, and these include, but are not limited to, the following:

1. Each run implements a different intervention.
2. Each run uses a different assumption on a parameter such as discount rate.
3. Each iteration applies the intervention to different sub-populations, such as different age groups, and the result for the population is the sum over these sub-populations.
4. Microsimulation, where an iteration stands for an individual, and a run for a population.

The case of microsimulation is discussed in the section of that name above, here I will discuss the first three instances.

Different interventions, assumptions, or sub-populations

When multiple runs are used to process a number of different interventions and/or using various assumptions that affect the outcome, Ersatz is basically being used in batch mode. Each of these interventions/assumptions could have been analysed in a single run, but the user prefers to lump them all together in one big go.
I must admit to doubts about the usefulness of this. It requires to put in special functions to save the results, but you will miss out on much of the detailed output anyway, and it will tie up your computer for an extended period. But some people prefer it that way, and Ersatz lets you.
A stronger case for multiple runs, and one that I have used repeatedly, is when the same model (with different data) is applied to several sub-populations, and the result you are after is the sum (or average) over those sub-populations. In particular, since age is such an important determinant of health outcomes, we always model age explicitly for our health economic evaluations. However, the model structure is the same for each age group, it is just the parameters that differ.
In those cases the multiple runs option allows to model each age group explicitly without having to replicate the model for each age group. The way we did this was to set the number of iterations equal to the number of age groups we distinguished, and set the number of runs equal to what normally is the number of iterations (say, 2000). With the ErIteration() function, that returns the number of the current iteration, we cycled through the age groups in each run, while using the Excel Index and Match functions to look up the appropriate data for the age group at hand.
While this works a treat, it also gets beyond basic Excel use and moves into the realm of programming. The example workbook ‘BreastCaMarkovMC’ shows how to do this, and the next section discusses the Ersatz and Excel functions that are particularly useful for this kind of modelling.

Useful functions for multiple runs

ErRunoutput and ErRunSensInput

The ErRunOutput function is a variant of the ErOutput function that picks up only a single value at the end of each run. When the option “Use ErRunOutput functions” is checked, Ersatz will use the ErRunOutput function values from multiple runs to calculate medians, means, and confidence intervals across those multiple runs, in effect treating the outcome of a run the same as the ErOutput function treats the outcome of each iteration.
ErRunSensInput is the function similarly matched to ErSensInput, and can be used to do sensitivity analysis across multiple runs.

ErCondStoreArray and ErCondRetrieveArray

The ErCondStoreArray and ErCondRetrieveArray functions are variants of the ErStoreArray and ErRetrieveArray functions that store and retrieve Excel ranges, depending on a Boolean input parameter being true. Of course this Boolean can be made true on any condition, but one of them is at the end of a run, which would make these function behave similarly to the ErRunOutput function discussed above.

ErTotal and ErMean

The statistical functions ErTotal and ErMean return a single value at the end of a run. Used together with the ErRunoutput function these (and other statistical) functions allow you to determine what will be stored in the ErRunoutput function. ErTotal gives the sum of its input parameter over the iterations in a single run, while ErMean gives the average. Other statistical functions that report a value at the end of a run are available, see the *Ersatz Function Overview* for details.

ErIteration, ErSetItno, and ErRunno

Often when using multiple runs it is necessary to know which iteration or run is currently being executed. For example, when there are a number of age groups to loop through in each run, you often need to know which age group (=iteration) you are currently dealing with in order to look up the applicable data. The ErIteration function returns the number of the current iteration, and ErRunno of the current run.
ErSetItno is handy when you need in a particular model a specific number of iterations at all times. For example, you have modelled 15 age groups, or 2448 patients (in a microsimulation model). The ErSetItno function allows you to enter that specific number in the spreadsheet, making sure that always the correct number of iterations is executed.

ErFixed

If you need to look up data depending on the current iteration or run number, there are several options. Ersatz offers the ErFixed function, that lets you get a specific value from a range. Excel has the Hlookup and Vlookup functions, but a better (faster) choice is often the Index function. In this context Excel’s Match function can be useful too, see the Excel Help for details and the example spreadsheets for applications.

ErConditional

By default all Ersatz random functions draw a new value at each iteration, and in most cases this is the desired behaviour. However, when using multiple runs there likely

are variables you want new random draws for only some of the iterations, or even just once per run. For instance, an effect size often applies to all age groups, so if you are using the multiple runs option to model various age groups in your population, you'll want to draw a single value per run and apply that to all age groups.
Similarly, in microsimulation, where an iteration stands for an individual, you have to do uncertainty analysis by doing multiple runs, with some individual level functions (such as ErSurvival) drawing a random value at each iteration, while population level functions (such as ErRelativeRisk for effect size) draw a random value only once for each run.
For this situation Ersatz offers the conditional firing option and the ErConditional function. Which one of the two is most suited will depend on your application, see the section on Conditional firing below.

Excel macros

Introduction

Extended functionality in Excel is often implemented by macros, based on Visual Basic for Applications (VBA) code. Ersatz lets you execute Excel macros before and/or after each iteration, and before and/or after each run.

Recalculation during macro execution option

During macro execution the spreadsheet is likely to be recalculated, which would normally prompt the Ersatz functions to draw a new value. However, by default the Ersatz functions will not draw new values during macro execution. If this is not the desired behaviour you must uncheck the check box on the Macro window.

Excel macros are slow

A word of warning is in place here: Excel macros are slow. This may not be obvious when you execute a macro just once, but when a macro is executed hundreds or thousands of times, it starts to add up.
For example, in one of my own applications an uncertainty analysis was done for an admittedly large spreadsheet (Gartner, Barendregt et al. 2009). On each iteration Ersatz would draw prevalences of smokers, former smokers and non-smokers from survey data using Dirichlet functions. The results from these draws were then used to fit a model of smoking uptake and cessation, the fit being done with the Excel Solver add in that was called on each iteration by a macro[8]. The 2000 iterations took about 24 hours to run!
This may be an extreme example, but the take-home message is that if you can avoid using macros, for example by using the techniques discussed in the section on Multiple runs, you will usually be much better off.

[8] This was before Ersatz version 1.1 implemented its own optimization functions.

Conditional firing

Introduction

By default the Ersatz random functions draw a new value at each iteration. In most instances this will be the desired behaviour, but there are exceptions. Above I discussed microsimulation and the use of the multiple runs option, both cases where you might want to restrict the number of times some of the Ersatz functions draw to, for example, once at the start of each run. And of course there may be other reasons to restrict the number of times the functions should fire.
Ersatz offers two ways to implement this: the ‘Conditional firing option’ and the ‘ErConditional’ function. They are operated very differently, one as an option in Ersatz, the other as a function in the spreadsheet.

Conditional firing option

This option is accessed by choosing ‘Options|Conditional firing’. This will open the Conditional firing window, which displays a list of the Ersatz input functions in the connected spreadsheet, together with the conditions that apply to each of them (which will be the default value of ‘Every iteration’ at first display). The condition fields in the list can be changed for each function individually using the drop-down list of that field. Available options are ‘Once at the start of the run’ and ‘None’, the latter meaning that the function will return its mean value at each iteration. Please note that in order for the conditions to take effect, the check box on this window needs to be checked.
You can save the settings of the condition fields to a spreadsheet (use Ctrl-C), edit them, and paste them back in (use Ctrl-V), or equivalently use the popup menu when you right-click the list. The ‘Reset list’ button will reset all conditions back to the default condition specified in the ‘Default condition’ radio box.

ErConditional

ErConditional is one of the functions that Ersatz adds to Excel. ErConditional takes a boolean and an arbitrary other value as parameters, see the *Ersatz Function Overview* for details. This other value may come from an Ersatz random function or from any other source. When the boolean is TRUE ErConditional returns the current other value, when FALSE the previous one, or, to put it differently, the value returned by ErConditional is kept constant.
For example, when you obtain the boolean value from “(ErIteration()=1)” it will be TRUE only at the start of each run. A random value from a function that is the value parameter of ErConditional that is drawn at this first iteration will then be returned by ErConditional for the rest of that run.
Of course the boolean value can be made to depend on any user-defined condition, making this a very flexible choice.

Conclusion

Ersatz offers two possible solutions if you want to restrict the number of times its random functions will fire. The ‘Conditional firing’ option is rather limited, but requires no changes to the spreadsheet. The ErConditional function is the most versatile, but has to be implemented in the spreadsheet. Which one will work best for you will depend on your application.

Optimization

Introduction

Like some other topics touched upon in this User Guide, optimization constitutes a whole area of research by its own. It is possible, and people have done so, to write whole books on the subject, or even on a single method. This section will only give a very brief introduction to the topic, and primarily with a view to what has been implemented in Ersatz, and how to use it. For those looking for a more comprehensive overview of the topic, the Wikipedia page on ‘Optimization (mathematics)’ is a good place to start.

There are many different computer algorithms available for optimization, and new ones, or modifications of existing ones, are developed all the time. There are two reasons for this (and they are very important to take on board):

1. Optimization tends to be costly (as in ‘taking a lot of computer time’), so there is a constant drive towards more efficient algorithms.
2. There exists no ‘one size fits all problems’ algorithm. An algorithm that blazes through one kind of problem can spend an inordinate amount of time on another, or even fail miserably. And some algorithms are designed to tackle very hard problems (which means they throw a lot of computing power at it), and you would waste your and the computer’s time to apply it to a problem where a more efficient method could do the job just fine in a fraction of the time.

The take-home message here is that you should always be aware of the kind of optimization problem you have on your hands, and what method would be most suited to solve it.

Because no single method is best under all circumstances, Ersatz implements four different ones. The remainder of this section will first discuss some basic concepts in optimization, and then describe the kinds of optimization problems the user should be aware of, matching them with the most suitable optimization method. Finally, I will give details of the four methods, the various options they take, and give some rules of thumb when to use what.

Some basic concepts

Setting the stage

An important distinction is the one between combinatorial and continuous function optimization. Much of the mathematical literature is on combinatorial optimization, which looks at problems with a countable but very large number of distinct outcomes. A famous example is the ‘travelling sales person’ problem: find the shortest route to visit N places without visiting any place more than once. While this is an interesting problem (and certainly not without practical application), it is not the kind of problem that we are concerned with here.

Continuous function optimization comes in when we need to find the parameters of a function that will make the function return a specific value[9]. More formally, we have a function f which takes a N-dimensional vector of parameters $\mathbf{x}$, and we want to know the value of $\mathbf{x}$ which will make $f(\mathbf{x})=y$. This kind of problem returns time and

[9] Please note that, despite the adjective ‘continuous’, this includes functions that return discrete outcomes, such as ‘number of patients developing metastases’. For the purpose of the discussion here, this is a continuous outcome, albeit one with granularity.

again in health economic evaluation (as in many other applications). For example, what combination of probability to develop metastases and parameters for survival with metastatic disease will reproduce the observed number of deaths from breast cancer?
The reader should note two potential difficulties at this point. Firstly, there may not exist an **x** such that $f(\mathbf{x})=y$. And secondly, there may exist multiple **x**s such that $f(\mathbf{x})=y$. The second difficulty, while not trivial, is one that can be solved by using additional information. For example, a solution may exist that requires one or more of the x in **x** to be negative, which may be impossible (because they are probabilities, for example). This leads to the topic of constrained optimization, see below. Nevertheless, even after adding information, you may end up with a range of possible parameter values instead of some point values. Such is life.
The first difficulty, no **x** exists such that $f(\mathbf{x})=y$, is tackled by reformulating the problem from achieving a specific target outcome to a minimization problem. In particular, we define a loss function such that minimizing the loss function with respect to the input vector **x** will result in an outcome that is as close as possible to the desired outcome y. A frequently used and effective loss function is the squared difference, $(f(\mathbf{x})-y)^2$, which can be generalised to a sum of squared differences in order to achieve a range of desired outcomes (see below for a more detailed discussion of the loss function).
A further advantage of using a loss function is that all optimization problems (finding a maximum, a minimum, or a specific value) are reduced to the same problem: finding the minimum of a loss function. This is why optimization in Ersatz is implemented as minimization only.

An example: the OptimizationIt workbook
To make all this more concrete, consider the example workbook OptimizationIt.xls (as all examples available from the Ersatz entry in the Windows start menu). This workbook has (made-up) data from three annual surveys about smoking prevalence. In order to estimate the annual trend in smoking prevalence, a linear function has been fitted through the three observations, which leads to an annual change in smoking prevalence of -2.7 percent point (cell B18).
However, the smoking survey data are subject to sampling uncertainty, and therefore the estimated annual trend should get an uncertainty interval. The sampling uncertainty of the survey data has been modelled by Beta distributions with parameters number of smokers and non-smokers, respectively (if you wonder why Betas and why these parameters, check out the topic on Good Modelling Practice in this User Guide).
In order to obtain an uncertainty interval for the estimated annual trend we need to draw random values from the Beta distributions, refit the linear function, and repeat this procedure many times.
In the workbook this has been implemented using the Ersatz optimization functions ErMinimize (cells B12 & 13) and ErMinimizeResult (cell B16). These two functions always work in tandem, and both need to refer to the same name in order to do so (in the example 'fit1' in cell B10).
In addition to the name parameter, ErMinimize takes an Excel range as an input parameter. These values act as the starting point of the minimization, see below for more on starting values. The output of ErMinimize is a range of the same dimensions

as its input range, and to achieve that ErMinimize has to be entered as an Excel array formula[10].
The ErMinimizeResult function takes, in addition to the name parameter, the outcome of the loss function as a parameter. The loss function is modelled as the sum of squared differences (using Excel's SUMXMY2 function) between the observed survey data points and the fitted values from the linear function.
To run the optimization, check the 'Optimization' box in the Ersatz area of the Settings tab: this will open the Optimization tab. You then type the name of the ErMinimize function ('fit1') into the Name textbox, leave the other settings unchanged, and click the 'Calculation' button. Ersatz will do the specified number of iterations, fitting the linear function after each iteration. Output of the run is the distribution of the trend in smoking prevalence, and the sum of squared differences between the survey data points and the fitted values (this should be positive but close to zero).

An iterative process
So how do optimization algorithms go about their business? Basically, by trial and error. The algorithm puts in a trial vector of parameters **x**, and obtains the outcome of the loss function. Next it puts in another trial vector, and evaluates the loss function again. If this looks better (i.e. is less), then it thinks it is on right track, and tries something similar. If it is worse, it tries something else. It stops when nothing it comes up with gives an improvement, or when it has reached the maximum number of tries.
In other words, it is an iterative process, and that is why it tends to be computationally costly. In large part, the computational load is determined by the number of loss function evaluations (in Excel speak: recalculations of the workbook) the algorithm requires.
Some algorithms, such as the Quasi-Newton, put a lot of smarts into guessing what the next trial vector should be, and consequently can be very fast, but are vulnerable to deviations from the assumptions they make about the problem. Others, such as the Down-hill Simplex method, don't try as hard, therefore require more loss function evaluations and generally are slower, but are also more robust.

Local minima: the bane of optimization
A common problem in optimization is the existence of local minima. What we want to find is the global minimum: the point where the loss function reaches its lowest value. But many optimization problems can be visualised as hilly landscapes: there will be one valley that is the deepest, but there may be many shallower ones around. Once the optimization algorithm has entered such a shallow valley, it will tend to find its lowest point, and then quit.
A number of strategies have been proposed to deal with this problem, none of which, however, offers a sure fire solution.

1. If you have a reasonably good prior idea which values of the parameters will give rise to the global minimum, supply these values as the starting point of the optimization. More in general: you should provide starting values for the

[10] See the Excel Help on array formulas. Briefly: select the range of cells you want the output range to appear in (needs to have the same size as the ErMinimize input range), type your formula, and then press CTRL-Shift-Enter. This will result (if all went well) in your formula appearing in all the selected cells, embedded in curly braces.

optimization algorithm that are sensible. If you provide starting values such that the model outcome will hardly change as a consequence of the trial parameters the optimization algorithm comes up with, it will likely give up right away.

2. Redo the optimization with different starting values. If you end up with a different outcome, you can be sure that local minima are a problem (of course you cannot be sure that you have found the global minimum). If you get the same outcome, you may have found the global minimum (but again, you cannot be sure).
3. Restart the optimization at the solution found (this is an option you can set in Ersatz). The optimization algorithms take some bold first steps, and resort to smaller and eventually tiny steps if they think to be close to a minimum. If the minimum you've found is a local one, the bold first steps of a restart might take you out of it (and then again, they might not).

With all the caveats in the above paragraph, it will be clear that these strategies are only partial solutions at best. Therefore a more radical approach has been developed: stochastic optimization.

Stochastic optimization

The discussion so far has (implicitly) been about deterministic optimization: algorithms that always will choose a direction that gives a lower outcome of the loss function. As we have seen, this makes them vulnerable to ending up in local minima. Stochastic optimization uses algorithms that try to avoid this pitfall by introducing a stochastic element into the decision making. Ersatz implements two such algorithms: Simulated Annealing and Cross-Entropy (more details on the methods are below). By casting a wider net, these algorithms can indeed avoid at least some of the local minima, but there is a price. The Cross-Entropy method, for example, needs something like 20 times as long to run the OptimizationIt.xls workbook than the deterministic Quasi-Newton and Down-hill Simplex methods. Clearly, you would rather not use Cross-Entropy unless necessary.

Noisy optimization

Yet another hard problem in optimization occurs when the outcome of your model (and therefore your loss function) is stochastic. It is easy to see why this will upset the optimization algorithm: when the target value moves around randomly, it becomes very hard to zoom in on it.

But some methods get more upset than others. The Quasi-Newton method as implemented in Ersatz cannot deal with noisy optimization at all: it goes completely off the rails. The Down-hill Simplex method is somewhat more resilient and does zoom in on a minimum of a moderately random loss function; however, when there, it tends to keep trashing around and only stops when it exceeds its maximum number of tries. The Ersatz implementation therefore offers an optional additional stopping criterion; see the discussion of the Down-hill Simplex method below.

By their very nature you would expect the stochastic optimization methods to be best suited for noisy optimization, and I have indeed obtained reasonable results with the Cross-Entropy method. However, in my experience its result is not very precise, and I tend to do a follow-up optimization with the Down-hill Simplex method.

Of course, when your model is very noisy, all the methods will fail, simply because as stochasticity increases, the amount of information about the location of the minimum of the loss function decreases. You may need to look into variance reduction methods when this is the case, see for example the section on ‘Combating randomness in microsimulation’ above.

The loss function

The loss function is a crucial element of the optimization process, and you should give it some careful thought. Importantly, you should realise that the optimization algorithm knows nothing about your model: it simply proposes a trial vector of numbers, and gets back a single number from the loss function.
So that single number should contain all the information it needs: it should be lower when the proposed vector produces a better result (and vice versa). Moreover, it should be proportionally lower: when the result is much better, the loss function should return a much lower number. And importantly, there should be no discontinuities in your loss function result: a vanishingly small change in one of numbers of the proposed vector should not lead to a sudden jump in the result.
Put differently, your loss function should be a smooth function of all the elements that determine whether you consider a particular solution better or worse. Smooth here means: no discontinuities and no flats (i.e. areas where the loss function returns the same value for a large range of the input variables).
In addition, it should preferably be a balanced function: the various elements should enter the loss function with a comparable size, rather than the physical size of the variable. For example, if you enter a total number of incident cases together with prevalence as a proportion, the optimization algorithm will basically ignore the proportion (a number between 0 and 1) and focus on the much larger number of incident cases exclusively.
So you might need to scale your variables that enter the loss function such that they are of comparable size. You might also want to weigh the variables such that the ones you deem more important are more closely matched in the solution than the less important ones.
If you come away from this discussion with the impression that designing a loss function is a bit of an art, then that is exactly the kind of message I wanted to convey. Typically, in complex cases with many variables in the loss function, its design is something you will revisit a few times, depending on the outcome of a previous optimization run. But simply taking the sum of squared differences between target and model outcomes as a loss function usually is a good place to start.

Constrained optimization

It often happens that you want to fit some variables that can take on only a limited range of values. For example, you want to fit the parameters of a Weibull distribution that models the time to failure of a hip replacement such that the number of secondary hip replacements fits the observed one.
The two parameters of the Weibull are constrained to be larger than 0. But remember, the optimization algorithm knows nothing about your model, and will happily propose negative values for the Weibull parameters. The result is mayhem: the Weibull function returns #NUM!, and so do your model and the loss function, and the optimization algorithm is thrown off course.
You might think you can fix this in the loss function: if one of the Weibull parameters proposed is ≤0, then return a large penalty instead of #NUM!. However, this is a bad solution: it creates a discontinuity in your loss function (which, see discussion above,

should be smooth), and the optimization algorithms have a tendency to get stuck on discontinuities.
A much better solution is to use a transformation on the number the optimization algorithm proposes such that the transformed number is never ≤0, and use the transformed number as the actual Weibull parameter. This lets the optimization algorithm roam freely without causing discontinuities in the loss function.
There are basically two kinds of constraints: one-sided and two-sided[11]. These can be handled by two solutions, with some modifications to accommodate more general cases. Let x be the number proposed by the optimization algorithm, then the transformation y, subject to some constraint, will be:

1. One-sided, y>0: $y = e^x$.
2. One-sided, y>A: $y = A + e^x$.
3. One-sided, y<A: $y = A - e^x$.
4. Two-sided, 0<y<1: $y = \frac{1}{1+e^{-x}}$
5. Two-sided, A<y<B: $y = A + (B - A)\frac{1}{1+e^{-x}}$

These transformations take much of the sting out of constrained optimization. Strongly recommended.

Iteration or Run

On its Optimization tab Ersatz offers, in addition to the choice of four optimization methods and their options, the choice between Iteration and Run. These are very different options for very different purposes, and it is important to understand their fundamental difference. First: what do they do?

The Iteration option

With the Iteration option you can fit the results of a single iteration (see the OptimizationIt.xls example workbook, discussed above). You do a normal single run with the Ersatz functions drawing random numbers on each iteration, and the number of iterations determined by the 'Number of Iterations' box in the Calculation panel (or by a ErSetItno function in the connected Excel workbook).
The difference is that after the Ersatz functions at each iteration have drawn their random number, they are switched off[12] and the optimization routine kicks in. The optimization will do whatever you have instructed it to, and only after it returns the ErOutput functions will pick up the values for that particular iteration. And next Ersatz will start the next iteration.
No questions are asked: the optimization is run and when it ends its result is summarily accepted. So you might want to put some ErOutput functions on various intermediate variables to check afterwards whether the optimization results make sense.
The Iteration option has an additional option: a check box allows limiting the optimization routine to the first iteration only. When checked, the optimization is run for the first iteration, and the value obtained is returned for all subsequent iterations. This will be useful only in rather specific circumstances. I needed it when running an uncertainty analysis of a micosimulation model, where each iteration is an individual

[11] Of course, theoretically you might have much more complicated constraints, but I haven't seen those in real applications.
[12] Switched off in the sense that they will return the same previously drawn random number for as long as the optimization routine is busy.

and the multiple runs option is used to simulate a population. In that case, population-wide uncertain parameters (so-called "second order uncertainty") are drawn at the start of the run, and then kept constant during the run (typically using an ErConditional function that checks whether ErIteration returns 1, see the BreastCaMicroUnc example for this set-up). Under these circumstances, this "First iteration only" option allows to run an optimisation after drawing the population-wide value, and make the result apply to the whole population.

The Run option
With the Run option you can fit the results of a run (see the OptimizationRun.xls example workbook, discussed in the Examples section below). Since optimization is an iterative procedure, this means to do multiple runs, with the number determined by the optimization algorithm.
Each run is a normal one, with the Ersatz functions drawing random numbers on each iteration, and the number of iterations determined by the 'Number of Iterations' box in the Calculation panel (or by a ErSetItno function in the connected Excel workbook). However, in order to get the run results into the loss function, you will need to use some of the Statistical functions (such as ErTotal, ErMean, etc., see the *Ersatz Function Overview* for the complete line-up), that return results at the end of a run.
Please note that choosing this option may mean a lengthy calculation. In addition, it is difficult to predict how lengthy, because it is the optimization algorithm that, depending on the options you set, determines how many runs will be done.

What to choose?
What to choose depends on what you need to do. The two example workbooks give typical applications of each option. In OptimizationIt.xls a line is fitted to the results of random draws from three ErBeta functions to obtain an uncertainty interval around the estimated annual trend. So this option is useful when the link between your random functions input and the model's outcome is not obtained by straight arithmetic, but involves such a fit procedure.[13]
Please note that, from the viewpoint of the optimization algorithm, this is not noisy optimization: while the results from the ErBeta functions differ between iterations, the optimization has to deal with a single set of results, and ends before the next set of random numbers is drawn.
In the OptimizationRun.xls workbook the outcome of a run, returned by an ErMean function, needs to be fitted to a target value. This would be a typical thing to do when you have developed a microsimulation model, where an iteration stands for an individual, and a run gives the outcome for a population by simulating a large number of individuals.

[13] This particular problem could also have been tackled using an Excel macro to call the Excel Solver after each iteration. However, as mentioned in the section on Excel macros, this is slow: the macro/Solver approach takes about 6 times longer than the solution with the Ersatz minimization functions. You don't have to take my word for it: the OptimizationIt workbook has a macro called 'solverfit' included. If you change the equations in B8-D8 to refer to C12 & C13 instead of B12 & 13, check the 'Execute macro after each iteration' box in the Options|Macros and type 'solverfit' into the associated text box, you can run this yourself. Make sure that in Visual Basic the Solver is referenced (Choose Tools|References, and check the SOLVER box). Please note that the optimization performed in the OptimizationRun workbook cannot be done with any Solver and macro combination.

Since the outcome of a microsimulation model, and of an Ersatz run in general, is typically random, this option will usually result in noisy optimization, with consequences for the kind of method to use, see below.

Four methods

This section discusses the four optimization algorithms implemented in Ersatz, including the options they take. The reader should be aware that options with the same name between methods do not mean the same thing. For example, 'Tolerance' is an option that determines how hard the algorithm will try to achieve the loss function's minimum, but that does not mean that the same value for this option results in the same level of precision.
A general option all methods share is the number of restarts. As discussed above, it can be useful to restart the optimization at the result of a previous one, in an attempt to avoid having ended-up in a local minimum. The default number of restarts is 0 (meaning no restarts), the maximum is 10 (but that is way over the top). If you want to use this feature, one or two restarts should do the trick (unless they don't, but then more probably won't help either).
The first two are deterministic, the second two stochastic optimization methods. All are multivariate (i.e. take a vector of input variables) but they can be used for univariate optimization (i.e. take a vector of length 1) as well. In the subsequent section I will discuss when to use what.

Quasi-Newton

This is an implementation of the Broyden-Fletcher-Goldfarb-Shanno version of the Quasi-Newton (QN) algorithm, using the finite differences method to calculate partial derivatives. It is a deterministic method, based on (Press, Teukolsky et al. 2007). Very briefly, this method is a generalization to multiple dimensions of a line minimization method that gathers information on the gradient of the loss function to make an informed guess on what to propose as the next vector of input values. If the loss function is reasonably approximated by a quadratic function, then the QN-algorithm can be very fast indeed. However, it requires the loss function to be deterministic, and is therefore quite unsuitable for noisy optimization.
Options:

1. Tolerance. Default value: 0.01. Lower values will force the algorithm to use more tries to get closer to the loss function minimum and will require more loss function evaluations (and therefore more run time), and vice versa.
2. Maximum tries. Default value: 500. This is the maximum number of loss function evaluations, and it set so high that it should never be reached. Putting in a much lower number will possibly reduce the number of function evaluations (and therefore run time), but when the algorithm is stopped by this criterion, it implies that its Tolerance criterion is not met.

Down-hill Simplex

The Down-hill Simplex (DS) method is a native multivariate optimization deterministic method due to Nelder & Mead. It uses a very simple algorithm to guess the next vector of input values, and, unlike the QN algorithm, does not use information on the gradient of the loss function. As a consequence, it usually requires more loss function evaluations and tends to take more time. But in my experience,

this depends very much on the actual optimization problem, and the difference with the QN method is often negligible and sometimes even in its favour.
DS begins by using the *N* start values to set up a simplex: *N*+1 alternative input sets, where all values equal the original one except one which equals the original start value times 1 + the value of the initial simplex option. It then evaluates all the new combinations, picks the best one, and tries to improve by modifying it. And so on. The behaviour of the DS simplex as it crawls through our hilly landscape has been compared with how an amoeba would squeeze itself through narrow holes, and the implementation in Ersatz is based on the same-named algorithm from (Press, Teukolsky et al. 2007).
Because it relies on loss function evaluations only, the DS method is less vulnerable to noisy loss functions than the QN method. While it does not completely go off the rails for moderately noisy loss functions, it does not really know when to stop: as mentioned above, when it gets near the minimum, it tends to keep trashing around and only stops when it exceeds its maximum number of tries.
This is because the standard stop criterion kicks in when the difference between the target and the model outcome value becomes sufficiently similar for the best and worst solutions. But this never happens when the model outcome keeps jumping around.
Therefore I've implemented an additional stopping criterion for use with noisy optimization: when the simplex becomes sufficiently small (meaning that the amoeba is not going anywhere), this will cause a stop as well.
Options:

1. Initial simplex. Default value: 0.1. This sets the starting simplex. Increasing this number increases the number of loss function evaluations needed, decreasing it increases the likelihood that the algorithm gets stuck in a local minimum near the starting values.
2. Maximum tries. Default value: 500. The same comments as for the QN maximum tries option apply.
3. Tolerance. Default value: 0.01. See the QN comments here as well.
4. Noisy optimization check box. Default value: unchecked. When checked, the additional stopping criterion described above becomes active.
5. Noise tolerance. Default value: 0.0001. Enabled only when the Noisy optimization check box is checked. Lower values will cause the additional stopping criterion to kick in later, and vice versa.

Simulated annealing

Simulated annealing (SA) is a stochastic optimization method, that has been successfully applied to very hard problems. However, most of the action has been in the field of combinatorial optimization (for example, the travelling sales person problem), and much less in the continuous function optimization we need. The algorithm implemented in Ersatz is based on a stochastic variant of the DS method described above (Press, Teukolsky et al. 2007).
SA derives its name from an analogy: annealing is the slow cooling of liquid metals which allows all atoms to find their place in a crystal structure, which represents the lowest possible energy state of the system (and also the configuration that is strongest). The method sports options such as 'starting temperature' and 'cooling factor' that play on this analogy.

While the deterministic DS algorithm always takes the down-hill direction, the SA variant will randomly go up-hill as well, with step sizes depending on the temperature at the time. The process starts at a high temperature, with big random jumps through the hilly landscape for a maximum number of tries or when it has reached the precision set by the tolerance level, whichever comes first (it will mostly be the maximum number of tries, especially when the temperature is still high). Then the cooling factor is applied to the temperature, and the process repeats. When the temperature hits 0, the algorithm will end. The algorithm remembers the best outcome ever, and as it ends will report that.
The maximum number of tries is per annealing level, so the maximum number of loss function evaluations equals the maximum number of tries times the number of annealing levels, the latter depending on the starting temperature and cooling factor, but this product will generally be very large. In other words, this method requires considerable time.
Options:

1. Initial simplex. Default value: 0.1. Inherited from the DS method, but the stochastic nature of the SA algorithm makes this parameter much less important.
2. Maximum tries. Default value: 500. See the discussion above, lowering this value will do much to speed up the optimization, but it will increase the probability to end up in a local minimum.
3. Starting temperature. Default value: 10,000. Higher values will cause the algorithm to make wilder random jumps, and will also cause, given a cooling factor, to increase the number of annealing levels. Both will increase calculation time, but decrease the probability of getting stuck in a local minimum.
4. Tolerance. Default value: 0.01. Inherited from the DS algorithm, but of less importance here because the stochastic nature will make the maximum number of tries the more likely stopping criterion.
5. Cooling factor. Default value: 0.1. A smaller cooling factor will increase the number of annealing levels, and therefore the total calculation time, but will decrease the probability of ending up in a local minimum. And of course vice versa.

Cross-entropy

The cross-entropy (CE) method, due to Rubinstein and Kroese, is a rather new kid on the block of stochastic optimization.(Rubinstein and Kroese 2008) It originated in rare event estimation, but soon proved useful for optimization as well. Like the SA method, most applications seem to be in the field of combinatorial optimization, but the method allows continuous function optimization as well.
For this latter application, a number of Normal distributions equal to the number of input values into the optimization are defined with means the starting values and sufficiently large standard deviations. From these Normal distributions, a sample of values is randomly drawn, and the loss function is evaluated for each of them. Then the proportion best results (determined by the Proportion elite option) is used to calculate a new mean and standard deviation, and with these another sample is drawn and evaluated. The process is repeated until it runs out of the maximum number of tries or the average coefficient of variation reaches a threshold minimum, defined by the variance threshold option.

It is a surprisingly simple algorithm (also to implement), with a high degree of face validity, and in my experience it works rather well. However, as with all stochastic optimization, computational costs are high. While the literature on the CE method by Rubinstein and Kroese makes much work of the small number of iterations needed to get to the minimum, it ignores that for each iteration the method needs a number of loss function evaluations equal to the sample size. Given that the sample size should not be small, this adds up.
Options:

1. Proportion elite. Default value: 0.1. This is the proportion of the sample size with the best results used to calculate the mean and standard deviation for the next iteration. A larger proportion is more conservative, leading to more iterations, but less probability to get stuck in a local minimum.
2. Coefficient of variation. Default value: 1.0. This determines the standard deviation of the Normal distributions (standard deviation = mean times coefficient of variation). A higher value will cause the algorithm to need more iterations, with a lower probability to return a local minimum.
3. Sample size. Default value: 150. This is the number of randomly drawn parameter vectors the algorithm will evaluate at each iteration. A larger number will need more loss function evaluations per iteration, but decrease the number of iterations needed.
4. Maximum tries. Default value: 100. Initially, when the standard deviations are still large, this is the most likely stopping criterion, but one hopes that in the end the variance threshold will be the one. A larger value increases the probability that the variance threshold will be the stopping criterion, but will cause a longer calculation time, and vice versa.
5. Variance threshold. Default value: 0.01. If the average coefficient of variation of the current set of Normal distributions is below this value, the algorithm is considered to have reached the minimum. A lower value will cause longer calculation time, but higher precision. And, as always, vice versa.

When to use what?

There are few hard and fast rules about when to use which method and with what options. And if there is any area in computation where the old economic saying of “there is no such thing as a free lunch” applies, it is in optimization. You can try to get better results, but always at a cost, and with marginal costs rising steeply. So a number of rules of thumb might come in handily.

- Where possible, use the deterministic algorithms (QN & DS). They use far less computation time than the stochastic ones.
- If you need to use the stochastic algorithms, try the CE method first. The SA algorithm seems to spend far more time thoroughly searching through areas where obviously no minimum is to be found (perhaps, you can never be sure, there is a bug in the implementation).
- Use the stochastic algorithms when there is evidence of local minima. Getting different outcomes from your deterministic algorithms for different starting values is strong evidence of the existence of local minima.
- The probability of local minima steeply increases with the dimensionality of the problem (which equals the length of the starting vector). Be aware, I’ve seen the deterministic algorithms getting nowhere on problems with a

  dimensionality as low as 6. So you might at least want to try one of the stochastic methods once if you have a problem of high dimensionality.
- Never use the QN method for noisy optimization. Use instead the DS method with the noisy optimization option checked, or else the stochastic methods (SA & CE). Or use a combination: first the CE method to get in the neighbourhood of the global minimum, and next the DS method
- With the 'Run' option you are most likely to deal with noisy optimization, with the 'Iteration' option you are not.
- Experiment with the tolerance parameters. If the optimization method is trying to achieve a precision way beyond what is useful for your application, a reduction in the tolerance option will greatly speed up the process.

Basically, what I'm trying to say here is that you should not rely on a single method and its default parameters for all problems. Optimization benefits by tinkering: try a different method, or change the options, or both. The four methods and their options offered by Ersatz give you ample opportunity.

Documentation

Examples

Introduction

The Ersatz installation comes with a number of example Excel workbooks. These come in two categories: the purpose of the first one is give an implementation example of each Ersatz function; the purpose of the second category is to illustrate some more general techniques, in particular multiple runs and microsimulation. These workbooks use a cost-effectiveness analysis loosely based on the example of trastuzumab (Herceptin™) for early breast cancer. In addition there is a workbook illustrating the use of Ersatz for probabilistic bias quantification.
All example workbooks are accessible through the Ersatz entry in the Windows Start menu, and through the Ersatz 'Help|Example spreadsheets' menu. Where they are located on your harddisk is difficult to say: it depends on your Windows version, your privileges on the PC, and possible changes to the proposed default directories that you may have made during installation. Details can be found in the section Installation issues of the Technical appendix of this guide.
Please note that the example workbooks are not protected: any changes that you make can be saved. If you want to preserve the original example workbooks but also experiment with them, you should first make copies of them to experiment with, for example by using 'Save as' to copy them to your standard Excel workbook directory. Of course you can always revert to the original example workbooks by re-installing Ersatz.
Below is a brief description of each of the example workbooks. All example workbooks can be run, and contain, where appropriate, ErOutput functions to inspect the results in Ersatz. The examples should be studied with the *Ersatz Function Overview* document at hand.

FunctionLineup

This workbook contains examples of most of the functions that Ersatz adds to Excel. Exceptions are mostly rather more complex functions such as the component functions in the special ComponentFunctions workbook.
The FunctionLineup workbook has several worksheets:

- StandardDistributions contains the normal random distribution functions (including the Normal).
- ErEmpirical is devoted to the two modes of this function: discrete and continuous.
- ErSurvival gives examples of the ErSurvival function (including the effect of having survived to a certain age) and the ErSurvival2 function, which allows using a single uniform random draw to get survival durations from different survival data. See also the Survival123 and BreastCaMicro example workbooks for the use of these functions.
- SpecialFunctions contains examples of the special (i.e. non-random) functions, such as ErIteration.

To run this workbook, start Ersatz, set the number of iterations and press Calculate.

Output2Workbook

In most cases, outputs from your workbook are picked-up by ErOutput functions, and transferred and displayed in the Ersatz executable. However, in some circumstances it is more convenient or even necessary to get the outputs into the workbook itself. This example workbook illustrates the use of functions that allow just that. There are two worksheets:

- StatisticalFunctions implements the Ersatz statistical functions that return summary statistics at the end of a run. Basically, these are the outputs that you get in the Ersatz Summary outputs tab. Not present here is ErCorrelation, which can be found, among others, in the ComponentFunctions workbook.
- The DataFunctions worksheet implements the ErData and ErDataArray functions. The former gives the value of an output at a specific iteration, the latter at a range of iterations, starting at the first. Note that the ErDataArray function returns #NUM! before the workbook is run.
- There is also an ErRunDataArray function for use with the multiple run option. Please consult the BreastCaMarkovMC workbook for an example implementation.

Please note that these functions are not available in the trial version of Ersatz, and that this workbook is not included in the trial download.

ComponentFunctions

As described in the *Ersatz Function Overview* document, the Ersatz component functions implement situations where several random numbers need to be drawn that require some coordination. An example is correlated random draws from various distributions. Ersatz implements this using component functions that are linked to a corresponding master function.
Setting up such a system is decidedly more complex than the use of the standard random functions, and therefore this workbook contains a separate worksheet for each of the six component functions. When you first start using these functions, you may want to copy the example into your own spreadsheet, and work from there.
A warning: when implementing a set of component functions and their master function, make sure the checkbox 'Show mean values while not running' in the Excel box on the Ersatz Settings tab is checked. If not, you may get rather puzzling #NUM! errors.

- The Nonparametric worksheet contains an example with 10 records, each of three fields. The data is hypothetical, but could be the results from 10 respondents that assessed disability weights for three diseases. The outcome of interest is the average disability weight (and its variance) for each disease.
- The Randomisation worksheet example is an implementation of a randomisation test to see whether the observed downward trend in prescriptions for anti-hypertensives could be due to stochastic variation (outcome: extremely unlikely).
- The Multinomial distribution is a generalisation of the Binomial to multiple categories. Each marginal distribution is Binomial, but the sum over the categories is always equal to the total N. An essential distribution when modelling, for example, decision trees with nodes that have more than two branches.

- The Dirichlet distribution is a generalisation of the Beta to multiple categories. Each marginal distribution is Beta, but the sum over the categories is always equal to 1. An essential distribution when modelling, for example, prevalences or transition probabilities that have more than two categories.
- The CorrNormal worksheet gives two examples, one using a covariance and one using the equivalent correlation matrix. Note that changing the input correlations can easily lead to an invalid covariance/correlation matrix (just change the cov(N1,N3) to -0.2): in that case Ersatz will not run and display a fatal error. See the section on Valid correlation matrix in the Random numbers topic for more information.
  The output correlations are examples of the ErCorrelation statistical function, that will give a meaningful result after a completed run only.
- The RankCorr worksheet shows how to obtain rank correlated random draws from arbitrary functions, in this case a Gamma, a Weibull, and a Poisson. The remarks in the previous dot point on valid correlation matrices and the ErCorrelation function apply here as well.

To run this workbook, start Ersatz, set the number of iterations and press Calculate.

OptimizationIt

This workbook illustrates the Iteration option of the Ersatz Optimization algorithms. Please refer to the section on Optimization above for a description of this example.

OptimizationRun

This workbook illustrates the Run option of the Ersatz Optimization algorithms (refer to the section on Optimization above for a description of the Iteration and Run options and other concepts mentioned here). This is a very simple optimization problem (in fact, it is not an optimization problem at all because it is easy to derive its solution analytically, see the E column), but it serves to illustrate the Run option, which will almost always, as in this case, imply noisy optimization.
The task is to find the two parameters for a Beta distribution such that it will return a specific mean (set to 0.77 in the example, cell B17) while the sum of the two also has a specific value (set to 123 in the example, cell B18). This can be interpreted as obtaining parameters such that the Beta distribution returns a specific prevalence, given a population size (see the section on Good Modelling Practice if you don't understand this).
The ErMinimize function is entered as an array formula in cells B8-9, taking as its starting values the range C8-9. Note that the ErMinimize function works on log-transformed values for the ErBeta function because the Beta distribution requires both parameters to be > 0, see the subsection on Constrained optimization in the Optimization section above.
The ErBeta function itself is in cell B11, with the exponentiated parameters from the ErMinimize function in cells B12-13. The target prevalence and population size are in cells B17-18. The realized mean value from the draws of the ErBeta function is returned in cell C17, using an ErMean function. The realized population size is simply the sum of the two Beta distribution parameters (cell C18).
Finally, the loss function is in cell B20, consisting of squared differences functions for both prevalence and population size. Note that the squared difference for the prevalence is multiplied with twice the population size of cell B18: this is to scale the

parameters for the loss function to similar size. The ErMinimizeResult function picks up the value of the loss function in cell B22.
To run this example, check the Optimization box in the Ersatz section, type the ErMinimize function name (’fit2’) in the designated box, choose the run option and one of the optimization methods, and press Calculate. Warning: depending on method and options, this may take anything between a minute and many hours.
This workbook illustrates the following issues for the use of the Run option (and for noisy optimization in general):

1. Do not use the Quasi-Newton method for noisy optimization. If you try this, you will see it goes completely off the rails, and gives up after a short while, returning a clearly useless result.
2. The Down-hill Simplex method does a lot better: it moves towards the minimum, but unless you check the ‘Noisy optimization’ box, it will keep thrashing around that minimum until it runs out of tries. If you do check that box, it will return quickly with a good result. Please note that this method only works when the loss function is only moderately noisy, but in that case it is the most efficient choice.
3. The stochastic methods (Simulated Annealing and Cross-Entropy) both can handle this case of noisy optimization. But both, and in particular the Simulated Annealing method, take a lot of time.

CorrMultivariate

This workbook gives example implementations of the correlated multinomial and Dirichlet distributions. In addition, it shows how to obtain a similar effect for multivariate Normal distributions.
As with the Ersatz component functions, correlated multivariate distributions require coordination between distributions, and a similar set up has been implemented for these correlated multivariate distributions. The difference is that now three kinds of functions collaborate: a single master function and input and output functions for each of the correlated distributions.
Again as with the component functions, setting up such a system is decidedly more complex than the use of the standard random functions, and therefore this workbook contains a separate worksheet for each distribution. When you first start using these functions, you may want to copy the example into your own spreadsheet, and work from there. Also consult the *Ersatz Function Overview* document for implementation details.

- The CorrMultinomial worksheet contains an example of three multinomial distributions, each with five categories. There is also a correlation matrix, all data is hypothetical. The master function ErMultinomialCorr picks up the correlation matrix and the number of correlated distributions as parameters. Linked to the master function are three ErMultinomialCorrIn input functions, each picking up the corresponding input numbers and its distribution number as parameters. Finally, there are three ErMultinomialCorrOut functions, each linked to its corresponding ErMultinomialCorrIn function, and taking its distribution number as the second parameter. These ErMultinomialCorrOut functions are entered as ‘array functions’, see the note in the *Ersatz Function Overview* document if you do not know what Excel array functions are.

- The CorrDirichlet worksheet implements the same data as the CorrMultinomial worksheet, and the set up is, apart from the function names, identical.
- When the multivariate distribution is set up using a number of marginal distributions and a correlation matrix, as with the multivariate Normal, a similar effect of two correlated multivariate distributions can be achieved by specifying a combined appropriate correlation matrix for all distributions involved. The CorrNormal worksheet shows how to do this.
  Please note that the requirement for the correlation matrix to be valid (positive semi-definite) puts strong restrictions on the values in the correlation matrix. You should use the Ersatz option to check validity and, if needed, replace it with the valid matrix Ersatz suggests (Options|Correlation).

ConditionalStore

Ersatz has functions that allow storing workbook values in memory, and retrieving the values at a later point. This workbook contains an example implementation of the ErCondStoreArray and ErCondRetrieveArray functions, which allow storing and retrieving arrays of numbers, prompted by a Boolean parameter.
The example is based on a scenario where a number of interventions for a particular health problem are available, each having an effect size and cost. But after the first intervention is implemented, the second one will have a smaller effect because part of the problem is already resolved by the first intervention. This is modelled by making the potential impact fractions (PIFs) multiplicative.
To run this model, you should check the ‘Multiple runs’ options, and set the number of runs to 100. Please note that this example also uses the ErDataArray and ErRunDataArray functions which are not available in the trial version and that this workbook is not included in the trial download.

BreastCaMarkovSC

This workbook implements a Markov model of breast cancer incidence and survival, with an evaluation of trastuzumab (Herceptin™) which improves survival of a specific sub-group of breast cancer patients: those with her2-positive tumours.
Survival is based on a South Australian follow-up study, I fitted a lognormal distribution with a proportion cured to this data (see the Data & Results worksheet).
From the modelled survival an annual excess mortality since year of incidence is calculated. The effect size of trastuzumab (0.55, cell L4) is applied to this excess mortality for 5 years in full (based on a meta-analysis), after that it is assumed to attenuate (see column O).
Incidence is the number of breast cancer cases in Australia in 2003 by age times 0.2: the proportion of her2-positive cases. The model is single cohort (that’s what the SC stands for): you select an age group (cell F19), and the model calculates the life years lived, costs, and incremental costs per life year gained (more generally known as ‘incremental cost-effectiveness ratio’, or ICER) for that age group. Note the use of the Excel Index function (in worksheet Base, column B) to pick up the correct ‘all other causes’ mortality, given the age at incidence. Also note the steeply increasing ICER by age.
Uncertainty is implemented as an ErRelativeRisk function on the effect size (cell L4) and an ErGamma function for the costs of disseminated/terminal disease (cell L9).

Cost of primary treatment other than trastuzumab is ignored, because assumed to be the same for both baseline and intervention.
To run this workbook, start Ersatz, set the number of iterations (say 2000), select an age group (cell F19), and press Calculate.

BreastCaMarkovMC

The BreastCaMarkovSC workbook calculates the cost-effectiveness for the various incident age cohorts, but as a rule the question is not what the cost-effectiveness is for a specific age group, but for the whole population. You could use the BreastCaMarkovSC workbook to calculate a central estimate by totting up the results of each age group, but doing a proper uncertainty analysis that way would be a real hassle.
The BreastCaMarkovMC workbook (MC for multiple cohorts) solves the problem by using the Ersatz multiple run option, see the section on Multiple runs above. This workbook is largely identical to the BreastCaMarkovSC workbook, with the following changes:

1. Selecting the age group (cell F19) is no longer done manually, but by an ErIteration function. When the number of iterations is set to the number of age groups (=14 in this case), each run will cycle through the age groups one by one.
2. Cell F21 contains an ErSetItno function, with the number of age groups (cell F17) as parameter. This function sets the number of iterations Ersatz will do to the correct one.
3. Cell F23 contains a formula that will return TRUE if the selected age group equals 1, and FALSE otherwise.
4. This boolean is used to make sure that for all ages the same randomly drawn effect size and disease costs is used. The ErRelativeRisk (cell L4) and ErGamma (cell L9) functions are now embedded in an ErConditional function which takes the boolean of cell F23 as its first argument. When TRUE ErConditional returns the currently drawn random value, when FALSE it returns the same value as it did at the previous iteration.
5. In cells G27..29 an ErCondStoreArray and corresponding ErCondRetrieveArray function are used to sum over ages. Note that the ErCondRetrieveArray function returns #NUM! until the model is run. These functions sum up the difference in life years and costs over the age groups. Cell G30 calculates the ICER over all age groups.
6. Cells H28, 29, & 31 now have ErRunOutput functions, the special output functions for multiple runs that store the last input value of each run.
7. Similarly, the ErSensInput functions of cells P4 and P9 have been replaced by their multiple run equivalent ErRunSensInput.

I've described the differences between the BreastCaMarkovSC and -MC workbooks in detail to drive home the point that you can start by developing a single cohort model, and then easily convert this to a multiple cohort model using the Ersatz functions mentioned here and in the topic on Multiple runs.
To run this workbook, start Ersatz, check the 'Multiple runs' and 'Use multiple runs input/output functions' boxes on the Ersatz section of the Settings tab, set the number of runs (say 2000), and press Calculate. It will take a bit of time (1 minute, 20 seconds on my laptop).

Survival123

This workbook implements examples that are discussed in the Microsimulation section of this guide. Briefly, Survival123 compares three ways of modelling empirical (as opposed to parametric) survival, with the Ersatz ErSurvival function as the preferred way.
To run this workbook, start Ersatz, set the number of iterations and press Calculate.

BreastCaMicro

This workbook is also mentioned in the section on Microsimulation, in particular on reducing randomness. It implements the same model of breast cancer incidence and survival as the two Markov models discussed above, but now uses the technique of microsimulation.
Incidence is by age, and when a woman becomes incident first an age at death from all other causes is drawn, conditional on having survived until the age at incidence (using ErSurvival). Next a breast cancer survival time is drawn (again using ErSurvival), using the same fitted lognormal excess mortality as in the Markov models. The age and cause of death of the woman is then determined by the one that comes first: all other causes or breast cancer.
Also the same intervention is implemented that models the decrease in excess mortality due to trastuzumab (Herceptin™), with a different survival curve and less breast cancer deaths as a result.
The difference between the BreastCa1 and BreastCa2 worksheets is that in the latter a randomness reduction technique is illustrated, see the corresponding section on 'Combating randomness in microsimulation' above.
The workbook also implements the ErSetItno function (BreastCa1, H19). This function sets the number of iterations Ersatz will do equal to the number of women in the microsimulation. Remember: in microsimulation each iteration stands for an individual.

BreastCaMicroUnc

This workbook implements the same model as the Breastca2 worksheet in the BreastCaMicro workbook. The difference is that it is now set up to run an uncertainty analysis for this model. As with the BreastCaMarkovMC workbook you need to use the Multiple runs option for at least 1000 runs.
The difference with the Breastca2 worksheet is that the effect size and cost of disseminated/terminal disease now both have random functions (the same as in the BreastCaMarkovMC workbook).
To run this example, start Ersatz, check the 'Multiple runs' and 'Use multiple runs input/output functions' boxes on the Ersatz section of the Settings tab, set the number of runs (say 2000), and press Calculate. Warning: this will take time (1 hour, 20 minutes on my laptop).

ResinBias

This workbook implements the calculations described in a draft paper (Barendregt and Blakely Draft). The work was done in response to earlier work on probabilistic bias quantification by Fox, Lash, and Greenland (Fox, Lash et al. 2005). They used data from a case control study on the risk of lung cancer from exposure to resin to illustrate the method of probabilistic bias quantification, and the same data is used in this example.

Using assumptions on sensitivity and specificity, a bias corrected dataset is calculated from the observed data. From this bias corrected dataset two-by-two tables are calculated for cases and controls. We assumed these two-by-two tables to have correlated Dirichlet distributions, scaling the numbers such that the standard deviations of the sensitivities and specificities are similar to the ones obtained by Fox et al.
In a final step we calculate the probabilistic bias corrected dataset, and combine the uncertainty of the bias correction with the sampling uncertainty to a single uncertainty interval. Like Fox et al we do the calculations for non-differential and differential misclassification. By using the correlated Dirichlet distribution to produce the probabilistic bias corrected datasets, we obtain much narrower combined uncertainty intervals than Fox et al.

Error messages

Fatal errors

The following error messages are fatal in the sense that Ersatz will not run, and no output will be produced. I've grouped individual error messages in, hopefully, useful categories where appropriate.

1. "There are duplicate XX function names in the connected workbook. Run aborted." and:
   "There is an empty string as an XX function name in the connected workbook. Run aborted."

   Ersatz has a number of functions that require you to name them. Moreover, Ersatz puts requirements on these names. The first is that within each function category the name should be unique, the second that empty strings as a name do not qualify. If you transgress these requirements you will get these error messages.
   In these error messages XX can stand for one of the following: Eroutput, ErRunoutput, ErMultinomial, ErNonparam, ErDirichlet, ErSensinput, ErRunSensinput, ErCorrNormal, or ErRankCorr.
   The remedy is simple: make sure your functions have different names (first case) and none of them is an empty string (second case).
   Please note that when more than one Excel workbook is open, Ersatz will, contrary to appearances (only the name of one of them will be showing in the title bar), be connected to all[14]. If you have, for example, two versions of the same workbook open simultaneously, you will get the 'duplicate names' error when they have the same named functions. You will need to quit the other workbook(s) in order to run Ersatz.

2. "An error occurred while executing an Excel macro".

   This error occurs when Excel runs into a problem while executing a macro that you call from Ersatz. The remedy is to make sure that your Excel macro is functioning correctly. This may sound like a cheap answer, but Ersatz really cannot know what a macro you want it to call is up to, so this error is outside its remit.

3. "An unknown error occurred".

   This error can occur when some Ersatz functions show #NUM! or #VALUE! errors before you click the Calculation button. In that case the remedy is simple: make sure all errors are eliminated before running the workbook. See the section on Trouble shooting below.
   If all functions are fine and you still get this error, it is potentially serious. It may occur because Excel has become unstable, or Windows itself has. The first remedy is to quit Ersatz and Excel and try again. If that doesn't work,

[14] This is not a design objective, but there seems to be no way to avoid this Excel behaviour.

reboot your PC. If the problem persists, contact EpiGear (info@epigear.com).

4. "Run cancelled by user".

   Not really an error message because you clicked the 'Cancel' button during a run.

5. "ErFixed reports a number index < 1".

   Not sure why this error message is here, because it seems more appropriate in the non-fatal category. Anyway, you've fed an ErFixed function a second parameter that is < 1, which is a no-no.

6. "ErTruncate does not have an eligible embedded Ersatz random function. Run aborted".

   Not all Ersatz functions can be subject of an ErTruncate function, and this error occurs when you've picked one outside the list. See the *Ersatz Function Overview* topic on the ErTruncate function to see which functions are permitted.

7. "An ErCorrNormal function reports an invalid correlation matrix. Run aborted".
   "An ErRankCorr function reports an invalid correlation matrix. Run aborted".

   The ErCorrNormal and the ErRankCorr functions require either a correlation or a covariance (ErCorrNormal only) matrix. These matrices need to be positive (semi) definite, if they are not these error messages occur. See the topic on correlated random draws in this Guide to remedy this.

8. "Run-time Er(Run)output or Er(Run)Sensinput function is not accounted for. Run aborted".

   One or more of the mentioned functions shows a non-numeric outcome (like #VALUE!) when Ersatz is not running, but gets a valid numeric outcome as soon it does. Since Excel does not call a function when its result is not valid, Ersatz does not find it when it takes stock before the run, but then unexpectedly finds it during the run. Remedy: see which of your functions has this problem, and fix it.

9. "An ErRankCorrCom function does not have an eligible embedded Ersatz random function. Run aborted".

   Most but not all of the Ersatz random functions can be used with the ErRankCorrCom function. See the *Ersatz Function Overview* topic on the ErRankCorrCom function to see which functions are permitted.

10. "This is the workshop version of Ersatz, which has a maximum of 5 input functions. Run aborted".

Students in the cost-effectiveness workshop at the School of Population Health, University of Queensland, get a free but limited copy of Ersatz. If you exceed the function number limit, this message will be shown. Remedy: either limit the number of input functions, or buy the fully licensed version (www.epigear.com).

Non-fatal errors

In addition to fatal errors there are also non-fatal ones: Ersatz will run, and output will be produced, but probably not for all output variables: some or all of them will show up as “NaN” in the summary output. “NaN” stands for “not a number”, and it implies that either an input function has been fed a parameter value outside its range, or that your calculations made Excel throw a fit, such as divide by zero. The error messages will be displayed in the Message window (choose View|Messages), and Ersatz will warn the user when there are any. The messages specify the iteration and the function that encountered a problem.

Trouble shooting

Sometimes the combination of Ersatz and Excel will not behave as the user expects, with a frustrated user as a consequence. In this section I try to pre-empt some of the situations where this might occur by explaining what is going on and offering a solution. Most of this is based on user experiences in my own work environment (including myself). I am happy to hear of any other potential conundrums: if you have any, please email them to info@epigear.com and I will either try to solve the issue or add them to this list.

1. The calculation button is disabled.

   Ersatz is probably not connected to a spreadsheet. If this is the case, its title bar will read 'Not connected'. This will occur when Excel is not running, or when you had more than one instance of Excel running and you quit one of them. If Excel is not running, start it and open the spreadsheet you want to work with. Then choose 'File|Connect to Excel' in Ersatz. If that doesn't work, you may have to quit Ersatz and Excel and start anew.
   Please note that if you have more than one instance of Excel running, and you quit one, Ersatz will become disconnected. Establish a new connection by choosing 'File|Connect to Excel'.

2. The Ersatz functions are not recognised by Excel (they show the #NAME? error).

   If you see this error on just a single or a few Ersatz functions, while other Ersatz functions are fine, you most likely made a typo in the function name. If you are unsure how a particular function name is spelled, use the Excel function wizard: all Ersatz functions are available in the category 'Ersatz'.
   If all Ersatz functions show this error, and you are confident that they are spelled correctly, most likely Ersatz is not properly (or not at all) installed. Check whether the Ersatz add-in is listed and active (Excel 2003 and earlier: Tools|Add-ins; Excel 2007: Office Button|Excel Options|Add-ins|Go). Make sure the add-in is listed and the checkbox next to its name is checked.
   If the add-in is not listed, you can try to use the 'Browse' button of the Excel Add-in Manager to add it to the list, but most likely something went wrong during installation (e.g. you had Excel running while installing Ersatz). Remedy: quit Excel, and install Ersatz.

3. The Ersatz functions are recognised by Excel, but not correctly (they show the #VALUE! error).

   If you see this error on just a single or a few Ersatz functions, while other Ersatz functions are fine, you most likely made an error in the number of parameters the function takes. If you are unsure about the parameters of a particular function, use the Excel function wizard: all Ersatz functions are available in the category 'Ersatz'.
   Occasionally a workbook that was previously OK will all of a sudden return these #VALUE! errors for all Ersatz functions. It seems that for some reason on these occasions Excel unregisters the Ersatz functions. You can check for

this condition by using the Excel function wizard: if it lists the Ersatz functions, but without their parameters (e.g. ‘ErNormal()’), this is what happened.
Remedy: uncheck the Ersatz add-in in the Excel Add-in Manager (Excel 2003 and earlier: Tools|Add-ins; Excel 2007: Office Button|Excel Options|Add-ins|Go) and click OK, and then open the Add-in Manager again check it again: this will force Excel to register the Ersatz functions. If you then force a recalulation of the workbook by entering a cell containing an Ersatz function for editing and then hit ‘return’, the errors should go away.

4. An Ersatz function returns #NUM!.

   You most likely made an error in the value of the parameters the function takes. For example, if you enter ‘ErNormal(4,-1)’ you will get this error because the standard deviation of the Normal distribution needs to be > 0. If you are unsure about the values the parameters of a particular function can take, use the Excel function wizard: all Ersatz functions are available in the category ‘Ersatz’, and help is given on the parameter values they can take. Or look up the function in the *Ersatz Function Overview*.

5. Ersatz runs but gives no output.

   There are a number of situations where this will occur:
   a. You forgot to put in ErOutput functions. The remedy is obvious.
   b. You did put in ErOutput functions, but you do not feed them a valid output value. For example, if you link the second parameter of the ErOutput function to a cell containing text, this will happen. The ErOutput function will, by the way, return #VALUE!.
      Remedy: make sure you link the ErOutput functions to valid numerical values.
   c. You did put in ErOutput functions, but you have checked the ‘Multiple Runs’ and ‘Use multiple run in/output functions’ checkboxes. When you check these options, Ersatz expects ErRunOutput instead of ErOutput functions.
      Remedy: replace ErOutput with ErRunOutput functions.
   d. You did put in ErRunOutput functions, but did not check the ‘Use multiple run in/output functions’ checkbox.
      Remedy: if you need ‘Multiple Runs’, check the the ‘Use multiple run in/output functions’ checkbox. Otherwise replace the ErRunOutput by ErOutput functions.

6. Ersatz used to run fine, but now all of a sudden requires me to apply for a release code.

   This may happen when you change the PC configuration, for example by installing a new motherboard. Remedy: apply for a new release code.
   This issue may also occur with illegal Windows software, see Known Issues.

Known issues

No software is without bugs, and that is undoubtedly true for Ersatz (and for Excel, for that matter). Things can go wrong, and once they have done so, the software may have become unstable and start to produce error messages whatever you do. In such cases it is often advisable to quit the software altogether and start anew.
In particular, when Ersatz has given error messages like 'Access violation', 'Invalid floating point operation', or 'Range check error', things have seriously gone wrong and a restart is often required. If the problem is reproducible (i.e. if it occurs predictably with some workbook and some calculation), I am very interested in receiving a description of the problem (if it is not one listed below), if possible with a copy of the offending workbook. Please send email to info@epigear.com.

1. The Partial Rank Correlation multivariate sensitivity option is not very robust and can produce unhelpful error messages such as 'Invalid floating point operation'. At this point I don't have a clue why this is (because I have not looked into the problem). The Pearson, Spearman, and Kendall's tau options are robust. Currently not very high on the priority list.

2. Non-English Windows versions can cause problems. Some local versions of Windows use a comma as decimal separator, and consequently expect a semicolon instead of a comma as the separator of the parameters in the Ersatz functions. No other issues are known at this point in time.

3. The copy-protection scheme of Ersatz gets thrown off (or, if you like, works overtime) on at least one non-official (read: illegal) Windows version. Ersatz will repeatedly think it is running on a new computer, and require you to obtain a release code before it will run. Of course EpiGear will not provide you with new release codes if this is your problem. And of course the remedy is clear: get yourself a legal copy of Windows.

Technical appendix

Installation issues

A rant that can safely be skipped
Microsoft seems to delight in making things unnecessarily complex, and over the years it has got pretty good at it. The general strategy seems to be to keep users as much as possible in the dark about what is actually going on on their PCs.
Hence, for instance, such unhelpful default settings in Windows Explorer as suppressing file extensions (with the consequence that you cannot distinguish between an Access database and its lock file, for example) and showing large icons (which take up maximum screen space while providing minimal information). Why keeping users ignorant and thus helpless would be a good thing I'll gladly leave to conspiracy theorists.
One of the things Microsoft has come up with is that files should go in different places, depending on their kind. So executables go in C:\Program Files, while related data files and such go somewhere else, usually buried deeply in some C:\Documents and Settings sub folder. It would seem much simpler and transparent to keep related files in the same place.
With Windows Vista new heights of obfuscation have been reached. Vista actually refuses to show whole directories, even when you have administrator rights. What's more, while in Windows XP and earlier the ideas of Microsoft about where files should go could safely be ignored, Vista and later enforce these rules and software that ignores them will not run correctly or at all. So whether I like it or not (and I don't), I'm forced to play along with Microsoft's game of file hide and seek.

Where are your files?
After I've had my little rant, let's get down to business. Installation is usually painless, the most important thing to remember is that Excel should not be running when you install Ersatz. If Excel is running, you will get an error message during installation that the add-in could not be installed. Remedy: from now on read the messages installation programs display, quit Excel and try again.
Once you've installed Ersatz, where are the files? This depends. For one thing, the installation program gives you a choice. But even if you go with the defaults, it still depends, in particular on the user rights you have on the PC. If you have administrator rights, the Ersatz executable and xll add-in will go into Program Files, while the example spreadsheets, temporary data files, and such go into a subdirectory of Documents and Setting\All Users\Ersatz.
If you do not have the rights to install software on your PC, you nevertheless can install Ersatz. Ersatz will then install all files under Documents and Setting\Your User Name\Ersatz. In this case you may get the message during installation that "You may have to install the add-in manually". In practice this usually turns out to be not the case: if you open one of the example spreadsheets and do not get #NAME? errors, you're fine. If you do get those errors, you will have to go to Tools|Add-ins (Excel 2003 and earlier) or Office Button|Excel Options|Add-ins|Go (Excel 2007) and browse where the file 'Ersatzdll.xll' is located (now you know why it is important to know where your files are).

To minimise the pain of finding the Ersatz files, all supplementary files (documentation and examples) are accessible through the Start|Programs|Ersatz menu that is created by the installation program, and from Ersatz's Help menu.

A note to IT personnel
As described above, users can install Ersatz even when they have no administrator rights. What is more, when you install Ersatz as administrator, other users may not be able to use it. The reason is that the installation program writes to the Windows registry key HKEY_LOCAL_USER, not to HKEY_LOCAL_MACHINE. There are good reasons for that (which need not be elaborated here), but the upshot is that IT should leave it to the users themselves to install the software, even on multiple user machines in computer labs.

An unhelpful Excel offer
Sometimes Excel will offer to move the add-in file 'Ersatzdll.xll' to its 'AddIns' folder. Never, repeat never, accept this. If you do, it will cause all kinds of mischief, in particular after upgrading to a newer version of Ersatz. It is one of those features of Excel where you really wonder what they were thinking at Microsoft.
If you have already accepted the offer, here are the steps to undo the harm:

1. Go to the AddIns directory (you can find where it is by going through the steps outlined in the previous section to manually add the add-in: when you click the Browse button, Excel will start in the AddIn folder).
2. Delete the Ersatzdll.xll file. You cannot do so when Excel is running, so you have to quit Excel first.
3. Start Excel. It will complain that it cannot find the Ersatz add-in, and ask you whether it should be removed from its list. Click 'yes' to that.
4. Re-install Ersatz.

After having gone through this, you will probably remember to decline Excel's offer in the future ☺.

Software

Ersatz was developed in Object Pascal, using Borland's Delphi 7 programming environment for the 32-bit version, and Embarcadero's Delphi XE2 for the 64-bit version. While many people seem to think C++ is essential for this kind of project, Object Pascal is actually just as powerful, easier to use, and beats all C++ compilers by its blazing speed.
Writing xll add-ins for Excel has been described as a black art. While I would hesitate to call it that, it certainly requires lots of stamina. The main reasons are that the Microsoft Excel xll software development kits are rather short on detail, and that Excel proves to be a wilful environment to program for, with a tendency to either crash or sulk if anything is wrong rather than be explicit about it.
Fortunately, where Microsoft left a lot to be desired, some people have stepped in. I am indebted to David Bolton whose article "Writing (Non Com) Excel Add-ins in Delphi" (which originally appeared in Delphi Magazine but is now floating around on the Web), despite some strange errors, gave me the necessary heads up in my first steps on this road. An invaluable resource is also Steve Dalton's "Financial applications using Excel add-in development in C/C++" (Dalton 2007). But as the title suggests, you need to be able to understand C(++) code to get the most out of this book.

Excel 2007 is a major upgrade from previous versions. Among other things, it sports a 'big grid', i.e. many more rows and columns than previous versions. This required a change in its basic data type, the XLOPER, to the XLOPER12. These special data types have the flexibility to describe the different things Excel works with: text, numbers, ranges, etc. Excel 2007 is largely backward compatible with add-ins written for earlier versions, but Ersatz actively supports the new features of Excel 2007 and higher by implementing a dual interface, and presenting the applicable one depending on which version of Excel is running. A not yet supported feature of Excel 2007 and higher is multi-threaded recalculation, which is planned for Ersatz version 2.0.
Few people will at this point have the 64-bit version of Excel 2010 and later installed: the default installation of Office 2010 and later is still 32-bit. Starting with version 1.3, Ersatz is compatible with both 32- and 64-bit Excel. The installation program contains two versions of the add-in, and installs the correct one depending on the Excel version. The Ersatz executable is still 32-bit.
The Ersatz installation program was written using Inno Setup 5 (www.jrsoftware.org). This is one of those amazing things: it is freeware, but beats commercial installation software such as InstallShield hands down on features as ease of use, power, and flexibility. Recommended.

Statistical and other scientific sources

There is a considerable number of scientific routines implemented in Ersatz. There are of course the random number generators and the routines to produce random deviates from specific distributions, but there are also lots of less visible supporting functions to do sorting, indexing, matrix manipulations such as the Cholesky decomposition, etc. There is a large and expanding literature on these matters. My general strategy is to find a suitable publication, study the math and code (mostly Fortran or C) or pseudo-code, and implement an Object Pascal version of it.
Most of the supporting functions are based on the venerable Numerical Recipes (www.nr.com) (Press, Teukolsky et al. 1992; Press, Teukolsky et al. 2007). The series, there are several editions, is one of those rare things: books on highly technical and mathematical subjects that are also a good read.
Ersatz's random number generators are all but one based on the Ultimate Random Number Suite as programmed by Peter N Roth and Stefan Hoffmeister (which seems to be no longer available). The one exception is the Mersenne twister, which is based on the algorithm developed by Makoto Matsumoto and Takuji Nishimura (see www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html).
The functions to obtain random deviates from specific distributions are from a number of sources (and I have indicated in the *Ersatz Function Overview* for each function what the source is). The canonical book here is the one by Luc Devroye (http://cg.scs.carleton.ca/~luc/rnbookindex.html) (Devroye 1986), but I have also used Numerical Recipes (Press, Teukolsky et al. 1992), Gentle (Gentle 2003), and Law & Kelton (Law and Kelton 2000), and articles in specialised journals. Special mention deserves Wikipedia, whose statistical pages have developed into a very useful resource (see: http://en.wikipedia.org/wiki/Statistics).

Version history

This is a running list of the Ersatz versions, with the latest version first. You can check which version you are running by choosing Help|About in the Ersatz executable, or typing '=ErVersion()' in an Excel spreadsheet cell[15].

Ersatz version 1.35

February 2, 2017
A minor update: improvement in the user interface of the Conditional firing option.

Ersatz version 1.34

July 16, 2016
A minor update: a bug fix.
A bug surfaced in the Component functions in Excel 2013 and 2016 which made Excel crash. Resolved.

Ersatz version 1.33

April 6, 2015
A minor update: a bug fix.
The rank correlation functions (ErRankCorr, ErDirichletCorr, and ErMultinomialCorr) were not compatible with the Multiple runs option, and returned the same stream of correlated random numbers for each of the multiple runs. Resolved.

Ersatz version 1.32

February 26, 2015
A minor update: a bug fix.
A bug surfaced in Excel 2013 64 bit on Windows 8.1 which made Excel crash. Resolved.

Ersatz version 1.31

October 25, 2012
A minor update: an additional function and two bug fixes.

Additional function:

ErSortedArray

Please consult the *Ersatz Function Overview* about this function.

Bug fixes:

1. A bug in ErDataArray could make Excel crash. Resolved.
2. With more than one ErCondRetrieveArray function in a workbook, retrieved arrays could get mixed up. Resolved.

[15] Note that both reported versions should always be in sync. If not, something has gone wrong while updating Ersatz (e.g. you had Excel running while you were updating). See the User Guide section on Installation Issues for further guidance.

Ersatz version 1.3

August 15, 2012
A major update: support for 64-bit Excel is added.

Ersatz version 1.2

April 26, 2012
An important update, with additional distributions, functions, functionality, and some bug fixes.

Additional distributions:

1. Polya. Function name: ErPolya.
2. Four parameter, scaled Beta. Function name: ErBeta4.
3. Delaporte. Function name: ErDelaporte.
4. Correlated Dirichlet. Function names: ErDirichletCorr, ErDirichletCorrIn, ErDirichletCorrOut.
5. Correlated multinomial. Function names: ErMultinomialCorr, ErMultinomialCorrIn, ErMultinomialCorrOut.

Please consult the *Ersatz Function Overview* for details.

Additional functions:

1. ErData.
2. ErDataArray.
3. ErRunDataArray.
4. ErStore.
5. ErRetrieve.
6. ErStoreArray.
7. ErRetrieveArray.
8. ErCondStoreArray.
9. ErCondRetrieveArray.

Please consult the *Ersatz Function Overview* about these functions.

Additional functionality:
1. The Ersatz Help menu now has an entry 'Example spreadsheets' which allows viewing and opening of the examples that come with the installation.
2. The Optimization after each iteration routine now has the option of running the optimization after the first iteration only.

Bug fixes:

1. The histogram would sometimes cause an error when an output variable with zero standard deviation was chosen. Resolved.

2. The scatter plot would throw an unintelligible error when no Ersatz input functions were present in the spreadsheet. It now gives an informative error message.
3. The correlation coefficient of the scatter plot would cause a 'division by zero' error when one of the variables had zero standard deviation. The correlation coefficient is now 'not available'.
4. In the Dirichlet function the parameters were incorrectly assumed to be integers, they are now reals.
5. The multinomial function in some circumstances would produce too much variability for categories with a small number. Resolved.

Ersatz version 1.13

March 5, 2011

A minor update, with some bug fixes.

1. The workshop version of Ersatz (only available on request) would sometimes crash Excel when the Multivariate sensitivity output was chosen. Resolved.
2. The ErRankCorCom functions would return 0 instead of the mean value during the stock taking of Ersatz functions that precedes each run. This could cause a fatal error message if you had an ErOutput function that used the ErRankCorCom function result such that a 0 value caused an error. Resolved.
3. The 'Execute macro before iteration' option sometimes would cause an error. Resolved.

Ersatz version 1.12

September 16, 2010

A minor update, with a bug fix.

Some installations of Excel crashed when starting up with the Ersatzdll.xll file as an active add-in. Resolved.

Ersatz version 1.11

January 31, 2010

A minor update, with some extensions of functionality and some bug fixes.

The complete list of changes:

1. The Distribution viewer has added a number of distributions, now a total of 26 distributions are available for exploration.
2. The Distribution viewer has an added message field to inform you when your typed-in parameter values are invalid (could not be converted to a number or are outside the defined range) or when these values raised a calculation error (usually because of overflow: a value in the calculation exceeded the defined range for that variable).
3. Sensitivity analysis was disabled when the Multiple run option was checked. With ErRunSenseInput functions defining the input variables of interest in your workbook, you can now do multivariate probabilistic sensitivity analysis on these input variables, see the *User Guide* on Multiple runs for details.
4. Bug fixes:

a. In the Distribution viewer sometimes a discrete distribution graph would be shifted one number to the right on the X-axis.
b. The Distribution viewer would throw all kinds of error messages for some distributions and parameter values. These messages are now intercepted, translated, and reported in the message field (see point 2 above).

Ersatz version 1.1

January 21, 2010

A major update, because of an important extension of the functionality: Ersatz now has four optimization algorithms build-in. In addition there is a minor bug fix.
The complete list of changes:

1. Optimization is added, with four different algorithms: two deterministic (Quasi-Newton and Down-hill Simplex) and two stochastic (Simulated Annealing and Cross-Entropy). There are two added functions (ErMinimize and ErMinimizeResult) to give access to the optimization algorithms, a large added section on optimization in the *User Guide*, and two additional example workbooks to illustrate their use.
2. Some of the functions that take Excel ranges as input (such as ErFixed and ErEmpirical) could not handle correctly an input range consisting of a single row. This bug was introduced in version 1.01, and has now been fixed.

Ersatz version 1.01

October 10, 2009

This is the first version of Ersatz without the 'beta' qualification, so the first 'official' release. It is also a major upgrade from version 1.0 (beta), although many of the changes are under the hood and users (in particular of Excel 2003 and earlier) should hardly notice.
The main change is that Ersatz now actively supports Excel 2007. The previous version relied on the backward compatibility of Excel 2007, but this version has a dual interface and, after querying which Excel version is running, presents the appropriate one. This required a major revision of the Ersatz plumbing, and my expectation is that an added benefit will be greater stability, in all supported versions of Excel.
Here is the complete list of changes:

1. Active support of Excel 2007 new features, in particular the 'big grid'. The issue that 2007 users could not run Excel macros from Ersatz has been resolved as well. However, multi-threaded recalculation will have to wait until version 2.0.
2. The Excel function wizard now displays help texts on the Ersatz functions and the parameters they take. If you do not know what the 'Excel function wizard' is, check it out in the Excel Help: it is really useful. Unfortunately, Excel does not support 'tool tips' for user defined (that is: not innate Excel) functions.
3. The documentation is now complete. In particular, the Ersatz User Guide has sections on Error messages, Trouble shooting, Known issues, and a Technical

appendix with a detailed discussion of installation issues, among other things. This should be your first port of call if things go wrong (or seem to).

4. Some functions that take Excel ranges as a parameter (such as ErFixed) were limited to a maximum range size of 8192 cells. This limit has been lifted.
5. The Bernoulli distribution has been added (function name: ErBernoulli).

About the author

Jan Barendregt is the founder and owner of EpiGear International Pty Ltd, Sunrise Beach, Queensland, Australia (www.epigear.com). Epigear makes free and low-cost software and provides consultancy services. Email: jan@epigear.com.
Previous positions were at Erasmus University, Netherlands, WHO Geneva, and University of Queensland, Australia.

References

Barendregt, J. J. (2010). "The effect size in uncertainty analysis." <u>Value in Health</u> **13**(4): 388-391.

Barendregt, J. J. and A. Blakely (Draft). "On the choice of distributions in probabilistic bias analysis."

Briggs, A., M. Sculpher, et al. (1994). "Uncertainty in the economic evaluation of health care technologies: the role of sensitivity analysis." <u>Health Econ</u> **2**(3): 95-104.

Briggs, A., M. Sculpher, et al. (2006). <u>Decision Modelling for Health Economic Evaluation</u>. Oxford, Oxford University Press.

Conover, W. J. (1999). <u>Practical Nonparametric Statistics</u>. New York, John Wiley & Sons.

Dalton, S. (2007). <u>Financial Applications Using Excel Add-in Development in C/C++</u>. Chichester, Wiley.

Devroye, L. (1986). <u>Non-uniform random variate generation</u>. New York, Springer Verlag.

Fox, M. P., T. L. Lash, et al. (2005). "A method to automate probabilistic sensitivity analyses of misclassified binary variables." <u>International Journal of Epidemiology</u> **34**: 1370–1376.

Gartner, C. E., J. J. Barendregt, et al. (2009). "Predicting the future prevalence of cigarette smoking in Australia: how low can we go and by when?" <u>Tob Control</u> **18**: 183-189.

Gelman, A., J. B. Carlin, et al. (2004). <u>Bayesian data analysis</u>. Boca Raton, Chapman & Hall/CRC.

Gentle, J. E. (2003). <u>Random number generation and Monte Carlo methods</u>. New York, Springer.

Law and Kelton (2000). <u>Simulation analysis</u>.

Marsaglia, G. and A. Zaman (1991). "A New Class of Random Number Generators." <u>Annals of Applied Probability</u> **3**(3): 462-480.

Press, W. H., S. A. Teukolsky , et al. (1992). <u>Numerical Recipes in FORTRAN 77: The Art of Scientific Computing</u>. Cambridge, Cambridge University press.

Press, W. H., S. A. Teukolsky , et al. (2007). <u>Numerical Recipes: The Art of Scientific Computing</u>. Cambridge, Cambridge University press.

Rubinstein, R. Y. and D. P. Kroese (2008). <u>Simulation and the Monte Carlo method</u>. Hoboken, Wiley.