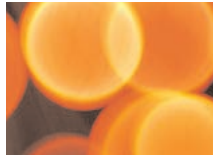
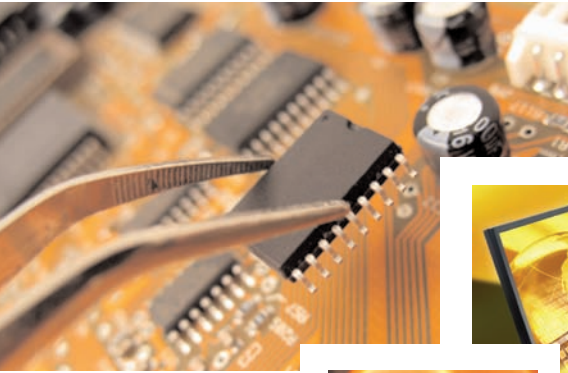


TECHNOLOGY SECTION

"A KNOWLEDGE COMMUNITY FOR THE SOCIETY OF ACTUARIES"

CompAct

Issue No. 26 • January 2008



Inside

Letter from Incoming Editor _____ **2**
by Howard Callif

Letter from the Chair _____ **3**
by Kevin Pledge

The Microsoft Excel OFFSET Function _____ **4**
by Damian A. Birnstihl

Parallel Computing on Multi-Processor Computers Using Freely Available Tools _____ **8**
by N.D. Shyamal Kumar

To Err Is Human; To Correct, Divine _____ **11**
by Mary Pat Campbell

Actuaries
Risk is Opportunity.sm

Letter from Incoming editor

by Howard Callif



The Technology section council met at the annual SOA meeting in Washington, D.C., and discussed plans for the next year. There was a luncheon for all section members, and Paula Hodges was presented with a memento for her hard work and leadership as council chair (see accompanying picture). A hearty "thank you" to all of the council members for their hard work and effort for the Technology Section over the past year.

As part of the transition, I will be taking over as Editor of CompAct from N. D. Shyamalkumar (a.k.a. Shyamal). Shyamal has done a superior job as editor, and he will be a tough act to follow. Our goal is to provide valuable information in every issue, and Shyamal has done more than his share to ensure that the

newsletter is useful to all of our members. We extend a special "thank you" to him for his hard work!

We will definitely try to address some major items from the survey of our members in upcoming issues of the newsletter. Several areas stood out as items members wanted to learn more about: Actuarial software (valuation and pricing), Communication software; Project management software; Data warehousing/management, and Grid computing/distributed processing. We already have someone working on an article on Grid computing for the next issue. We will be working to provide articles on many of the other topics, but we will need your help. Addressing some of these topics is difficult, since they require technical and detailed understanding of complicated topics. The members of this section have this knowledge, and we need you to volunteer to "step up" and share it with the community.

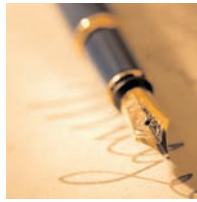
For example, a specific suggestion from the survey was to have an independent assessment of various valuation platforms, and how companies are using each. This could be a comprehensive review of the software, including how it is used "in the field." Although this type of article would require a significant amount of input and expertise, we could provide a valuable resource to the actuarial community. We could also review pricing software, either as part of the same article, or as a separate project. If you are interested or able to provide guidance on this topic, please contact me, or one of the council members.

We also welcome other suggestions and comments, especially regarding ideas you have on how this newsletter can provide value to you.

Howard Callif



Howard Callif is a senior system architect at COSS in the Illustrations unit. He can be reached at howardc@cooss.com.



Letter from the Chair

by Kevin Pledge

2008 is going to be an exciting year for the Technology Section; a number of projects are underway while at the same time your section council is keen to take on new challenges.

Three projects that should come to fruition this year are:

- **Scenario Manager**—the Standard Scenario Format Working Group has agreed on an XML scenario and developed a utility program to present yield curves and other economic indicators. During 2008 we will be seeking adoption by major software vendors. Contact Steve Strommen or Carl Nauman if you are interested.
- **Rate Table Manager**—a number of volunteers are busy validating the tables, while we investigate a long-term solution for managing this service. Contact Joe Luizzo if you are interested in helping with this.
- **Section Portal**—this will help coordinate volunteer activities. We currently have a proof of concept and this should become a useful tool in the coming year. Contact Paula Hodges if you are interested in contributing.

Also continuing from last year:

- **Technology Education**—last year we formed a subcommittee to give input on technology education.

In addition, we plan to build on our recent successes:

Publications:

- Four newsletters (January, April, July and October).
- Bi-monthly Tech Update e-mails from the chair.

Networking:

- Providing opportunities for members to network at actuarial meetings.

Education:

- Last year we had our first webinar—this was a great success, I hope we can follow up on this with more in the coming year.

We welcome your input on these projects and any activities that you feel we should be undertaking. Also, we always welcome volunteers—it is the enthusiasm and contribution from the members that make this section what it is.

Please contact me directly if you have any suggestions regarding activities we should be taking on or if you would like to volunteer.



Kevin Pledge, FIA, FSA, is president and CEO of Insight Decision Solutions in Markham, Ontario. He can be contacted at kpledge@insightdecision.com.

The Microsoft Excel OFFSET Function

by *Damian A. Birnstihl*

Many of us use Microsoft Excel on a daily basis. We have mastered the basics, such as VLOOKUP and SUMPRODUCT. We can create charts and professional looking documents. We may even have ventured into more advanced features, like pivot tables and macros. But there are a number of functions and features of Excel that despite their power are used infrequently. One example is the OFFSET function. Learning and applying this function can significantly improve your efficiency in creating and using spreadsheets.

The OFFSET function returns the value in the cell a specified number of rows and columns away from a specified starting point. The syntax is **OFFSET (reference, rows, cols, height, width)**. The **reference** parameter is the starting point. Typically, it will be a single cell, but it can be a range of contiguous cells. The **rows** and **cols** parameters represent how many **rows** and columns away from the starting point you want to go. A positive value of the rows parameter means the target is below the reference cell, and a negative value means the target is above the reference cell. Similarly, a positive value of the **columns** parameter means the target is to the right of the reference cell, and a negative value means the target is to the left of the reference cell. The height and width parameters are optional; if omitted, Excel assumes the same height and width as **reference**.

For example, `OFFSET(B5,2,1)` returns the value two rows below and one column to the right of the reference cell B5, or in other words, the value in cell C7. `OFFSET(B5,-2,-1)` returns the value two rows above and one column to the left of cell B5, or in other words, the value in cell A3.

The real power of OFFSET is realized when it is used in conjunction with the ROW and COL-

UMN functions. The syntax of these functions is simple: `ROW(reference)` and `COLUMN(reference)`. For example, `ROW(F2)` returns a value of two, and `COLUMN(F2)` returns a value of six. If the reference parameter is omitted, the ROW returns the value of the row in which the formula is typed, and likewise for `COLUMN()`. For example, typing `=ROW()` in cell A3 returns a value of three, and typing `=COLUMN()` in the same cell returns a value of one.

Now let's look at an example that combines the use of OFFSET and ROW. Suppose you have a spreadsheet containing the months of the year in cells A2:A13, as shown in Figure 1.

Figure 1

	A
1	Month
2	January
3	February
4	March
5	April
6	May
7	June
8	July
9	August
10	September
11	October
12	November
13	December

Now suppose you want to reverse the data so that December is at the top of the list and January is at the bottom, with the results to be shown in cells C2:C13. No need to re-type the data or cut and paste; one simple formula can do the trick. In cell C2 type the formula `=OFFSET(A13,2-ROW(),0)`. Then copy this formula to cells C3:C13, and you have the desired result as shown in Figure 2.



Damian A. Birnstihl, FSA, MAAA is a director of actuarial services with Aetna. He can be reached at 602.659.1759 or at Damian.Birnstihl@SchallerAnderson.com.

Figure 2

	A	B	C
1	Month		
2	January		December
3	February		November
4	March		October
5	April		September
6	May		August
7	June		July
8	July		June
9	August		May
10	September		April
11	October		March
12	November		February
13	December		January

The key is figuring out how to set the parameters in the OFFSET function. This can be done using the following four-step process:

- First, determine what value you want in each target cell. In this example, we want cell C2 to have the value "December" from cell A13, C3 to have the value "November" from cell A12, etc.
- Second, choose a reference. This can be any cell on the spreadsheet, but a convenient choice is to use the cell in the original array that you want to appear in the first cell of the revised array. In this example that would be A13.
- Next, determine the offsets from this reference needed to produce the desired results. We stated above that we want C2 to contain the value from cell A13; this is zero rows below and zero columns to the right of the reference cell A13. In other words, C2 needs to evaluate to `OFFSET(A13,0,0)`. Similarly, we want C3 to contain the value from A12; this is one row above and zero columns to the right of the reference cell A13. In other words, C3 needs to evaluate to `OFFSET(A13,-1,0)`. Following the same logic, C4 must evaluate to `OFFSET(A13,-2,0)`. For cells C5:C13 the emerging pattern holds: in each subsequent cell, the row offset decreases by one.
- Finally, look for a relationship between these offsets and the rows and columns of the cells in which the formulas will be entered, and translate the relationship into a formula. From step 3 we know that the row offset must be 0 in cell C2, -1 in cell C3, -2 in cell C4, etc. Notice that each row offset can be expressed as `2-ROW()`. The column offset in this example is always zero. Thus, the desired formula is `OFFSET(A13,2-ROW(),0)`.

The process of determining the appropriate parameters becomes second nature with a little practice.

A word of caution is in order. The formula in the preceding example may no longer produce the desired results if rows are inserted into the spreadsheet. This can be seen by inserting a row at the top of the spreadsheet. Ideally, this action would not cause our results to change. In fact, inserting a row does indeed cause our resulting list to start at November instead of December. However, this can easily be avoided. In the example, we used a row offset of `2-ROW()`. The fixed value two was appropriate before the row was inserted, but is not appropriate afterwards. So instead of a fixed value, we can reference the cell itself. If you're working the example on your computer, delete the row that you added, and in place of the original formula in cell C2, type in `=OFFSET(A13,ROW(C2)-ROW(),0)` and copy this formula to cells C3:C13. Now that the fixed reference has been replaced with a ROW reference, inserting rows at the top of the spreadsheet will not affect the results. Note that I have "anchored" both `A13` and `C2` using dollar signs.

Now, let's look at a similar example using OFFSET and COLUMN. Suppose you have the spreadsheet shown in Figure 3.

... there are a number of functions and features of Excel that despite their power are used infrequently.

(continued on page 6)

Figure 3

	A	B	C	D	E
1	Alpha	Bravo	Charlie	Delta	Echo

Our task is to reverse the list and put the results in cells A3:E3. In cell A3, type =OFFSET(\$E\$1,0,COLUMN(\$A\$3)-COLUMN()), and then copy this formula to cells B3:E3. This produces the desired result as shown in Figure 4.

Figure 4

	A	B	C	D	E
1	Alpha	Bravo	Charlie	Delta	Echo
2					
3	Echo	Delta	Charlie	Bravo	Alpha

Next, let's take the original list in Figure 3 and convert it from a 1x5 array into a 5x1 array, or in other words convert the row of data into a column with Alpha at the top and Echo at the bottom, and let's put the results in cells G1:G5. We'll use the four-step process introduced earlier to guide us:

- We want cell G1 to contain the value "Alpha" from cell A1, G2 to contain "Bravo" from B1, etc.
- We choose \$A\$1 as the reference since it contains the desired value in the first target cell, G1.
- G1 must evaluate to OFFSET(\$A\$1,0,0), G2 must evaluate to OFFSET(\$A\$1,0,1), G3 must evaluate to OFFSET(\$A\$1,0,2), etc.
- We see that the row offset is always zero and that the column offset is the current row minus one. Thus, the formula to type into G1 and to copy to G2:G5 is =OFFSET(\$A\$1,0,ROW()-ROW(\$G\$1)).

Now that we've looked at some basic examples, let's look at some more advanced applications of OFFSET. Suppose we have the 3x5 array shown in Figure 5, and suppose that we want to transpose it to a 5x3 array and put the result in cells G1:I5.

Figure 5

	A	B	C	D	E
1	10	40	70	100	130
2	20	50	80	110	140
3	30	60	90	120	150

Using the four-step process:

- We want the first row of the original array to become the first column of the resulting array, and likewise for the second and third rows of the original array.
- A convenient choice for the reference is cell \$A\$1.
- G1 must evaluate to OFFSET(\$A\$1,0,0). We want cell H1 to contain the value from cell A2, so H1 must evaluate to OFFSET(\$A\$1,1,0). Similarly, we want cell G2 to contain the value from cell B1, so G2 must evaluate to OFFSET(\$A\$1,0,1). If necessary, look at some additional cells until the pattern becomes evident.
- From the previous step, we note that the row offset relates to the column in the transposed array and that the column offset relates to the row in the transposed array. The formula is =OFFSET(\$A\$1,COLUMN()-COLUMN(\$G\$1),ROW()-ROW(\$G\$1)). Type this formula in cell G1 and then copy the formula to cells G1:I5 to obtain the desired result as shown in Figure 6.

Figure 6

	G	H	I
1	10	20	30
2	40	50	60
3	70	80	90
4	100	110	120
5	130	140	150

I hope that you are beginning to see the power of the OFFSET function. But it can do more than just flip arrays. For our final example, we will look at a practical example of how OFFSET can be used along with SUM. Consider the hypothetical claim triangle in Figure 7.

Figure 7

	A	B	C	D	E
1		July	August	September	October
2	July	10			
3	August	20	30		
4	September	40	50	60	
5	October	70	80	90	100

Here, each column represents an incurred month and each row represents a month of payment. Now suppose that we want to build a table of cumulative paid claims by

incurred month as shown in Figure 8. This table can be built with a single formula!

Figure 8

	G	H	I	J	K
1	Incurring Month	Lag 0	Lag 1	Lag 2	Lag 3
2	July	10	30	70	140
3	August	30	80	160	
4	September	60	150		
5	October	100			

For example, cell J3 represents claims incurred in August and paid through lag 2, i.e., October. The value is 30 + 50 + 80 = 160. Figure 9 shows the calculations necessary to produce the desired results. This is just an intermediate step that we will use as a guide to determine the appropriate OFFSET formula.

Figure 9

	G	H	I	J	K
1		Lag 0	Lag 1	Lag 2	Lag 3
2	July	=B2	=B2+B3	=B2+B3+B4	=B2+B3+B4+B5
3	August	=C3	=C3+C4	=C3+C4+C5	
4	September	=D4	=D4+D5		
5	October	=E5			

So, in each case we want to sum starting at an offset, an equal number of rows and columns from cell B2, with both the row and column offset determined by the row within the array. Further, each summation is over an nx1 range, where n is determined by the column within the array. We also note that the lower right half of the array is blank because these represent payment months in the

future. We can even make our formula account for this by using an IF statement. Here is the formula:

```
=IF(ROW()-ROW($H$2)+COLUMN()-COLUMN($H$2)>3,"",SUM(OFFSET($B$2,ROW()-ROW($H$2),ROW()-COLUMN($H$2)+1,1)))
```

Typing this formula into cell H2 and copying it to cells H2:K5 produces the desired result. Note that this formula uses the optional height and width parameters in the OFFSET function. Let's evaluate the formula in cell J3 to see how it works. In J3 the formula is evaluated as follows:

```
=IF(3-2+10-8>3,"",SUM(OFFSET($B$2,3-2,3-2,10-8+1,1)))
```

```
=IF(3>3,"",SUM(OFFSET($B$2,1,1,3,1)))
```

```
=SUM(C3:C5)
```

```
=160
```

As you can see, the OFFSET function is both powerful and efficient for manipulating arrays of data. We have looked at several examples of how OFFSET can be used along with ROW, COLUMN, and SUM to reverse, transpose, and sum arrays with a single formula. Master the use of OFFSET and you may never have to cut and paste again! 📄

Parallel Computing on Multi-Processor Computers Using Freely Available Tools

N. D. Shyamal Kumar

Increasing computational prowess is driving the demand for more realistic modeling and is also partly responsible in regulation becoming less simplistic. But this increasing prowess in the future is going to be delivered by increasing the number of processing units rather than by increasing the clock speed. This trend is already seen with new workstations usually having two or more computing cores. Hence it is imperative that quants acquire parallel computing skills. In this article we show that it takes little to write parallel code to implement embarrassingly parallel algorithms, which is exciting given the prevalence of problems yielding to such algorithms. In one such problem that we discuss here, to our surprise, on a dual core processor we were able to get a speedup greater than two—close to 2.6 in fact! And all of this using only freely available tools for the Windows® operating system.

Programming Paradigm: In this article we will constrain ourselves to shared memory parallel computing, i.e., parallel computing where all threads have access to the same shared memory. The discussion of distributed memory parallel computing, the paradigm for grid computing, will be left for another article. Moreover, we will further restrict our attention to the use of the OpenMP API, which is based on the fork and join execution model. This execution model, see Figure 1, is one where the main thread spawns out multiple worker threads to simultaneously execute sections of code that can be run in parallel, thus speeding up the overall computation. This will be made clearer below when we discuss the implementation of the solution to our problem. The OpenMP site is at <http://www.openmp.org> where one can find useful tutorials (especially, see [3]), list of books on OpenMP, etc.

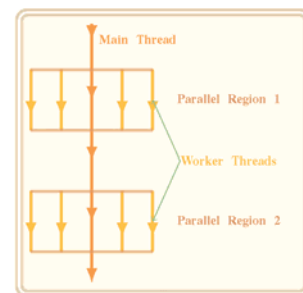


Figure 1: Fork and Join Model

Actuarial Problem: The computational problem we consider is the calculation of the break-even survivorship benefit for the disability product from the Danish market considered in [1]. The age-to-65 product is sold to healthy (able) insureds with an annual premium rate (assumed to be paid continuously) of DKK 10,000, with a premium waiver while the insured is disabled (or invalid), a death benefit at time t defined to be the benefit reserve at that time for a healthy insured, and a survivorship benefit of S payable at the end of the term of the product. Figure 2 depicts the multi-state model underlying the product with disability rate $\sigma(\cdot)$ and mortality rates $\mu(\cdot)$ and $\nu(\cdot)$ for the able and disabled, respectively. The motivation for this product with a strong savings element was to enable insurance companies to compete with banks and other savings institutions, which were only allowed to sell products with an element of insurance (the waiver of premium is included in this product for this sole purpose). This product has didactic value as many companies did not price the product using the correct reasoning, see [1]. The approach taken in [1] is that of using Thiele's differential equation to derive a double-integral expression for S . Here we will adopt a different approach which will lead to a simple Monte-Carlo solution.



*N.D. Shyamalkumar
ASA, an assistant
professor of
statistics &
actuarial science
at the University of
Iowa. He can be
contacted at
shyamal-kumar
@uiowa.edu*

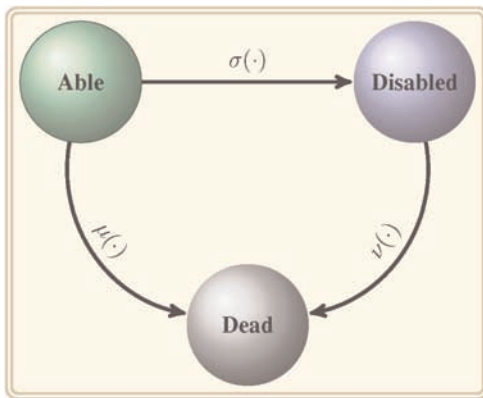


Figure 2: Multi-State Model Underlying the Product

<http://www.iro.umontreal.ca/~lecuyer/myftp/streams00/c>. Since the hazard rates pertain to the Makeham distribution, we used the algorithm as given in [2] to simulate the random occupation time in each state. This algorithm uses two independent exponential variables, and we generated these from uniforms using the log transform.

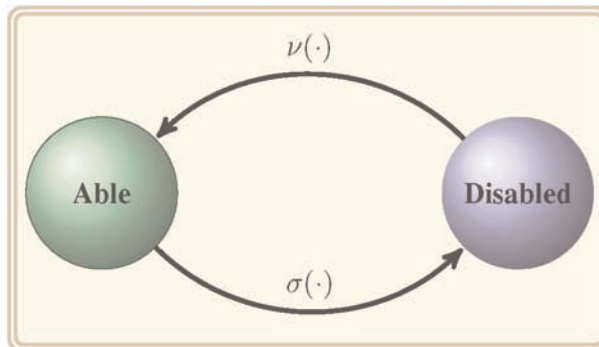


Figure 3: The Reduced Multi-State Model Driving the Algorithm

Figure 2: The Multi-State Model Underlying the Product

Embarrassingly Parallel Algorithm: It is not difficult to see that the survivorship benefit S in the above problem is the same as in the following problem: consider an age-to-65 product sold on a healthy (able) insured for a premium at the annual rate of DKK 10,000, carrying a survivorship benefit of S and a waiver of premium while the insured is disabled; Disability rate is $\sigma(\cdot)$ (as above), and unlike the original product the rate of recovery is taken to be $\nu(\cdot)$ and mortality is ignored (see Figure 3). Since mortality is ignored, S is equal to the expected value of the accumulated premiums. This characterization of S lends itself to Monte-Carlo simulation of S . Hence we will be able to achieve a speedup almost equal to the number of processors (i.e., by dividing the number of simulation runs equally among the processors). Algorithms achieving such speedups are termed embarrassingly parallel.

Caveat for Simulation in Parallel: Unlike sequential simulation where independence of the pseudo-random numbers is more or less guaranteed, one has to take extra care to ensure this in the parallel setting. For one, often pseudo-random number generators are by default initialized using the system clock and this could make many if not all of the simulations carried out on different processors to be identical. A well tested C package that helps to create and easily manage independent streams is RNGStreams, written by Pierre L'Ecuyer. For details and to download the library, see the webpage at

Comments on the Code and Tools: We decided to program in C (see listing) and use the MinGW (Minimalist GNU for Windows) packaged GNU compilers, see <http://www.mingw.org>.

For code using the OpenMP API one will also need the pthreads library which can be found at <ftp://sourceware.org/pub/pthreads-win32/prebuilt-dll-2-8-0-release/>. The OpenMP paradigm allowed us to write parallel code in such a way that it compiles as a sequential program without the `-fopenmp` compiler option. The C code of this project is available on request from the author.

The sequential part of the code is comprised of the lines of code excepting the pragma compiler directives and the code in between `#ifdef` and `#endif` pre-processor conditionals. We observe that the key lines of this part of our code are those from lines 50-69. These simulate the accumulated amount of premiums paid by a single insured. This is repeated for NRUNS number of insureds in order to reduce estimation error.

Now, moving on to the parts of the code that help parallelize the simulation, we notice that

(continued on page 10)

between the first pre-processor conditionals we include the OpenMP library and between the second we set the number of threads. The first pragma directive on line 39 implies that the code between lines 40-70 will be run by each thread, and that all threads will be sharing the global variable `delta` while keeping private copies of all the variables declared within the section. Moreover, the reduction clause creates for each thread its own copy of `mean` (resp., `var`) for the duration of the parallel section, and at the end of the parallel section, unlike private variables which are simply released, the values in these copies will be merged (in our case added) and stored in the global variable `mean` (resp., `var`).


The call to the function `RngStream.CreateStream` on line 44 ensures that each thread has its own independent pseudo random number generator, and that these generators are mutually independent of each other. In order to achieve these goals the function `RngStream.CreateStream` uses a static variable to store its state, and hence is not (and cannot be) thread safe. For this reason, this function call is encapsulated in a pragma `omp critical` directive that allows a thread at a time to make the function call. The last pragma directive on line 48 splits the range of the loop index equally among the threads (load balancing options are available too), and this is the part that achieves a speedup equal to exactly the number of processors.

Results: The computations were done on an Intel® Core™ 2 Duo Processor E6600 (2.4GHz, 4MB shared L2 Cache) box with 4GB RAM and running the Windows Vista™ Enterprise OS. The value of `S` was estimated with a relative error less than 10 basis points by simulating the accumulated premiums for 100 million insureds. The code was run five times with each value for the number of threads listed in Table 1, wherein we report the observed mean running times. It was expected that with two threads we will get a speedup of close to, but less than, two. The

surprises in Table 1 were that we achieved a final speedup far greater than two (likely due to threading allowing more efficient usage of different units of Core™ 2 Duo), and that this required using many more than two threads. In summary, we have found that it is easy to implement embarrassingly parallel algorithms using the OpenMP API, solely employing tools freely available on the Web, with the reward being a speedup by a factor close to the number of processors—or possibly even greater as in the case of Intel® Core™ 2.

Num. of Threads	Time in Seconds	Speedup
1	134.52	1
2	67.41	1.996
4	60.10	2.238
8	55.68	2.416
16	53.38	2.520
32	52.58	2.559
64	52.17	2.578
128	51.94	2.590
256	51.87	2.593

Table 1: Effect of Number of Threads on Computational Time

Acknowledgement: I thank Luke Tierney and Kate Cowles for their enjoyable course on High Power Computing (at the U. of Iowa) which exposed me to many of the technologies discussed in this article. Also, I thank the actuarial science students in my topics course whose interest motivated me to bring this article to its final form. 

References

- [1] Ramlau-Hansen, Henrik (1990). Thiele's Differential Equation as a tool in Product Development in Life Insurance, *Scandinavian Actuarial Journal*, 1990:97-104.
- [2] Pai, Jeffrey S. (1997). Generating Random Variates with a given Force of Mortality and finding a suitable Force Of Mortality by Theoretical Quantile-Quantile Plots. *Actuarial Research Clearing House*, 1997(Vol. 1), 293-312.
- [3] van der Pas, Ruud (2005). An Introduction to OpenMP. Available at http://www.nic.uoregon.edu/iwomp2005/iwomp2005_tutorial_openmp_rvdp.pdf

C Code to Calculate the Endowment Amount

```

#include <stdio.h>    /* Standard C input/output library */
#include <math.h>     /* C library for math functions like log, pow etc.. */
#include <hr_time.h>  /* hr_time.c - Support for Timing; http://cplus.about.com/od/howtodoth-
ingsin1/a/timing.htm */
/*
* RNGStreams - Library for Multiple Streams of Random Numbers.
* http://www.iro.umontreal.ca/~lecuyer/myftp/streams00/c/
*/
#include "RNGStream.h"
#include "SRandom.h"  /* Provides rmakeham for Generating from Makeham */
#ifdef _OPENMP
    #include <omp.h>    /* OpenMP Library */
#endif

#define AGE 30
#define TERM 35
#define INTEREST 0.045 /* Annualized Interest */
/* Makeham Hazard Rate: A + B * exp(C*t) */
    #define AMU 0.0005    /* MU - Hazard Rate for Mortality */
    #define CMU 0.08749823558
    #define ASIGMA 0.0004 /* SIGMA - Hazard Rate for Mortality */
    #define CSIGMA 0.1381551353
    #define BSIGMA 0.00021877616 /* pow( 1 0 . 0 , ( 0 . 0 6*AGE-5.46) ) */
    #define BMU 0.001047128548051 /* pow( 1 0 . 0 , ( 0 . 0 3 8*AGE-4.12) ) */
#define NRUNS 100000000 /*No. of Runs*/

int main()
{
    double delta=log(1.0+INTEREST), mean=0, var=0;

#ifdef _OPENMP
    int nthreads;
    printf("Please Enter the Number of Threads You Wish to Use: ");
    scanf("%d",&nthreads);
    omp_set_num_threads(nthreads);
    printf("The Number of Threads Used: %i\n", nthreads);
#endif

    stopWatch t;
    startTimer(&t);

#pragma omp parallel default(none) shared(delta) reduction(+:mean,var)
    { /* Parallel Section */
        RngStream g;
#pragma omp critical
        {

```

```

        g = RngStream_CreateStream (""); /* One Thread at a Time */
    }
    int j;
    double time_to_term, tsigma, tmu, bmu, bsigma, premium;
    #pragma omp for
    for (j=0;j<NRUNS;j++) {
        premium=0;
        time_to_term=TERM;
        tsigma=0;
        tmu=0;
        bmu=BMU;
        bsigma=BSIGMA;
        /* Generate the Premium Paid by a Single Insured*/
        while (time_to_term>0.00000001){
            /* Insured in Able State*/
            bsigma=bsigma*pow(10.0,(0.06*(tmu+tsigma))); /* Shift Hazard Rate Sigma */
            tsigma=rmakeham(&g, ASIGMA, bsigma, CSIGMA); /* Generate Time in Able State */
            if (tsigma> time_to_term) tsigma=time_to_term;
            premium=premium+pow(1.0+INTEREST,time_to_term)-pow(1.0+INTEREST,time_to_term-tsigma);
            time_to_term=time_to_term-tsigma;
            if (time_to_term>0) { /* Insured in Disabled State */
                bmu=bmu*pow(10.0,(0.038*(tmu+tsigma))); /* Shift Hazard Rate Mu */
                tmu=rmakeham(&g, AMU, bmu, CMU); /* Generate Time in Disabled State */
                if (tmu> time_to_term) tmu=time_to_term;
                time_to_term=time_to_term-tmu;
            }
        }
        /* Increment the sum and sum of squares */
        mean=mean+premium;
        var=var+premium*premium;
    }
} /* End of Parallel Section */

mean=10*mean/NRUNS/delta; /* Point Estimate of the Endowment Amount in Thousands*/
var=(100*var/NRUNS/(delta*delta)-mean*mean)/NRUNS; /*Its Estimated Variance */
stopTimer(&t);

printf("Elapsed Time: %g \n", getElapsedTime(&t));
printf("A Point Estimate: %6.3f\n", mean);
printf("Its Std. Error: %10.8f\n", sqrt(var));
printf("A 99.9%% CI: ( %6.3f,%6.3f)\n", mean-3.090232*sqrt(var), mean+3.090232*sqrt(var));
}

```

To Err Is Human; To Correct, Divine

by Mary Pat Campbell



How error-riddled are your spreadsheets? How much can a simple Excel flub cost your company? When you do make a mistake, how likely are you to catch it?

Though we often work with specialized software, Excel is the central tool for most of us, easily adaptable and giving us results faster than much more complicated and targeted programs. However, the danger for material error is there, and the bad news is it's well-nigh impossible to escape error. Even worse, Excel errors can cause large financial damage: at the European Spreadsheet Risks Web site, they've got a page of spreadsheet error horror stories. As an example: a Canadian power company took a 24-million-dollar loss in 2003 when a cut-and-paste error led to a mispriced bid. This is not a one-off event: as of September 2007, the EUSPRIG has 89 news stories, dating back to 1995, of substantial spreadsheet errors. Minor mistakes don't show up in newspapers.

Of course, we're experts, so our spreadsheet errors are rare—right? Let's go to the research and see.

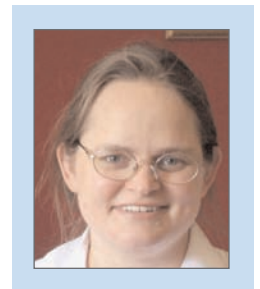
Ray Panko, a professor at the University of Hawaii and a researcher into error rates in spreadsheets, has found that in audit research of spreadsheet errors, 94 percent of spreadsheets reviewed had errors, and about 5 percent of cells in the reviewed spreadsheets contained errors. Some of these errors are immaterial, such as minor typos, but the most insidious type of errors are the ones least likely to be found: omission errors, where something is missing; and logic errors, where the model or calculation is just plain

wrong. We are very unlikely to discover the errors in our own thinking (the source of logic errors), and it's very hard to see that something is not there without explicitly looking for it.

We may think, "Sure, those studies show high error rates, but that's because they're looking at the spreadsheets of a bunch of schmoes ... most likely MBA students." OK, yes, some of the research subjects were MBA students, but in controlling for expertise level, error rates were similar for novices and experts. Even when the spreadsheet task was greatly simplified, the cell error rate was 2 percent for a very artificial situation, as opposed to operational spreadsheets from real businesses. Spreadsheets used in business often were much more complicated, involving links to other files and macros doing a great deal of the calculations.

The reason for novice error rates is simple: they don't know what they're doing. But what about us experts? The problem there is we may underestimate the likelihood of error.

It's hard to detect what you don't expect. If you do only the most cursory of error checks, because you are confident about what you've done, it's highly unlikely that you'll discover that you set up your model incorrectly. Someone else may not catch the error because they don't know enough about the spreadsheet to understand when you've made a mistake. So it's a bind: other people might be more inclined to consider the possibility of an error, but they are unable to find it from ignorance; you would be able to find it, but you're overconfident about your work.



Mary Pat Campbell is a senior actuarial associate for TIAA-CREF in New York, N.Y. She can be contacted at marypat.campbell@gmail.com

(continued on page 14)

The good news is that there are ways to mitigate the errors, and to reduce your chances for error. At the end of the article, I've provided links to resources I've found to be very helpful. The most helpful of all the articles is Philip Bewig's article "How do you know your spreadsheet is right?" If you do nothing else, check that one article out. Some of the sites and articles focus on error rates and types of error; don't discount these—if you know how material errors are most likely to occur, you're more likely to catch them or prevent them.

	Year 1	Year 2	Year 3
Cost of Revenues by Months &	0	176,608	0
	0	176,608	0
	0	247,252	0
	0	600,468	0
	73	211,930	0
	0	247,252	0
	0	0	0

From reading these papers, there seem to be three main ideas that greatly reduce error rate initially and improve error detection:

- Think before you create. Plan your model structure in advance, and consider extreme values that should break your model (you can use those for testing later). Work out the logic in advance, not on the fly. This will result in better structured spreadsheets and reduce the likelihood of logic errors.
- Expect errors as you work. In lab research, when people were made aware of how common spreadsheet errors were, they were much more likely to catch their errors, especially material ones. People

who expect errors examine spreadsheets more carefully, perform more stress tests, and make error-checking part of their routine. Keeping your "spreadsheet ego" in check is a must.

- Work in groups. Research has found that there is a great improvement in error detection when spreadsheets are reviewed in groups. We can be blind to our own errors but very able to see the mistakes of others. As well, different people may be apt to find different types of errors, so that in combination they improve the overall error-correction rate. Panko's research has found a statistically significant improvement in error detection when people work in groups of four (a two-thirds improvement in the error-detection rate); working in pairs did not improve detection to a significant extent.

The sources listed have even more practical and technical ideas (data validation, cell protection, named ranges, R1C1 notation, and more), but the central concept is to be mindful and to be humble.

Spreadsheets have become part of the quantitative sea we swim in—let's make sure they're our Queen Mary, and not our Titanic.

Helpful tips:

"How do you know your spreadsheet is right?" <http://www.eusprig.org/hdykysir.pdf>

"52 ways to prevent spreadsheet problems" http://www.mailbarrow.com/services_excel_prevent.php

"Block that Spreadsheet Error!" <https://www.aicpa.org/PUBS/jofa/aug2002/callahan.htm>


Other Sources:

European Spreadsheet Risks Interest Group: <http://www.eusprig.org/>

Ray Panko's Spreadsheet Research: <http://panko.shidler.hawaii.edu/SSR/index.htm> Tuck School at Dartmouth: Spreadsheet Engineering Research Project

Other spreadsheet error news stories: <http://www.eusprig.org/stories.htm> http://mba.tuck.dartmouth.edu/spreadsheet/product_pubs.html

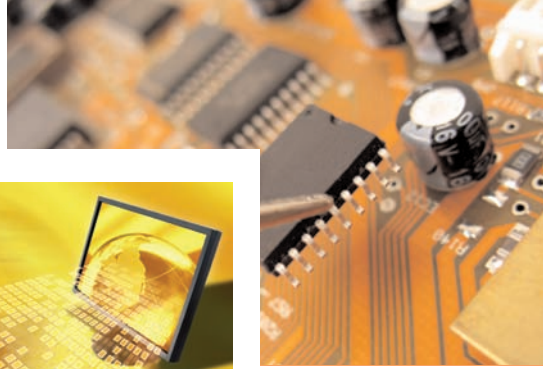
Ray Panko, "What We Know About Spreadsheet Errors", January 2005. Systems Modelling Ltd., Spreadsheet Research Resources

<http://panko.shidler.hawaii.edu/SSR/Mypapers/whatknow.htm> <http://www.sysmod.com/sslinks.htm#Research> 

Winner of the CompAct Article of the Year Prize 06-07



Joe Alaimo (left), winner of the CompAct 2006-2007 "Article of the Year Prize," is presented an iPod Nano from Section Chair Kevin Pledge for his two-part series on Illustration Software Testing.



**Technology Section Newsletter
Issue Number 26
January 2008**

Published quarterly by the
Technology Section of the Society
of Actuaries

World Wide Web: www.soa.org

Nariankadu D. Shyamalkumar

CompAct Editor
Assistant Professor
Statistics and Actuarial Science
241 Schaeffer Hall
The University of Iowa
Iowa City, IA 52242-1409
phone: 319.335.1980
fax: 319.335.3017
e-mail: shyamal-kumar@uiowa.edu

Technology Section Council

Kevin Pledge, **Chairperson**
Tim Pauza, **Vice Chairperson**
Joseph Liuzzo, **Secretary/Treasurer**

Council Members

Van Beach, **Web Site Coordinator**
Carl Nauman, **Scenario Generator**
Timothy Rozar, **Communications
Coordinator**
Carl Desrochers
Holly Loberg
David Minches

Other Volunteers

Howard Callif, **Newsletter Editor**
Matthew Wilson, **Newsletter Sub-Editor
(Web Technologies)**
Robert LaLonde, **2008 Spring Meeting
Program Committee Coordinator**
Paula Hodges, **Past Section Chairperson
and Education Subcommittee
Coordinator**

SOA Staff

Staff Partner

Meg Weber
mweber@soa.org

Staff Support

Susan Martz
smartz@soa.org

Staff Editor

Sam Phillips
sphillips@soa.org

Graphic Designer

Julissa Sweeney
jsweeney@soa.org

Facts and opinions contained in these pages are the responsibility of the persons who express them and should not be attributed to the Society of Actuaries, its committees, the Technology Section or the employers of the authors. Errors in fact, if brought to our attention, will be promptly corrected.



SOCIETY OF ACTUARIES

475 N. Martingale Road Suite 600
Schaumburg, Illinois 60173
www.soa.org