# The Ghost in the Model
# (and other effects of floating point arithmetic)

Presented at ESSA/SIMSOC-VI Conference, 19-21 September 2003, Groningen, Netherlands

J. Gary Polhill, Luis R. Izquierdo, & Nicholas M. Gotts
Macaulay Institute, Aberdeen, UK
{g.polhill,l.izquierdo,n.gotts}@macaulay.ac.uk

*Bistromathics ... is ... a revolutionary new way of understanding the behaviour of numbers. ... Numbers written on restaurant bill pads within the confines of restaurants do not follow the same mathematical laws as numbers written on any other pieces of paper in any other part of the Universe.*

Douglas Adams, "Life, the Universe, and Everything" (1982)

## 1 Introduction

This paper will explore the effects of errors in floating point arithmetic in two published agent-based models: the first a model of land use change (Polhill et al., 2001; Gotts et al., in press), the second a model of the stock market (LeBaron et al., 1999). The first example demonstrates how branching statements with floating point operands of comparison operators create a high degree of nonlinearity, leading in this case to the creation of 'ghost' agents — visible to some parts of the program but not to others. A potential solution to this problem is proposed. The second example shows how mathematical descriptions of models in the literature are insufficient to enable exact replication of work since mathematically equivalent implementations in terms of real number arithmetic are not equivalent in terms of floating point arithmetic. (Henceforth 'mathematics' and 'arithmetic' will refer to the use of real numbers by default.)

## 2 A model of land use change

The following describes the set up of a model of land use change that has emergent effects arising from errors in floating point arithmetic. The model is based on a particular parameterisation, P, of FEARLUS (Framework for Evaluation and Assessment of Regional Land Use Scenarios), described in Polhill et al. (2001) and Gotts et al. (in press). In what follows, terms referring to objects in the model are capitalised, and italicised on first use. FEARLUS consists of a set of *Land Parcels* arranged on a 2D grid of squares, each of which is assigned a *Land Use* by *Land Managers* (the agents in the model) using a *Land Use Selection Algorithm* each *Year* (a cycle in the model). After Land Uses have been assigned, the *Yield* from each Land Parcel is calculated according to the Land Use selected, and this is used to derive an amount of *Wealth* accrued by the Land Manager owning the Land Parcel. This is followed by a process of exchange of Land Parcels between Land Managers. Those Land Managers with negative Wealth put up enough Land Parcels for sale to bring their Wealth to zero or more. Once all Land Managers who need to have put Land Parcels up for sale, a new owner for each such Land Parcel is selected at random from a set consisting of the neighbouring Land Managers with sufficient wealth plus one potential new Land Manager. The cost of the Land Parcel is determined by a model parameter, the *Land Parcel Price* which is constant over space and time. If a Land Manager has sold all of its Land Parcels, it is deemed to be bankrupt, and plays no further part in the simulation.

Land Managers are divided into *Subpopulations* according to their Land Use Selection Algorithm. In P, there are two Subpopulations, RS, whose members choose a Land Use at random for each Land Parcel they own, and HR, whose members choose at random if they lost money on that Land Parcel in the previous Year, but otherwise retain the same non-loss-making Land Use for

the current Year. Newly created Land Managers have an equal probability of belonging to HR or RS.

There are two Land Uses to choose from in P, *lu*1 and *lu*2. The Wealth accrued to a Land Manager who puts *lu*1 on a Land Parcel in the decision phase is –0.6, whilst for *lu*2, it is +0.4. The Land Parcel Price is set to 10000 in P, which is intended to prevent Land Managers from ever accruing sufficient Wealth to be able to afford to buy a Land Parcel during the 200 Years for which the model is run. The process is summarised in Figure 1.

All Land Managers begin with Wealth 0 after having been assigned their first Land Parcel. At the start of the run, Land Parcels are initially assigned Land Uses at random, with an equal probability of *lu*1 or *lu*2 being applied.

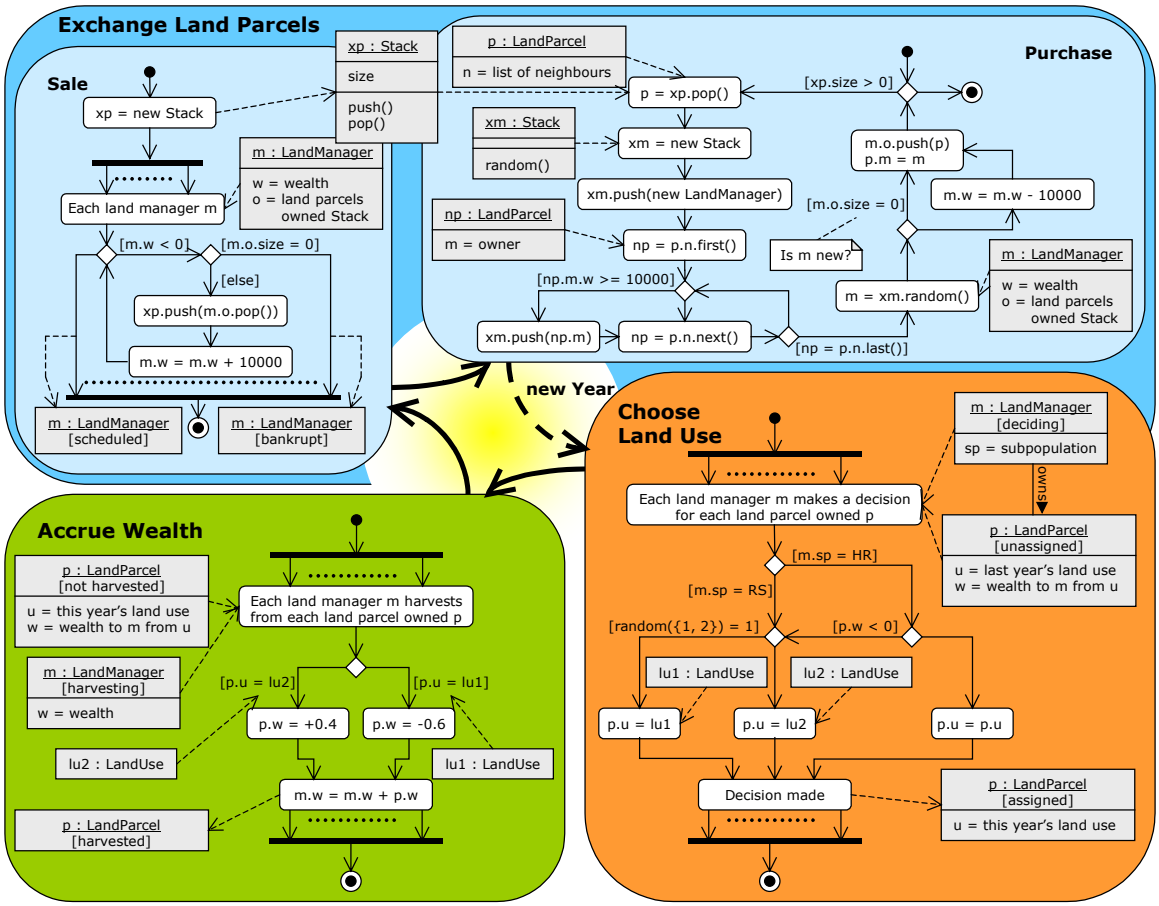



**Figure 1** — UML-style diagrams based on Booch et al. (1999) showing the yearly cycle. At the simplest level of detail, the coloured boxes represent a statechart diagram showing the flow from one state during the yearly cycle to the other, with the dashed line showing the transition from one Year to the next. Inside each coloured box is an activity diagram showing more detail on the processing involved in each state. The dotted lines between the arrows to/from the concurrent process lines are intended to indicate many objects engaged in the same activity. In the 'Purchase' activity diagram, the 'random()' method in the Stack class returns a random member of the stack with equal probability of any member being selected.

The expected outcome, given enough time, is for all Land Parcels to be owned by members of HR on a one-to-one basis, each using the profit-making Land Use (*lu*2); though there is a small non-zero probability that a member of RS will survive for any finite number of Years. However, whilst some runs do yield the expected result in the 200 Year time period, more runs than expected end up with at least one member of RS apparently persisting for many years, and in

some cases even acquiring an extra Land Parcel. What is particularly strange is that these members of RS continue using the loss-making Land Use (*lu*1), but never go bankrupt.

To understand what is happening requires an in-depth analysis of the output of the model, focusing on Land Managers acquiring an extra Land Parcel. In one run (environment size 25 × 25), one such Land Manager, *lm*879, receives a Land Parcel, *lp*254 at the end of Year 2 and has initial wealth 0. Over the next 25 Years, *lm*879 chooses *lu*2 fifteen times, and *lu*1 ten times. The accumulated Wealth should be $(15 \times 0.4) - (10 \times 0.6) = 0$ at the end of Year 27 and *lm*879 should not have to sell *lp*254 — the code stipulates that this happens only if the Wealth is less than 0 (see Figure 1, in the 'Sale' box). However, accumulated floating point errors in adding 0.4 fifteen times and subtracting 0.6 ten times from the Wealth of *lm*879 means that instead of 0, *lm*879 has a Wealth of $-2.22045 \times 10^{-16}$. As a consequence, *lm*879 puts *lp*254 up for sale in the 'Sale' step, and receives 10000 units of Wealth. Since $2.22045 \times 10^{-16}$ is too small a quantity to subtract accurately from 10000 in floating point arithmetic, *lm*879 now has a Wealth of 10000.

At the same time in the 'Sale' step of Year 27, the owner of a neighbouring Parcel to *lp*254, *lp*195, has Wealth –0.6 after choosing *lu*1 in the 'Choose Land Use' step. Neighbouring Parcel *lp*195 is then also put up for sale. Once all Land Managers that need to have put their Land Parcels up for sale, new owners are sought for them. When writing the code for FEARLUS, the programmer (GP) assumed that no Land Manager who had put a Parcel up for sale could possibly be eligible to buy one since: (a) A Manager only sells a Parcel if its Wealth is less than zero, (b) on putting a Land Parcel up for sale, a Manager receives the Land Parcel price, and must therefore have Wealth less than the Land Parcel Price, (c) the only Land Managers eligible to buy Land Parcels are those with Wealth greater than or equal to the Land Parcel Price. Mathematically, this is correct, but as the analysis has shown so far, (a) and (b) do not quite hold when errors in floating point arithmetic are involved. Since *lp*254 has not yet been fully transferred, *lm*879 is still the owner of a neighbouring Parcel to *lp*195,[1] and since *lm*879 has Wealth 10000, it can also afford to buy it. (See the 'Purchase' box in Figure 1.) With probability 0.5, *lm*879 is selected over a potential new Land Manager as the new owner of *lp*195. A further programmer assumption, that only new Land Managers have no Land Parcels when becoming the owner of a Land Parcel (also mathematically correct for real numbers but not for floating point), means that *lm*879 loses no Wealth on acquiring *lp*195.

Land Managers that have had to sell all their Land Parcels in step 3 are removed from the 'Choose Land Use' schedule step and from the 'Accrue Wealth' schedule step. The software objects that store these Land Managers are not destroyed by the program since they may be used for reporting purposes. The situation at the end of Year 27 is therefore that *lm*879 owns *lp*195, has Wealth 10000, is still a valid object in the system, but does not appear in the list of Land Managers to be called in any of the steps of the simulation schedule where 'each Land Manager' appears in the UML diagram in Figure 1.

In Year 28, the new owner of *lp*254, *lm*2042, chooses *lu*1 in the 'Choose Land Use' step, and loses 0.6 units of Wealth in the 'Accrue Wealth' step. Meanwhile, *lm*879 has not made any selection of Land Use for *lp*195, and the latter's Land Use from Year 27, *lu*1, remains unchanged. In the 'Accrue Wealth' step, *lm*879 is not asked to harvest, as it is not scheduled to do so, and its Wealth remains unchanged despite *lu*1 being applied to its Land Parcel. In the 'Exchange Land Parcels' step, *lm*2042 puts *lp*254 up for sale, and *lm*879, with Wealth 10000, is an eligible new owner. Once again, the dice fall in *lm*879's favour, and it is selected as the new owner of *lp*254 rather than a potential new Land Manager. This time, *lm*879 does pay 10000 units of Wealth to own *lp*254, and the situation at the end of Year 28 is that *lm*879 owns Land Parcels *lp*195 and *lp*254, has zero Wealth, and is not on the schedule.

[1] Technically, *lm*879 has no members on its Land Parcels owned stack (they have all been removed during the 'Sale' step), but the owner property of *lp*254 is still set to *lm*879 (and is not changed until the Land Parcel is found a new owned in the 'Purchase' step). See Figure 1.

From then on, *lm*879 remains in this 'undead' state until the simulation is terminated, never making any Land Use decision on its accumulated estate of *lp*254 and *lp*195, never gaining or losing any Wealth, and never putting either Parcel up for sale. Both Parcels retain the same loss-making Land Use *lu*1 chosen by the owners who lost them to *lm*879.

The creation of 'ghost' Land Managers thus requires a number of coincidences to occur: (a) The real-valued Wealth of a Land Manager, *A*, must reach exactly zero. For this to happen *A* must choose *lu*2 one and a half times as often as *lu*1 despite both having equal probability of selection by a member of RS. Such sequences of Land Use selection are of length 5*i*, with 3*i* selections of *lu*2 and 2*i* selections of *lu*1, where *i* is a positive integer. A member of HR will not get into this situation because when it chooses *lu2* for the first time it will apply it from then on. Since a member of HR choosing *lu*1 the first time it makes a Land Use decision will go bankrupt, the surviving members of HR are all those that chose *lu*2 in their first decision. (b) The floating point error in the Wealth calculation of *A* must result in a negative number small enough in magnitude that when subtracted from 10000 the result is 10000. To get an idea of how likely this is, we checked all possible sequences of length 25 involving 15 selections of *lu*2 and 10 selections of *lu*1 for which the real-valued Wealth is greater than zero until the 25th selection. Of the 112227 possible such sequences, 7496 had the floating point Wealth exactly equal to zero, 7195 had floating point errors resulting in a positive Wealth, and 97536 (about 87%) had floating point errors resulting in a negative Wealth. All of the cases with negative Wealth had the potential to create ghost Land Managers. For shorter sequences, the proportion that have the potential to create ghost Land Managers is smaller, with 75% for sequences of length 20, 54% for length 15, 21% for 10, and 0% for 5. (c) A neighbouring Land Manager, *B*, must go bankrupt in the same Year. (d) Land Manager *A* must then be selected (with probability 0.5) over a potential new Land Manager as the new owner of *B*'s Land Parcel.

It is then reasonable to ask, given that some runs have no such problems, whether the problem is sufficiently rare that it can be ignored. To explore this, we ran a model consisting solely of members of RS. Given that these Land Managers choose at random with an equal probability of selecting *lu*1 and *lu*2, a time-series graph showing the number of times each Land Use is applied in each Year should show roughly equal applications of each Land Use. Instead the outcome, for any seed, is much like that shown in Figure 2, with the application of *lu*2 gradually declining as more and more ghost Land Managers occupy the space. This demonstrates that floating point errors have the potential to create a systematic bias in the behaviour of a model.

This effect has arisen purely because of errors in floating-point arithmetic accumulating through repeated additions of numbers (0.4 and –0.6) that cannot be exactly represented in binary, which allowed the model to enter states that were not anticipated by the programmer because they are not possible mathematically. Fortunately, none of the results published using this model (Polhill et al., 2001; Gotts et al., in press) have used such settings, and their validity is unaffected by problems with floating-point arithmetic documented here. This example does raise the issue, however, of how sensitive an agent-based model can be to floating point errors. Any branching statement in the agent's decision mechanism (here in the decision to sell land) containing a comparison expression involving a floating-point variable creates a high degree of nonlinearity with respect to that variable. As this example has shown, the 15 decimal places of accuracy provided by double precision variables was not enough. For now, we can avoid these problems through using parameters that can be exactly represented in binary, but remedial steps will need to be taken to manage errors in floating point arithmetic should a scenario require us to use parameters that do not meet this demanding requirement.

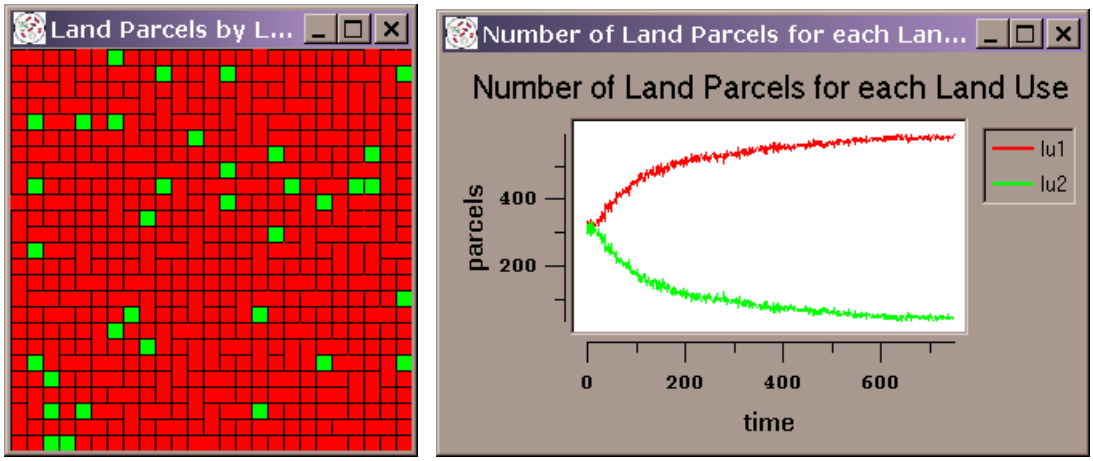



**Figure 2** — Systematic bias in a run of the FEARLUS model in Figure 1 using Subpopulation RS only. The space should thus contain a roughly equal proportion of *lu*1 (red) and *lu*2 (green) Land Uses, but by Year 750, the situation is as shown on the left. Black lines separate Land Parcels owned by different Land Managers, and the image shows a number of Managers owning two rather than one Land Parcels. The time-series graph on the right shows the gradual decline in the use of *lu*2 as more and more ghost Land Managers are created.

## 3 The Artificial Stock Market

The Artificial Stock Market (ASM; LeBaron et al., 1999) is a well-known agent-based model that has been used by other authors as a basis for their work (e.g. Chen & Yeh, 2002). In this model, agents have a choice between investing in a safe bond with a fixed interest rate, and a risky stock in limited supply with an autoregressive stochastic dividend $d_t$ calculated as per equation [1] (LeBaron et al., 1999):

$$d_t = \bar{d} + \rho\left(d_{t-1} - \bar{d}\right) + \mu_t \qquad [1]$$

where $\bar{d}$ is the average dividend (a model parameter), $\mu_t$ is a stochastic component sampled from a Normal distribution with zero mean and constant variance, and $\rho$ is the autocorrelation parameter: if 0 then $d_t$ and $d_{t-1}$ are uncorrelated, and if 1, $d_t$ and $d_{t-1}$ are maximally correlated.

Agents adapt their strategies for choosing how much to invest in a series of repeated cycles using a Genetic Algorithm. The ASM has been shown to replicate features in time-series from real stock markets, such as weak forecastability and volatility persistence (LeBaron et al., 1999, p. 1512). These phenomena cannot be explained by the well-established Efficient Market Hypothesis, under which prices only change when there is new relevant information, meaning that they are not forecastable at all and are less volatile than prices observed in real markets; from this Shleifer (2000) argues that, "More than news seems to move stock prices." (p. 20).

The source code for the ASM is written in Objective-C, and is available for download at http://prdownloads.sourceforge.net/artstkmkt/ (we have used version 2.2.1 — the most recent stable release at 28 April 2003). Having obtained the code, we created a baseline version using a separate random number generator for the stochastic component $\mu_t$ of the dividend. This is to prevent any other modifications we make from having a disproportionate effect on the model. The random number generator in the original code used to get a sequence of numbers for $\mu_t$ is also used for other purposes (e.g. in the GA). If the effect of other modifications we make is to change the number of times the random number generator is sampled in the GA or elsewhere, then this will affect the sequence of numbers for the dividend. With the dividend affected, it would not be possible to gauge whether there are any long-term effects on the behaviour of the model of the other modifications made to the code. The modification for the baseline is shown in Figure 3.

We then created two new versions to test the sensitivity of the ASM to errors in floating point arithmetic (Using Swarm release 2001-12-18 on a Sun Blade 1000 running Solaris 2.8). Version 1 modifies the implementation of equation (4) from LeBaron et al.'s paper (p. 1491) (Figure 4), and Version 2 modifies the implementation of equation (14) (p. 1496) (Figure 5). Neither modification should make any difference to the behaviour of the software according to the laws of mathematics. Version 1 computes (A/B) – (C/B) rather than (A – C)/B, and Version 2 computes Z + Y + … + A, rather than A + B + … + Z.

**Dividend.m**

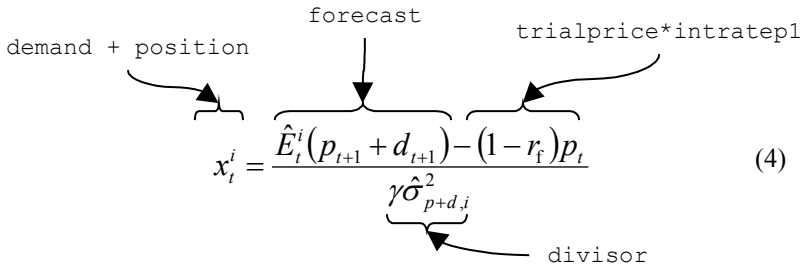
Original Code

```
26        normal=[NormalDist  create: [self getZone]
                  setGenerator: randomGenerator
                  setMean: 0 setVariance: 1];
```

Baseline

```
26        normal=[NormalDist  create: [self getZone]
                  setGenerator: [MT19937gen create:
                  [self getZone] setStateFromSeed: 456987123]
                  setMean: 0 setVariance: 1];
```

**Figure 3** — Modifications to the original code to create the baseline version with a separate random number generator for the dividend of the risky stock.




$$x_t^i = \frac{\hat{E}_t^i(p_{t+1} + d_{t+1}) - (1 - r_f)p_t}{\gamma \hat{\sigma}^2_{p+d,i}} \qquad (4)$$

**BFagent.m**

Baseline / Original Code

```
878       demand = -((trialprice*intratep1
                            - forecast)/divisor + position);
```

Version 1

```
878       demand = -(((trialprice*intratep1/divisor)
                          - (forecast/divisor)) + position);
```

**Figure 4** — Modifications required to create Version 1 of the ASM, involving a change to line 878 of the Objective-C source file BFagent.m. Since division is distributive over subtraction, this change should have no effect according to the laws of arithmetic.

**Specialist.m**

Baseline / Original Code

```
189         index = [agentList begin: [self getZone]];
190         while ((agent = [index next]))
191           {
193             demand = [agent getDemandAndSlope: &slope
                                          forPrice: trialprice];
195-196         if (demand > 0.0) bidtotal += demand;
197-198         else if (demand < 0.0) offertotal -= demand;
200           }
```

Version 2

```
189         index = [agentList begin: [self getZone]];
190         [index setLoc: End];
191         while ((agent = [index prev]))
192           {
194             demand = [agent getDemandAndSlope: &slope
                                          forPrice: trialprice];
196-197         if (demand > 0.0) bidtotal += demand;
198-199         else if (demand < 0.0) offertotal -= demand;
201           }
```

**Figure 5** — Modifications made to the ASM for Version 2 highlighted in bold. (Some irrelevant lines, which are the same in both versions, have been removed.) Essentially, the sum computed in equation (14) in LeBaron (1999) to clear the market for the risky stock is totalled in reverse order of the agents. Since addition is associative, the order the agents are called to make the sum should not make any difference to the behaviour of the model according to the laws of arithmetic.

The results are summarised in Figure 6. Using a graph plotted at a resolution of 250 time steps, the baseline, Version 1 and Version 2 show a difference in the behaviour of the volume at about the 6000 cycle point, and despite having the same dividend, differences continue to be observed thereafter.

The statistical properties of the output that the authors use to claim a match with properties of real markets seem by eye not to be affected. Formal verification that the statistical properties are unaffected is the potential subject of future work. Though unlikely, it is not inconceivable for the verification to fail: rounding errors in floating point numbers are not random (Higham, 2002), leaving open the possibility of introducing a bias through implementing an equation one way rather than another. This may be related to the systematic bias seen in the RS-only run of the FEARLUS model discussed above.

The fact that implementing the mathematical formulae in LeBaron et al.'s paper in different but mathematically equivalent ways changes the output of the model raises a question mark over any exact replication of their work. At the platform level, compiler optimisation settings can affect the way that expressions are evaluated (Goldberg, 1991), meaning that potentially, different outcomes could occur on different machines even when using the same source code. However, the modifications introduced were intended to illustrate the possible effect of implementing the ASM from its description in the literature rather than by using the authors' source code. If nothing else, the differences in behaviour highlight how critical making source code available to other researchers can be, since otherwise exact replication of results would be difficult. The source code for the popular and oft-cited agent-based model SugarScape (Epstein & Axtell, 1996) is not publicly available, for example, though the book contains many equations that have the potential to introduce errors in floating point arithmetic given appropriate parameter settings, e.g. the

pollution formation rule $\mathbf{P}_{\alpha\beta}$ (p. 47), the agent inheritance rule **I** (p. 67), and the agent welfare function (p. 97).

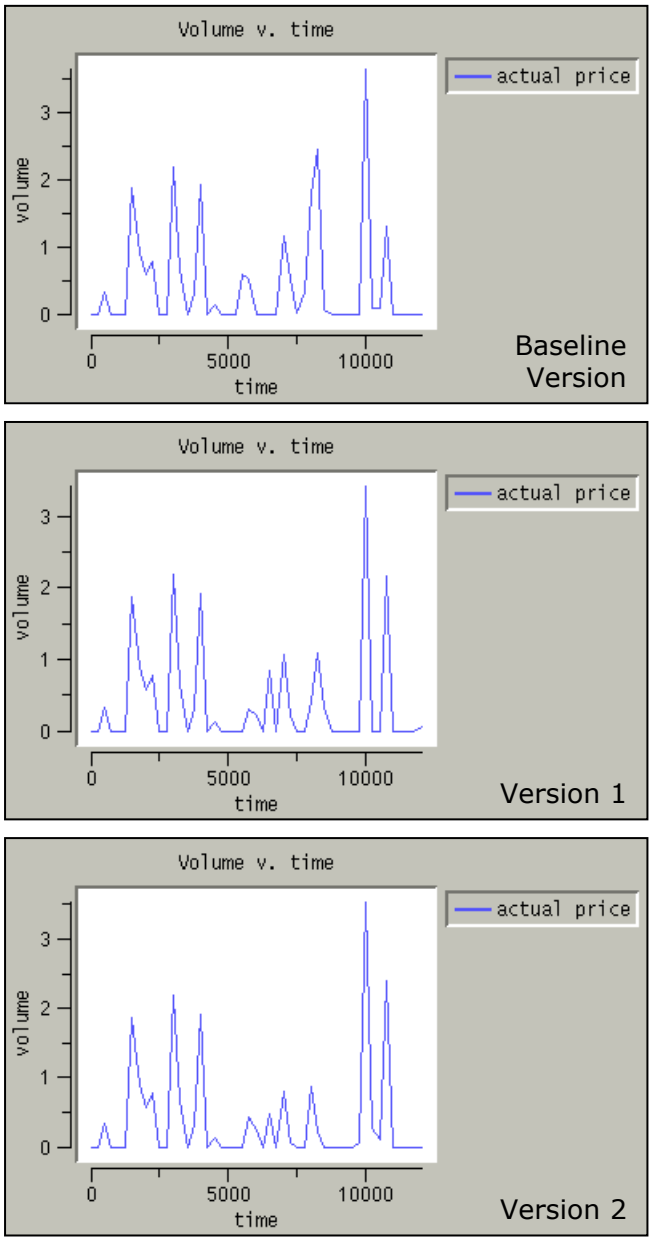



**Figure 6** — Divergence of behaviour in various versions of the ASM. The same output is seen for the first 6000 cycles, but thereafter there are differences in the number, height and width of peaks.

## 4 Floating point arithmetic and the IEEE 754-1985 standard

A real number $n$ can represented in any base $b$ without loss of precision using two variables, a fraction, $f$, also a real number, known as the mantissa or significand, and an integer exponent, $e$, such that $n = fb^e$, with $f$ being smaller in magnitude than $b$.

In a computer, there are a discrete set of values that the binary significand can take, meaning that most fractions are not exactly representable. For example, 0.1 (in decimal) is not an exactly representable number (Wallis, 1991a). Indeed, of all the fractions $0.1i$, $i = 1, \ldots, 9$, only 0.5 is exactly representable. An arithmetic operation (plus, minus, multiply, divide) on two floating point numbers is also not necessarily going to result in a number that is exactly representable (e.g.

1 / 3). There are thus three potential sources of error in a single arithmetic calculation: conversion from a decimal to a binary number for each operand, and a rounding error from the exact result to a representable result given the limitations on the significand.

Historically, computer manufacturers implemented floating point arithmetic in various ways, creating difficulties with porting code. Programs that worked in one machine could cause exceptions to be raised on another, even with simple operations such as setting $y = -x$ or $y = x / x$ (Wallis, 1991b). The IEEE 754 standard (IEEE, 1985) changed this, and now it is unusual to find a non-conforming floating-point unit in a computer.

The IEEE standard provides strict definitions for 32-bit (single) and 64-bit (double) precision floating point formats, and a more flexible definition for extended formats. It stipulates that the accuracy of arithmetic operations be such that the nearest representable number to the infinitely precise result be delivered to the destination.[2] It requires that the user be able to set the rounding mode to one of four possibilities: round-to-nearest even (the default), round towards positive infinity, round towards negative infinity, and round-to-zero (truncate). It further requires that conforming architectures generate trappable exceptions in various cases where the result of a calculation has not been exactly representable in the destination format.

The standard is not designed to enable precise reproducibility in all platforms, however. The format of the destination is not stipulated to be the format suggested by the type of the variable. For example, Intel floating-point units work using 80-bit double extended format by default, even if a variable is defined by a programming language to be a 64-bit variable. Although the floating-point unit can be configured to do calculations in 64-bit arithmetic (Intel, 2003), compilers do not necessarily issue instructions to do this (e.g. Borland C compiler, and the GNU C compiler in Cygwin). For example, the calculation $2^{53} / (2^{53} - 1)$ is, in binary, the repeating fraction $1.0\ldots_{52}\ldots010\ldots_{52}\ldots01$ …, where "$0\ldots_x\ldots0$" is replaced with $x$ zeros. The double precision 64-bit format has 53 bits for the significand, and therefore the nearest representable number is $1.0\ldots_{51}\ldots01$, rounding up the least significant bit. However, in Intel's 80-bit arithmetic, with 64 bits for the significand, the result delivered to the destination is $1.0\ldots_{52}\ldots010\ldots_{10}\ldots0$, which is then rounded to the 64-bit variable as $1.0\ldots_{51}\ldots00$. Nevertheless, Intel's floating point architecture is fully IEEE 754 compliant.

One platform that doesn't fully comply with IEEE 754 is Java (Kahan & Darcy, 2001). Java provides no facilities for checking exception flags for any errors in floating point computations, and no facilities for changing the rounding mode. Since many popular agent-based modelling platforms are written in Java (Ascape, RePast) or provide Java interfaces (Swarm), this is bound to be contentious. Without these facilities, Java programmers are severely impaired: they cannot easily check whether or not a series of calculations has caused an inaccurate result, and they cannot directly implement interval arithmetic (see below). It is possible to get equivalent functionality in Java, but not without considerable effort on the part of the programmer (Darcy, pers. comm.).

# 5 Interval arithmetic

A comparison of various techniques for handling errors in floating point errors concluded that interval arithmetic offered a safe approach (Polhill et al, subm.) in that it could be used to provide a warning when a comparison operator might deliver the wrong result due to accumulated floating point errors. The idea behind using interval arithmetic is always to ensure that the true value of a calculation is within known bounds. Of the available forms that interval arithmetic can take, Ullrich & von Gudenberg (1991) suggest that those representing the bounds explicitly are most

[2] However, this is the precise result of operating on the floating point operands, not on the original numbers the operands may have been rounded from. Thus, for example, the result of $0.4 \times 0.1$ in IEEE 754 floating point arithmetic is not the nearest representable number to 0.04, but the nearest representable number to the product of the nearest representable number to 0.4 and the nearest representable number to 0.1.

successfully implemented in a computer. In particular, it is possible to use the IEEE 754 stipulated rounding mode functions to compute a minimum and maximum result for each calculation: one computed using round towards negative infinity, and the other using round towards positive infinity. Let an interval be represented by [$r$, $s$], where $r \leq s$. Let $\lceil x \rceil$ represent the smallest floating point number that is not less than $x$, and $\lfloor x \rfloor$ represent the largest floating point number that is not more than $x$, then the arithmetic operators are defined (Alefeld & Herzberger, 1983):

$$[r_1, s_1] + [r_2, s_2] = [\lfloor r_1 + r_2 \rfloor, \lceil s_1 + s_2 \rceil] \quad [2]$$

$$[r_1, s_1] - [r_2, s_2] = [\lfloor r_1 - s_2 \rfloor, \lceil s_1 - r_2 \rceil] \quad [3]$$

$$[r_1, s_1] \times [r_2, s_2] = [\min\{\lfloor r_1 \times r_2 \rfloor, \lfloor r_1 \times s_2 \rfloor, \lfloor s_1 \times r_2 \rfloor, \lfloor s_1 \times s_2 \rfloor\}, \max\{\lceil r_1 \times r_2 \rceil, \lceil r_1 \times s_2 \rceil, \lceil s_1 \times r_2 \rceil, \lceil s_1 \times s_2 \rceil\}] \quad [4]$$

$$[r_1, s_1] \div [r_2, s_2] = [\min\{\lfloor r_1 \div r_2 \rfloor, \lfloor r_1 \div s_2 \rfloor, \lfloor s_1 \div r_2 \rfloor, \lfloor s_1 \div s_2 \rfloor\}, \max\{\lceil r_1 \div r_2 \rceil, \lceil r_1 \div s_2 \rceil, \lceil s_1 \div r_2 \rceil, \lceil s_1 \div s_2 \rceil\}] \quad [5]$$

There are then two forms for the comparison operators, echoing modal logic (Kripke, 1963): a ‘necessarily’ form where the comparison is true iff it is true for all pairs of members of each operand, and a ‘possibly’ form, where the comparison is true iff it is true for one or more pairs of members of each operand (Alefeld & Herzberger, 1983). Let $>_L$ represent ‘necessarily greater-than’ and $<_M$ represent ‘possibly less-than’, and similarly for the other comparison operators. Then:

$$[r_1, s_1] >_L [r_2, s_2] \Leftrightarrow r_1 > s_2; \quad [r_1, s_1] >_M [r_2, s_2] \Leftrightarrow s_1 > r_2 \quad [6]$$

$$[r_1, s_1] <_L [r_2, s_2] \Leftrightarrow s_1 < r_2; \quad [r_1, s_1] <_M [r_2, s_2] \Leftrightarrow r_1 < s_2 \quad [7]$$

$$[r_1, s_1] =_L [r_2, s_2] \Leftrightarrow (r_1 = s_2) \wedge (r_1 = r_2) \wedge (s_1 = s_2); \quad [r_1, s_1] =_M [r_2, s_2] \Leftrightarrow (s_1 \geq r_2) \wedge (r_1 \leq s_2) \quad [8]$$

In a section of code where $x > y$ leads to an action, it is possible to ensure that, regardless of floating point errors, the action only takes place when it should by replacing $x$ and $y$ with intervals containing the true values, and performing the action if $x >_L y$. To check that the action is only not executed when it should not be, then after $x >_L y$ returns false, there should be a further check whether $x >_M y$. If the latter is true, a warning should be issued to the effect that there is no longer certainty that the program has executed the correct instructions in accordance with the model design. A similar exercise for other comparison operators ensures that no action takes place unless it necessarily should and no action does not take place that possibly should without a warning being given. A model run can then be known to have executed without deviating from the design due to floating point errors given the absence of a warning.

Implementing intervals in FEARLUS should be a relatively trivial exercise, since FEARLUS only makes use of the basic arithmetic operators in calculations. However, this has not turned out to be the case. There are a number of places in the FEARLUS code where a weighted selection is made between various alternatives, in which the chance of selecting an alternative is given by its weight divided by the sum of the weights of all the alternatives. A trivial example is in selecting which Subpopulation a newly created Land Manager will belong to. This is not an issue in the example above because the probability of belonging to HR or RS is 0.5, which is exactly representable in binary floating point. If there were three subpopulations to chose from with equal probability, then an issue would arise because 1/3 would have to be represented using the interval [ $\lfloor 1/3 \rfloor$, $\lceil 1/3 \rceil$ ]. More generally, however, consider a choice between two alternatives U and V, where the

weight for U is $u = [1, 2]$, and that for V is $v = [2, 3]$. Looking at the numbers, the probability of choosing U, $P(U)$, has then a minimum of 0.25 when $u = 1$ and $v = 3$, and a maximum of 0.5 when $u = v = 2$. Similarly $P(V)$, is in the interval [0.5, 0.75]. Since there is uncertainty in their relative probabilities, it is not clear how to choose between U and V in an unbiased manner.

In the Artificial Stock Market, 250,000 cycles of the model are used to enable agents to learn before generating the time-series on which the analysis was based (LeBaron et al, 1999, p. 1499.) Given that simply rearranging terms in a single mathematical expression changed the behaviour of the Volume as early as 6000 or so cycles into the model, it is conceivable that after 250,000 cycles, there could be a considerable number of comparisons in which the outcome is uncertain were interval arithmetic to be used.

Approaches to managing floating point errors such as interval arithmetic have various properties that may need to be considered in evaluating them. Approaches such as interval arithmetic are *safe* in that they ensure the model behaves as though it is using real numbers until a warning is given. This might impose an unacceptable constraint on the model in that it will not be possible to run it for the required time period without a warning being given. A less stringent requirement could be *robustness*, in which the floating point error management approach could provide some guaranteed protection against the model entering unanticipated or invalid states through floating point errors. In FEARLUS, for example, ghost Land Managers could be prevented by explicitly representing the state of a Land Manager as 'bankrupt' and then preventing bankrupt Land Managers from buying Land Parcels. However, this would not protect against Land Managers with zero real-valued wealth from going bankrupt when they shouldn't. Another potential property an approach might have is *platform independence*, which would facilitate exact repetition of results. At the very least, however, floating point error management techniques should not allow a systematic bias in the behaviour of the model. Polhill et al. (subm.) contains a comparison of various approaches.

# 6 Conclusion

In an ideal world, floating point arithmetic should not be something that agent-based modellers need to trouble themselves with. This paper has demonstrated that the world is far from ideal, through two examples from published agent-based models. The first example illustrates how branching in agent decision mechanisms based on comparisons involving floating point variables can lead to emergent effects with an entirely unwelcome element of 'surprise' because of the high degree of nonlinearity this introduces, and the seeming consequent demand for infinite precision. The second example shows the necessity for authors to release their source code in public domain if there is to be any chance of other researchers repeating their results with any degree of precision. Mathematical equations in models can be implemented in a number of different ways in a computer program, and although these may be mathematically equivalent, they are not equivalent in terms of floating point arithmetic.

David Hales, in a presentation at Marseille (Edmonds & Hales, 2003), stated that reimplementation of agent-based models should not use the original source code, as they would then introduce the same artefacts. A distinction needs to be made between reimplementation and repetition in this context. A supposed advantage of using computer simulation is that experiments can be exactly repeated, allowing the confirmation of the results obtained by other authors, artefacts and all. Such functionality is not typically available in the natural sciences — it is like being able to reuse the apparatus and samples to take exactly the same measurements as in the original experiment. Reimplementation is, of course, a more rigorous test of a reported effect, but this does not necessarily mean that repetition is without value. Repetition is an exercise researchers can undertake themselves with relatively little effort. Compiling the model on different platforms (e.g. a dual-boot Linux/Windows PC), at least provides a check that results are not dependent on such things as operating system, compiler and software library versions.

Traditionally, a number of heuristics have been used for dealing with errors in floating point arithmetic, such as using small tolerances when making comparisons, or writing the result of an operation to a string buffer with a degree of accuracy not greater than that provided by the floating point numbers and then copying the result out of the buffer into the floating point variable. Whilst such measures provide some ability to cope with the problems of floating point errors, they do not provide any certainty that a given run has strictly followed the model design (Polhill et al., subm.). Such certainty is provided using interval arithmetic: it can be known at the point of comparison that, even allowing for floating point errors, the correct course of action is being taken according to the design of the model. However, even this is no panacea: use of intervals creates difficulties for certain algorithms such as weighted selection between alternatives and sorting; it also limits the number of cycles a model can be run before accumulated floating point errors mean there is uncertainty about whether the model is behaving according to the design.

# 7 Acknowledgement

This work is funded by the Scottish Executive Environment and Rural Affairs Department.

# 8 References

Alefeld, G., & Herzberger, J. (1983) *Introduction to Interval Computation*. New York: Academic Press.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999) *The Unified Modeling Language User Guide*. Addison-Wesley.

Chen, S-H., & Yen, C-H. (2002) On the emergent properties of artificial stock markets: the efficient market hypothesis and the rational expectations hypothesis. *Journal of Economic Behavior & Organization* **49** 217-239.

Darcy, J. D. *Personal communications*. 2 May 2003 and 29 July 2003.

Edmonds, B., & Hales, D. (2003) Replication, replication and replication: Some hard lessons from model alignment. *Model to Model International Workshop, GREQAM, Vielle Charité, 2 rue de la Charité, Marseille, France. 31 March - 1 April 2003*. pp. 133-150.

Epstein, J. M., & Axtell, R. (1996) *Growing Artificial Societies: Social Science from the Bottom Up*. Cambridge, MA: MIT Press.

Goldberg, D. (1991) What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys* **23** (1) pp. 5-48. Reproduced in Sun's Numerical Computing Guide (Appendix D) and available on-line at http://portal.acm.org/citation.cfm?doid=103162.103163

Gotts, N. M., Polhill, J. G., Law, A. N. R. (in press) Aspiration levels in a land use simulation. *Cybernetics & Systems*. To appear.

IEEE (1985) *IEEE Standard for Binary Floating-Point Arithmetic*, IEEE 754-1985, New York, NY: Institute of Electrical and Electronics Engineers.

Intel (2003) *IA-32 Intel Architecture Software Developer's Manual Volume 1: Developer's Manual*. http://www.intel.com/design/Pentium4/manuals/.

Higham, N. J. (2002) *Accuracy and Stability of Numerical Algorithms, 2nd Edition.* Philadelphia, USA: SIAM.

Kahan, W., & Darcy, J. D. (2001) How Java's floating point hurts everyone. Originally presented at the *ACM 1998 Workshop on Java for High-Performance Network Computing*, *Stanford University*. Revised and updated version (5 Nov 2001, 08:26 version used here) available for download at http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf.

Kripke, S. (1963) Semantical considerations on modal logic. *Acta Philosophica Fennica* **16**, 83-94.

LeBaron, B., Arthur, W. B., & Palmer, R. (1999) Time series properties of an artificial stock market. *Journal of Economic Dynamics & Control* **23** 1487-1516.

Polhill, J. G., Gotts, N. M., & Law, A. N. R. (2001) Imitative and nonimitative strategies in a land use simulation. *Cybernetics & Systems* **32** (1-2) 285-307.

Polhill, J. G., Izquierdo, L. R., Gotts, N. M. (subm.) What every agent based modeller should know about floating point arithmetic. *Submitted to Environmental Modelling and Software*. 20 August 2003.

Shleifer, A. (2000) *Inefficient Markets: An Introduction to Behavioral Finance*. Oxford UP.

Ullrich, C., & von Gudenberg, J. W. (1990) Different approaches to interval arithmetic. In Wallis, P. J. L. (ed.) *Improving Floating Point Programming.* Chichester, UK: John Wiley & Sons, ch. 5.

Wallis, P. J. L. (1990a) Basic concepts. In Wallis, P. J. L. (ed.) *Improving Floating Point Programming.* Chichester, UK: John Wiley & Sons, ch. 1.

Wallis, P. J. L. (1990b) Machine arithmetic. In Wallis, P. J. L. (ed.) *Improving Floating Point Programming.* Chichester, UK: John Wiley & Sons, ch. 2.