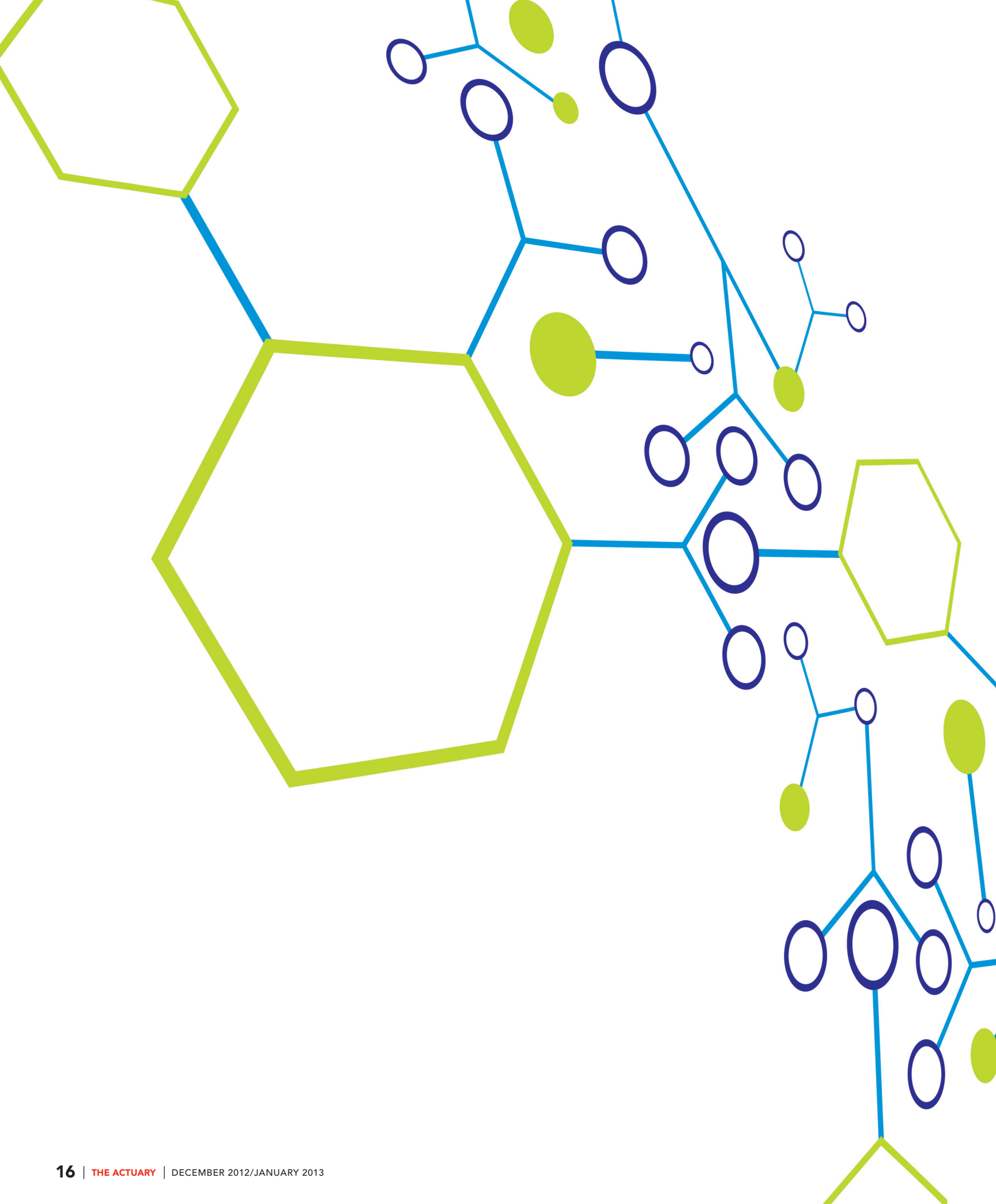
SOCIETY OF ACTUARIES

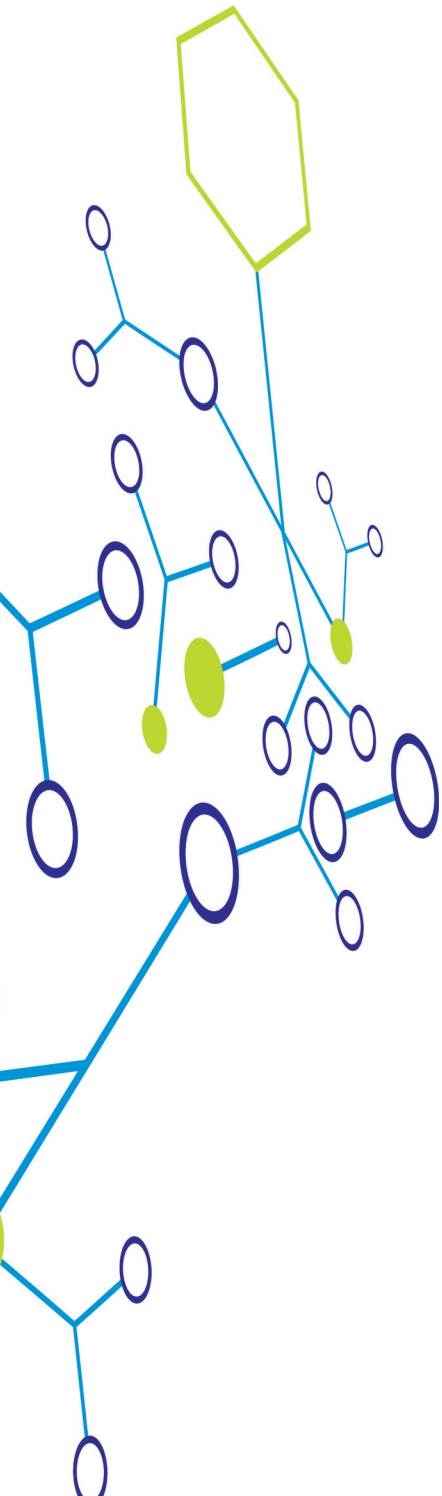
Article from:

The Actuary Magazine

December 2012/January 2013 – Volume 9 Issue 6



COMPLEXITY OR SIMPLICITY?



GENETIC ALGORITHMS SOUND VERY COMPLICATED. YET, A GENETIC ALGORITHM IS JUST ANOTHER TECHNIQUE TO FIND SOLUTIONS TO PROBLEMS. THIS ARTICLE DESCRIBES JUST HOW SIMPLE THIS TECHNIQUE CAN BE. BY DAVE SNELL

The opportunity to share my enthusiasm for complexity sciences at SOA annual meetings, health meetings, the Life & Annuity Symposium and lots of regional meetings has been a lot of fun. But then, after a keynote presentation I gave at the Actuarial Research Conference, an attendee came up to me and said that she had a suggestion for a major improvement. She said that instead of "Complexity," I should rename the title "Simplicity." She shared that she is an expert in her field of actuarial focus; and although she enjoyed my presentation, she almost skipped it because she did not want to have to listen to yet another complex actuarial topic outside her immediate area of expertise.

I thought a lot about what she said. She was correct. I was wrong. This is not more difficult than classical, deterministic, actuarial techniques. It is simpler. It is a way of solving problems with simple rules and building blocks. When you stop and seriously think about it, the complexity science techniques are actually more intuitive than some of our classical deterministic actuarial techniques. Complexity science seems to be an umbrella term for many topics; but in this article I shall try to explain just two of them, deterministic chaos and genetic algorithms, with fairly simple examples.

Deterministic chaos is a topic that sounds daunting but really is more simple than complex. My favorite example of this is the logistic equation for population growth that Pierre François Verhulst proposed in 1838. The famous actuary Benjamin Gompertz proposed a similar model for human mortality in 1825. Darwin noted a similar growth pattern in action on isolated islands he visited on the famous sea voyage in the 1830s that led to his theory of evolution. Assume you have a population of some animal on an island with no natural predators and a surplus of food. An actuary would be able to project that the population would, initially, increase rapidly—in fact, it would likely increase exponentially. As we all know though, exponential growth is seldom sustainable. Eventually, the food supply starts to become scarce and population growth is limited accordingly. Verhulst determined that the population growth rate at time 't+1' is going to be some constant, R (related to the Malthusian parameter of maximum growth rate), times the rate at time 't' times the quantity '[1 - the rate at time t]'. More concisely,

$$P(t+1)=R * P(t) * [1 - P(t)] \text{ where } -1 <= P(t) <= 1$$

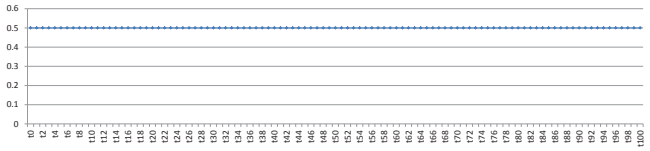
[Pierre François Verhulst's Logistic Equation—originally proposed in 1838]

The Logistic Map: X(t=1)=R * X (t)* [1 – X(t)]

Note: 0≤X(t) ≤ 1

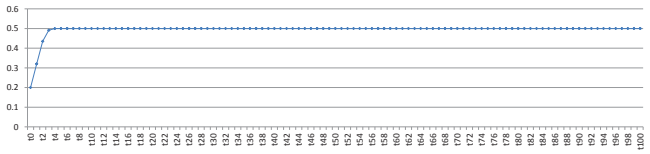
x(t+1)=Rx(t)[1-x(t)] with R=2 and x(0)=0.5

X(91) ... X(100): .5, .5, .5, .5, .5, .5, .5, .5, .5, .5



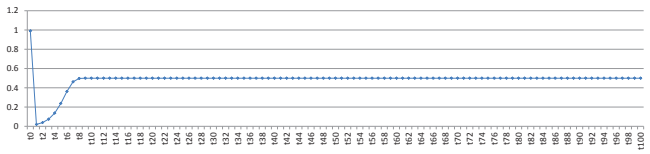
x(t+1)=Rx(t)[1-x(t)] with R=2 and x(0)=0.2

X(91) ... X(100): .5, .5, .5, .5, .5, .5, .5, .5, .5, .5



x(t+1)=Rx(t)[1-x(t)] with R=2 and x(0)=0.99

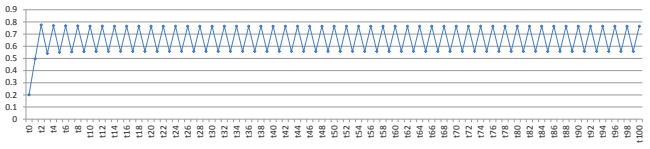
X(91) ... X(100): .5, .5, .5, .5, .5, .5, .5, .5, .5, .5



For R=2, the starting point is unimportant and there is a single attractor of .5

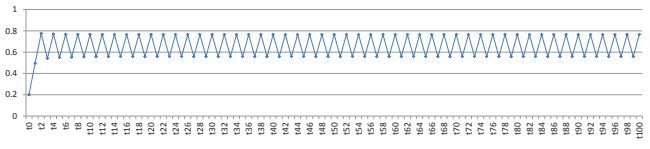
x(t+1)=Rx(t)[1-x(t)] with R=3.1 and x(0)=0.5

X(91) ... X(100): .765, .558, .765, .558, .765, .558, .765, .558, .765, .558



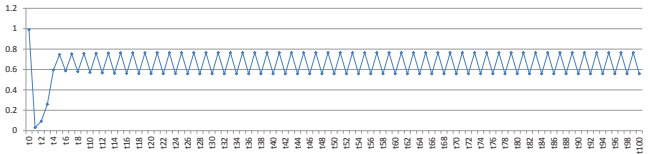
x(t+1)=Rx(t)[1-x(t)] with R=3.1 and x(0)=0.2

X(91) ... X(100): .558, .765, .558, .765, .558, .765, .558, .765, .558, .765



x(t+1)=Rx(t)[1-x(t)] with R=3.1 and x(0)=0.99

x(91) ... X(100): .765, .558, .765, .558, .765, .558, .765, .558, .765, .558



For R=3.1, the starting point is unimportant and there are two attractors of .558 and .765

Figure 1. Logistic Map with R=2 and R=3.1

When R is small, say R=2, it does not matter what starting value you choose for x(0). The resulting iterations will always converge to a single attractor of 0.5. As R increases, the number of attractors doubles (according to Feigenbaum's constant: 4.6692016), as shown in Figure 1, and the later iterations oscillate between them.

This seems orderly. Some grand mathematical process is controlling the affair; and we might expect the trend to continue. We would be wrong.

Looking at Figure 2, we see that once you reach R=4, just a tiny change in one of your assumptions may cause an undetermined effect on the validity of your model. The two graphs are somewhat similar; but there are definite differences in some areas. Everyone has heard of the butterfly effect. Here is the butterfly effect in action in very basic algebra. Keep in mind the only thing that caused the two graphs to differ so noticeably at the later durations is a starting assumption difference beyond the trillion decimal place. That's 0.2 versus 0.2000 blah, blah, blah, 001.

The implication for actuarial models, which may be far more complicated than the logistic equation, is that very small variations in starting values may have huge unforeseen consequences. Try stopping one of your projection models after 10 years and input the numbers you have at that point into the same model. Do you get the same results at the ending year? What if you reentered your output each year as the next year's starting values? The results 50 years from now may be significantly different from your expectations.

Deterministic Chaos

The Logistic Map: X(t+1)=R * X (t)* [1 – X(t)]

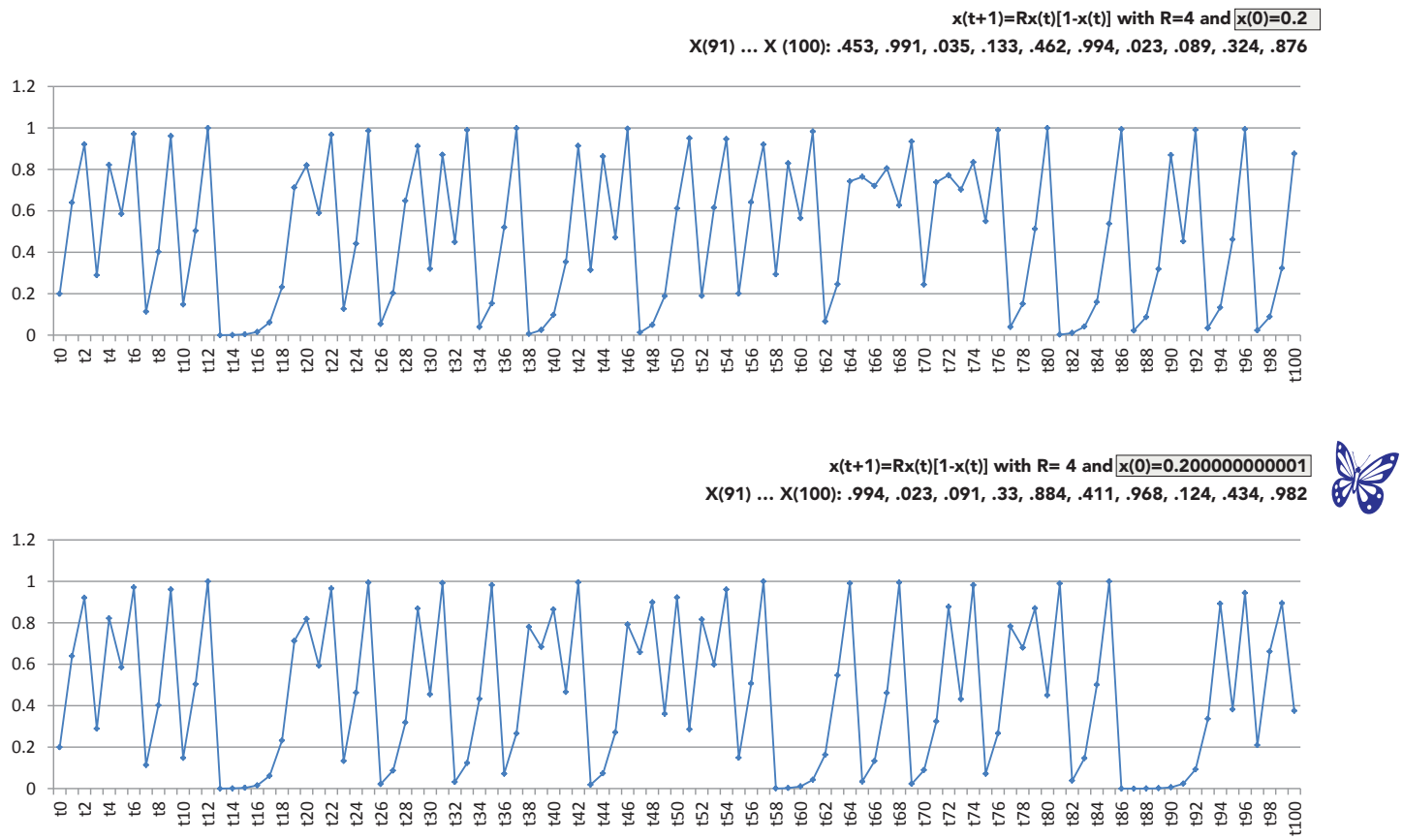


Figure 2. Logistic Map with R=4 — Deterministic Chaos Emerges

Deterministic chaos merely means that a simple non-periodic system may be completely determinable over the short term but it becomes unpredictable past its horizon of predictability, and according to James Gleick, a pioneer in chaos theory, "any physical system that behaves non-periodically is unpredictable." Common examples include weather predictions (highly accurate over a few days, seldom accurate past a couple of weeks) and financial markets—perhaps even actuarial financial models!

Read more about the science of chaos theory in *Chaos: Making a New Science*, by James Gleick.[1]

Genetic algorithms sound very complicated. Yet, a genetic algorithm is just another technique to find solutions to problems. It uses simple rules, comparative scoring, and selective modifications for the subsequent iterations. Even bacteria effectively employ them (to evolve stronger bacteria that are resistant to antibiotics). Is a fifth grader bacterium smarter than an actuary? In my workshops (with Brian Grossmiller, a kindred spirit in complexity sciences) on genetic algorithms, I often start with basic genetics: genes, alleles, mitosis and meiosis; but that is probably too complicated.[2]

A genetic algorithm can be ideal for a situation where:

1. You have no direct algorithm for an exact solution (or an exact solution would be too complicated or too time-consuming);

2. The number of potential solutions is too large to try them all; and
3. Solutions can be scored such that you can compare the value of solution X versus solution Y and easily see which is better.

In a genetic algorithm, we usually assign a set of actions or conditions and then we evolve better and better sets in a process that mimics the evolutionary process. The genetics terms and metaphors are historical. John Holland introduced them back in 1975 when he first described genetic algorithms in his book, *Adaptation in Natural and Artificial Systems*.[3] He was impressed by the speed at which species had evolved, as evidenced by the fossil record, and he developed ways for us to emulate evolution as a technique for solving problems too time-consuming by other means.

Let's take a simple example that Brian and I used in our workshop. Say you want to manage a health care provider system to reduce costs and still provide adequate coverage for the plan participants. In our example, Brian had empirical cost data from more than 3,000 provider groups. Each provider group offered one or more specialty services. These might range from acupuncture to urology. Each specialty has a relative cost (e.g., the average charge from an ophthalmologist might be higher than that from a pediatrician) and each provider group also has a relative cost (provider group 5 may be in a fancy location and charge, on the whole, much more than provider group 253).

We want to lower costs while maintaining some desired level of access to at least some minimum number of choices for each of the various specialties. If we only wanted to minimize cost, this would be easy. We could just include the lowest cost provider groups and exclude the higher priced ones. Unfortunately, some of those needed specialties are not available from the lowest cost provider groups. We might also want to include relative quality of services measures, based on patient feedback or a number of other comparative criteria.

A provider group will either be in our network or not. These are the only allowable choices so we can represent our set as a long string (analogous to the long DNA strands we have in each of our human cells) of zeros and ones representing whether a provider group is in (one) or out (zero) of our network.[4]

If we randomly generated 100 solution sets they might be as shown in Figure 3 below.

Here, we see that solution set 1 includes provider groups 2, 3 and 5 while solution set 100 includes provider groups 3, 5 and 6. I also added the relative scores for these sets (the score takes into account cost and coverage and potentially lots of other criteria) in the last column. Details of the scoring algorithm are unimportant here (but you can see them in the referenced Excel workbook). The point is that via the score we have an easy way to see if one set is better than another one.[5]

Given a situation like this, an actuary might try to figure out an exact solution; but the number of simultaneous equations (not necessarily linear) is immense (3,000+) and the result might take a very long time and effort. Alternatively, the option of trying out each potential set is unthinkable. There are 2^{3000} possible solution sets; and if you are suspecting that is a big number, you are correct—big time! The number $2^{3000} > 10^{903}$, but the number of atoms in the known universe is around 10^{82} and the number of seconds since the beginning of time (the Big Bang) is around 10^{17}; even multiplying these numbers together we are not even close to the number of possible solution sets.[6] Clearly, we do not have time to try comparing all the solutions. Yet, we can easily and quickly check to see if one potential solution set is better (i.e., gets a lower score) than another one.

This is a perfect place to try a genetic algorithm approach.

PROVIDER GROUP	1	2	3	4	5	6		3000	SCORE (lower is better)
SET 1	0	1	1	0	1	0		0	0.9873
SET 2	1	1	0	0	0	1		1	0.8206
SET 100	0	0	1	0	1	1		0	1.1393

Figure 3. Sample Sets of Provider Groups for a Health Care Network

For More Info

IF YOU WOULD LIKE MORE INFORMATION on this topic, please visit www.soa.org, click on the Presentation Archives link under the Professional Development tag, and then click on the Purchase Webcast and Virtual Session Recordings link. Or you can use the QR code provided here.

The following is a description of one of the sessions available for purchase: "This session will contain discussions on potential risks that may be emerging, and how to identify these risks and understand them. The discussion will include how to triage the identified emerging risks based on what would have a material impact on an organization. This session will leverage the emerging risk survey published earlier in the year by the Joint Risk Management Section.

At the conclusion of the session, attendees will be able to identify emerging risks to their organization, evaluate the impact on their organization and, ultimately, explain the resultant impact to their stakeholders."

We will start out with 100 potential provider sets.[7] Each of them will have a gene string of 3,000 genes, and each gene can be only a zero or a one (each gene represents the inclusion or exclusion of a specific provider group). Our first step will be to randomly assign zeros and ones to all of the genes in every set. For example, let's say we randomly generated the sets shown in Figure 3.

Then, we rank those sets according to their scores. The winners (lowest scoring sets) in this generation will not be terribly impressive. After all, they were randomly generated sets—no brainpower needed here.

Next, we'll decide upon some way to determine mating rights so that we can use these sets to spawn a new, hopefully smarter, next generation. Oh, that word "hopefully" is bothersome, isn't it? We don't want to risk our next generation being dumber; but if all we do is combine randomly created sets together—even the brighter ones—we could get bad combinations and our "species" might devolve instead of evolve. How does nature handle this?

In nature, the various members of a generation do not all live the same length of time. In essence, some die young and never get to have children; some have children and then die (perhaps even in childbirth); and some live on to coexist with the new kids on the block. In genetic algorithms, we call these latter ones "elites." In order to guarantee that our generations do not get dumber instead of smarter, we will specify that a certain number of the sets in generation 1 (the favored ones) will self-replicate into generation 2.

For now, let's set the percentage of elites to 10 percent. That means when we get around to building generation 2 from generation 1, the top 10 sets (of our 100 sets) of generation 1 will copy over exactly. Our next task then is to figure out how to generate the remaining 90 sets of the new generation 2.

Again, let's look at what happens in genetics. In most species of mammals, the biggest, or prettiest, or smartest, or strongest member or members of the group are deemed the most attractive mates for reproduction. For example, the norm in a kangaroo mob is that only the dominant male of the entire mob gets to mate with the various females. Among humans, we are not quite that strict (although in history, emperors and kings had many, many mates) and most folks have a chance at finding a mate; but still the smartest, richest, strongest or prettiest seem to have more choices.[8]

We will arbitrarily say that the top 20 percent of the sets will be the parents of the next generation. This will, of course, include our elites (our top 10 percent). I have to emphasize that these percentages are not scientifically determined and this is not the usual method I would employ; but it works fine for this example, and it shows that there is still a lot of "art" in the making of genetic algorithms.[9]

Now that we have established our pool of potentially preferred parents we can address the actual reproduction process. In nature, a child gets a DNA string that is composed of pieces from two parents. Let's say that we choose sets 2 and 5 as the parents. Then, on a gene-by-gene basis, each gene of the child will have either a copy of the corresponding gene from set 2 or the corresponding gene from set 5. The child will end up as some combination of sets 2 and 5. (See Figure 4 on page 22.)

OK, that works. However, we are limiting our possibilities here because of our experience. When Ben Wadsley, another genetic

	GENE 1	GENE 2	GENE 3		GENE 2,500	GENE 2,501	
SET 2	1	1	0		0	1	
SET 5	0	1	1		1	0	
CHILD	1	1	1		0	1	
SOURCE	SET 2	SET 5	SET 5		SET 2	SET 2	

Figure 4. Heredity in Action—Two Parents (Set 2 and Set 5)

algorithm cohort of mine, wrote an asset-liability management genetic algorithm he got faster results by drawing from five parents rather than two. As I thought more about this, I remembered that when we lived in Northern California, my older daughter once brought home her date, and he was surprised to discover that she had only two parents … and they were still married … and to each other! Clearly, I was naïve in assuming that we had to limit our genetic algorithms in this manner. In this example, we'll draw from our top 20 percent and let any one of the 20 of them be the dominant parent (gene contributor) for any gene in the child's gene string. This will provide a far better level of diversity, and our generations will continue to improve for a far longer time. Once I switched from two potential parents per child to 20 or more, I got a lot more diversity much sooner. Perhaps it does take a village.

Remember, inbreeding is bad among humans; and it is also bad in genetic algorithms. We want to keep the gene pool as diverse as we reasonably can in order to avoid marrying siblings or first cousins. Once again, I go back to genetics and see that a built-in mechanism exists to adapt to changing circumstances and add diversity. It's called mutation. Sometimes (perhaps most of the time) mutations result in a weaker cell; but sometimes it is an improvement. Some bacteria have developed the ability to mutate rapidly and thereby build immunity to antibiotics. We will generate our children as usual, and then randomly mutate some genes in the string. Assume we set our mutation rate to alter 30 genes of the 3,000. Again, play around with these parameters. You can learn (as do your genetic algorithms) through experimentation.

Figure 5 below is a revised picture of how the genes might be populated from five parents.

After we build all the sets for generation 2 (total = elites + children: 100 = 10 + 90), we repeat our test runs with this new generation and sort the scores again. Then, we repeat the process for many generations and watch as our "best set" results get better (lower) numbers. In the sample Excel workbook, all of the scores are normalized such that a solution set containing every provider group would have a score of 1.000. Initially, some solution sets will have scores larger than 1.000 since a random selection will likely include some sets drawn from the more expensive providers. Within just a few generations, though,

	GENE 1	GENE 2	GENE 3		GENE 2,500	GENE 2,501	
SET 1	0	1	0		1	0	
SET 2	1	1	0		0	1	
SET 3	1	0	0		1	1	
SET 4	1	0	0		0	1	
SET 5	0	1	1		1	0	
CHILD	0	0	0		1	1	
SOURCE	SET 1	SET 3	SET 4		SET 1	SET 2	

Figure 5. Five Parents (Our runs actually used more parents.)

we are seeing what looks like intelligence emerging. By generation 10, the score was below 0.85.[10] A few actuaries familiar with the problem were able to use their standard minimization techniques to beat that with scores around 0.78 and they projected that an optimal solution would score around 0.75. However, the genetic algorithm easily beat that within an hour and went on to reach results as low as 0.70. For a different set of adequacy conditions, the actuaries still thought the likely best case configuration was around 0.75, but the genetic algorithm got down to 0.55. Considering the amount of claim payments involved in a large health care network, this could result in a significant savings.

We instructed the program to repeat this process until we stopped getting any improvements (e.g., when the best score stayed the same for 25 generations) and that happened in a couple of days on a relatively inexpensive PC that did not have to be paid overtime for working through the night. The result was a dramatic improvement over the best analytical solutions we were able to achieve by classical actuarial means. The simple emulation of basic evolution got better results without any knowledge of multiple decrement contingencies, or advanced statistics, or differential equations … and yes, I think a student could be taught to do this without any knowledge of algebra!

Let's summarize what we did:

1. We chose a solution set (aka gene string) length of 3,000 where each respective provider group (aka gene) had to be 0 (not included) or 1 (included in our health care network).
2. We formed generation 1 by randomly assigning zeros and ones throughout each set; and we decided to have 100 sets per generation.
3. We tested each set of the generation and saved its score (penalizing, but not eliminating, any set that did not meet our coverage adequacy requirement).
4. We ranked the scores in order from best to worst.
5. We chose the top 10 sets and designated them as elites. Elites get to advance to the next generation intact.
6. We chose to have 20 parents per child, and we built the 90 children needed (to fill out the next generation) drawing from portions of the top-scoring 20 sets.
7. Each gene was chosen from some corresponding gene of one of the 20 parents (randomly choosing the dominant parent for that gene).
8. We went back through the children and randomly mutated 30 of the 3,000 genes (but we did not mutate genes of the elites).

9. We repeated steps 3 through 8 until the scores stopped improving.
10. We went out and partied while the genetic algorithm did all the grunt work for us; meanwhile the theoretical purist actuaries worked through the night trying to come up with a deterministic solution at our top competitor; and thousands of chimps at typewriters tried to pound out the exhaustive best solutions at our not-quite-top competitor.

Lessons learned: Building a genetic algorithm solution is pretty easy; but it is not a cookbook recipe process. The speed and the final solutions are influenced heavily by the starting assumptions. My first algorithms were plagued by inbreeding. Initially, I was so focused on early improvements that I placed too much weight on the scores when I assigned mating rights to the winners of a generation. I was assigning higher parenting probabilities in a direct proportion to better scores. Gradually, I backed off from that approach and found that by increasing the number of potential parents, and decreasing the relative probability of parent A contributing a gene rather than parent B contributing that gene, I avoided inbreeding for a longer time and ended up with better results. I recommend to the reader that you experiment with larger generations (i.e., much more than 100 sets per generation—I normally use at least 1,000), but still keep the number small enough to fit all of a generation into available program memory. I also found it useful to increase the mutation rate as the incremental improvement between generations starts to decrease. Another lesson I learned quickly was that it was important to be able to start from a previous generation of solution sets. That way I could experiment with my assumption parameters (parents,

elites, mutations, etc.) and avoid having to wait for hours to see the impact of those changes. There are many other tips I learned (and many more I am still learning) through these exercises. Whenever I hit a wall though, it was very handy to question my own initial assumptions; and to read more about genetics and then get another inspiration from how evolution accomplishes continual improvements.

Deterministic chaos and genetic algorithms sound like really complex topics. I believe they are not as complex to understand as many of the tools you are currently using. Let's think of them as potential topics in the simplicity sciences and embrace some of these handy tools. A

Dave Snell is technology evangelist with RGA Reinsurance Company in Chesterfield, Mo. He can be reached at *dsnell@rgare.com*.

ENDNOTES

1 *Chaos: Making a New Science*, by James Gleick, 1987, Viking Penguin Inc., New York, New York. Gleick describes the history of Chaos Theory; and how Feigenbaum's constant (4.6692016) became so pervasive throughout seemingly independent science applications. My quote about chaos in non-periodic systems is from p. 18 of this book. You can also read more about this example, and some related ones, in the January, 2012 issue of the *Forecasting & Futurism* newsletter in my article "When Algebra Gets Chaotic."

2 If you would like a more detailed, step-by-step description of how to design a genetic algorithm, see my article "Genetic Algorithms—Useful, Fun and Easy!" In the December 2012 issue of the *Forecasting & Futurism* newsletter. It also gives you a reference to a free workbook for the health provider network problem I discuss briefly in this article. Another good actuarial application of genetic algorithms, for asset and liability management, is in Ben Wadsley's article "Are Genetic Algorithms Even Applicable to Actuaries?" in the July 2011 issue of the *Forecasting & Futurism* newsletter.

3 *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence (Complex Adaptive Systems)*, by John Holland, 1992, The MIT Press, Cambridge, Mass. This is the seminal work that started the genetic algorithm movement.

4 In human genetics, a DNA strand has a limited number of choices at each position (A-T, T-A, C-G or G-C); but the 3.2 billion positions result in a lot of potential variety.

5 See note 2 for a reference to the details for this example.

6 Current thought is that the Big Bang occurred around 14 billion years ago, which is a little over 10^{17} seconds. An interesting thread can be found at *http://answers.yahoo.com/question/index?qid=20080525070816AAaZAOU*. Likewise, the number of atoms in the observable universe is obviously not known precisely; but it is generally thought to be in the range of 10^{78} to 10^{82} (*http://www.universetoday.com/36302/atoms-in-the-universe/*). If we say $2^{3000}=10x$ then $x=3000*\log(2)/\log(10)=903.09$. Taking the outside estimate, 10^{82} times ^{1017} = 10^{99} which is a tiny fraction of 10^{903}.

7 In a typical genetic algorithm application, you may decide to have thousands of sets per generation. I am choosing just 100 here to keep the example simple. The advantage of more sets per generation is a greater diversity and higher probability the smarter sets will be a lot smarter. The disadvantage is that your algorithm will run slower as you have to test every set in the generation before you see your comparative results. It also may be more difficult to hold this information in memory, which can result in a lot of slower disk drive interaction.

8 An excellent book about this concept is *The Red Queen: Sex and the Evolution of Human Nature*, by Matt Ridley (April 29, 2003), Penguin Books, Ltd.

9 My earliest algorithms for mating rights would base the probability of being chosen as a parent on the absolute score the robot (or set) obtained. Thus, a robot getting a score twice as good as the next robot would have twice the chance of mating. This approach works well in early generations; but gradually leads to inbreeding. A better approach was to base mating probabilities on the relative score. In this case, the top scoring robot of 100 would have 100/99 times the probability of mating versus the second place robot, and 100/90 times the probability of the 10th place robot. Try different reproduction schemes to see what fits your particular applications.

10 These numbers are illustrative but your results will vary from them based upon the randomization of initial sets, the PC computing power and other factors. The trend though will be rapid improvement in the first several generations and decreasing improvements as the score becomes closer to an optimal mix of provider groups.