Teaching Mathematical Modelling in a Technology-Enabled Environment

Keng-Cheng Ang
kengcheng.ang@nie.edu.sg

National Institute of Education
Nanyang Technological University
1, Nanyang Walk, Singapore 637616

Abstract:

This paper examines the use of computer simulation as a modelling approach in the teaching of mathematical modelling. In a technology-enabled environment, teachers can design modelling activities based on simulation models that are both suitable and accessible to their learners. We describe how an electronic spreadsheet equipped with coding and programming capabilities is used to investigate problems through suitable simulation models. Three examples of modelling activities will be discussed. Details of the problem situation, problem solution and construction of simulation models will be described in each case. Through these examples of tried and tested modelling activities, we demonstrate the underlying principles in designing simulation models appropriate for use in teaching mathematical modelling. In concluding the paper, we will briefly discuss the support that teachers need in designing, developing and delivering mathematical modelling lessons involving computer simulations.

Introduction

Mathematical modelling begins with a problem in real life and is, by nature, often an inter- and cross-disciplinary activity. As a classroom activity, mathematical modelling often requires both the learner and the teacher to draw on their knowledge outside of mathematics, and as such, it can be challenging, especially when the context is not a familiar one. It is therefore understandable that some learners and teachers find mathematical modelling demanding [9]. In addition, the rather unpredictable and sometimes unexpected outcome of mathematical modelling tasks in the classroom can be daunting and may discourage teachers from carrying out modelling activities [3]. It is no wonder that despite recognising the importance and relevance of mathematical modelling, some teachers remain uncertain and unconvinced about teaching it.

Nonetheless, in a technology-enabled environment, it is possible to find ways to bridge these cognitive gaps experienced by both learners and teachers. Some of the ways to address cognitive challenges using technology while performing mathematical modelling tasks have been examined by various researchers (for example, see [1], [2] and [6]). The proper, timely and appropriate use of a suitable technological tool can often alleviate some of the issues related to cognitive demands. However, a requisite before doing so would necessarily be the acquisition, and perhaps mastery, of the related technological skills and competencies.

In this paper, we describe how an electronic spreadsheet, equipped with built-in coding or programming capabilities may be used to construct mathematical models using the approach of simulation modelling.

Simulation Modelling

While there are several approaches to mathematical modelling, such as deterministic approach, empirical modelling and so on, modelling using simulation often requires the need for technology. In fact, computer simulation has been used as a form of mathematical model in many practical applications and as a teaching tool in a mathematics classroom. In many real world situations, the problem that one wishes to investigate cannot be analysed easily for a deterministic modelling approach to be used. At times, relevant data are not readily available or cannot be obtained easily. In such situations, simulation models can provide a way to study or investigate these problems.

Typically, one would use computer simulations to model a phenomenon or situation when it is either impossible or impractical to construct real physical experiments to model or study it. In such situations, a simulation model would be a useful and inexpensive means of studying the problem.

In simulation modelling, a computer program or some technological tool to generate a scenario based on a set of rules is often used. The rules that control the model or dictate the scenario usually arise from an interpretation of how a certain process or a phenomenon is supposed to evolve or progress. Based on these rules, a computer program can then be written, and executed, and the outcomes of the interactions of the variables or components in the model are generated.

It is quite clear that in order to perform a simulation, we need a technological or computing tool because calculations of values of variables in the model are performed iteratively, rapidly and even continuously throughout the simulation process. While there are many possible tools and computing languages or platforms available, in terms of teaching in the school, one has to consider how accessible these tools are to the learner or even the teacher. For this reason, in the current discussion, the tool that is used is *Microsoft Excel*. Though some may argue that it may not be the best tool, *Excel* does provide the convenience of a spreadsheet and graphing features, and programming capabilities through its Visual Basic Applications (VBA).

Random Numbers

The construction of a simulation model often requires the need to make decisions based on certain outcomes. These outcomes can sometimes depend on variables of the model taking certain specific values. As a simple example, we could simulate a person walking along a straight line with the rule that he takes a step forward if he tosses a coin and gets a "Head", but a step backward otherwise. In order to simulate this, we therefore need to be able to simulate the tossing of a (fair) coin. This can be done on the computer by using *random numbers*.

For the purpose of this paper, we define a random number as one selected purely by chance from some distribution. For instance, suppose the integers in the interval $[1, 10]$ are uniformly distributed. Then, if we are to *randomly* pick an integer from this interval, then every one of the ten integers would have an equal chance of being picked. The concept of randomness lies in the fact that we cannot quite predict which one of the integer will be picked, and thus refer to it as a *random*(ly chosen) *number*.

Many computers and computing tools are equipped with a "random number generator". A random number generator runs a procedure that gives a user a "random number". *Excel* has a built-in random number function, **`rand()`**, which returns a random number between 0 and 1. To use it, one simply enters the function as a formula in a cell (for example, in Cell A1) as shown in the screenshot in Figure 1 below.

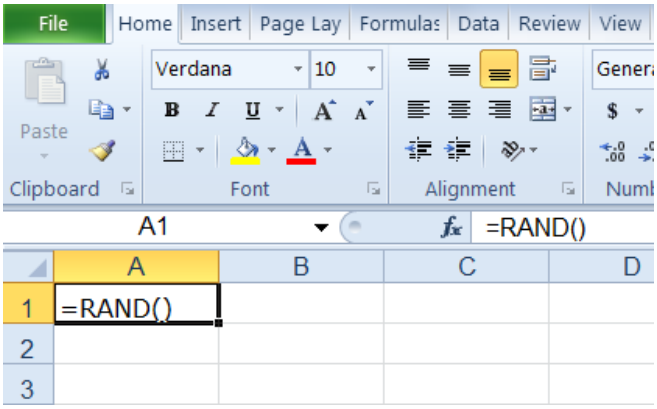

Figure 1: Formula for generating a random number in *Excel*

Once the formula is entered and the RETURN key is hit, a number between 0 and 1 is randomly chosen and placed in Cell A1. Pressing the function key F9 produces a new random number. One may assume that every number in the interval $(0, 1)$ that is *representable* by the computer has an equal chance of being selected. Another useful *Excel* function is the **`randbetween(m,n)`** function, which returns a random *integer* in the interval $[m, n]$. Here, the arguments m and n need not be integers; however, the condition $m \leq n$ must hold for the function to work.

Similarly, one may also generate random numbers using the *Excel*'s *VBA* function, **Rnd**. Like **`rand()`**, the VBA function **Rnd** returns a number randomly chosen from the interval $(0, 1)$. However, unlike **`rand()`**, the function **Rnd** in VBA can have either no argument, or one argument. Moreover, in *Excel* VBA, there is a related function, **Randomize**, which serves to initialize the random-number generator.

The two sets of VBA code shown below are linked to "button" controls in an *Excel* worksheet. Both generate a sequence of 10 numbers ranging from 1 to 6, used to simulate the roll of a fair dice. The use of the **Randomize** function with an argument creates a fixed sequence. The rest of the code is self-explanatory.

```
Sub Button1_Click()
Dim i As Integer
    Randomize (1)
    For i = 1 To 10
       Cells(1,i)=Int((6*Rnd)+1)
    Next i
End Sub
```

Using this button, each time the file is opened, and the button clicked, the 10 dice rolls will produce the sequence,

5, 1, 4, 6, 1, 1, 5, 2, 1 and 5

```
Sub Button2_Click()
Dim i As Integer
    Randomize
    For i = 1 To 10
       Cells(2,i) = Int((6*Rnd)+1)
    Next i
End Sub
```

Using this button, a sequence of 10 dice rolls are obtained but there is no (obvious) pattern in the sequence in each run.

Examples

In this section, we illustrate the use of simulation models in investigating certain problems. Three examples will be discussed, and all simulations are implemented on the *Excel* spreadsheet.

Example 1: Chaos Game

Consider a game called the "chaos game". In this "game", a triangle is first drawn and the vertices are labelled A, B and C. A starting point P_0 is drawn somewhere inside this triangle. Imagine that we now have a "three-sided dice"[1] which, when rolled, will give only three possible outcomes ("1", "2" or "3") with equal probability. The rules for the next course of action, based on each of the possible outcomes, are as follows.

- If the dice shows a "1": find the midpoint between P_0 and vertex A;
- If the dice shows a "2": find the midpoint between P_0 and vertex B;
- If the dice shows a "3": find the midpoint between P_0 and vertex C;

Let this midpoint be P_1.

The process is repeated for P_1 to get the next point, P_2, and so on. The figure shows the results of a simulation of the chaos game set up using a spreadsheet after 8 rolls of the dice. After a large number of repetitions of the same process, what happens?

The following results are obtained after about 8 simulated tosses of the dice

x	y	dice
1.00000	1.00000	2
1.50000	0.50000	2
1.75000	0.25000	1
0.87500	0.12500	3
0.93750	1.06250	3
0.96875	1.53125	3
0.98438	1.76563	1
0.49219	0.88281	2
1.24609	0.44141	2

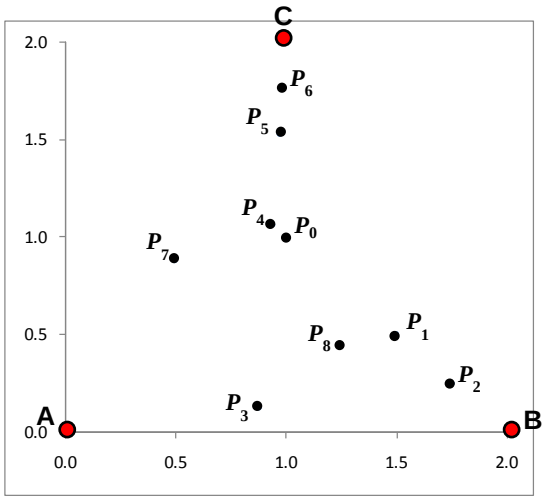

Figure 2: Table and graph of points generated after 8 rolls of the dice

To set up the simulation on *Excel*, we use Columns A and B on the worksheet to store the values of the coordinates (x, y) of points P_0 , P_1 , P_2 , …, and Column C to store the outcomes of the simulated rolls of a "three-sided dice". The **`randbetween(1,3)`** function is used to simulate the rolls of this dice. Subsequent values of the coordinates of the points generated are calculated using *Excel*'s **`IF`** statement or formula.

[1] This could also be a spinner with three equally likely outcome.

For a case where the vertices of the triangle are (0, 0), (0, 2) and (1, 2), the formulae to be entered in the respective cells are shown in Figure 3 below.

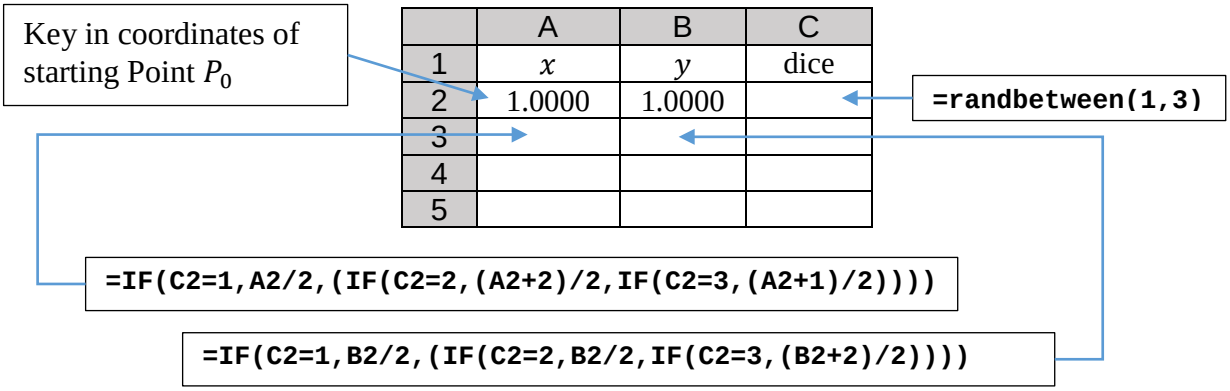

Figure 3: Setting up the *Excel* worksheet to simulate the Chaos Game

Once the formulae are all entered into the cells (that is, Cells A3, B3 and C2) as shown above, they may be copied down their respective columns (for instance, copy Cell A3 down to Cells A4, A5, …) till the desired number of points to be generated is reached. Pressing function key F9 will generate a new set of points. With a large number of points, a pattern will emerge from this seemingly chaotic and unpredictable process (for details, see [4]).

Although this example does not involve any real life problem, it serves to illustrate the use of simulation as a means of problem solving. In this case, without a simulation, it would have been impossible to predict the outcome of the process this "chaos game", or even imagine that a pattern can arise from such a process.

Example 2: Secretary Problem

Suppose a company would like to hire a secretary, with the following conditions and assumptions.

- There are ***n* candidates**, and they are **ranked**, from best to worst without ties.
- The candidates are interviewed sequentially in a **random order**, and each order is equally likely.
- Immediately after an interview, the interviewed candidate is **either accepted or rejected**.
- The decision to accept or reject a candidate can be based only on the **relative ranks** of candidates interviewed so far.
- Rejected candidates **cannot be recalled**.

The task here is to propose a strategy which will maximise the probability of choosing the best secretary from the set of n candidates. Clearly, it would not be wise to simply offer the job to the first candidate we interview. On the other hand, if we wait until the last candidate is interviewed, we may miss out on earlier better candidates, and be forced to accept this last candidate.

A possibly more reasonable strategy is to reject a certain number, for example $k-1$ of the candidates (where $1 \le k \le n$), and then choose the first candidate that is better than all the previous candidates. If no such candidate exists, then we accept the last candidate (implying that we have failed). This is illustrated in Figure 4 below.

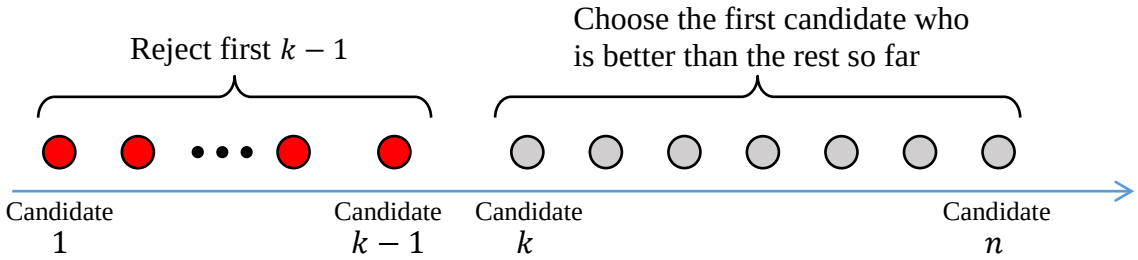

Figure 4: An illustration of a strategy to choose the best secretary

If $k = 1$, we choose the first candidate; if $k = n$, we select the last candidate. Otherwise, the selected candidate is randomly distributed on $\{k, k+1, \ldots, n\}$. We will refer to this strategy as *strategy* k, and the probability of success $p_n(k)$ of using strategy k with n candidates for small n can be computed.

It can be shown that it is optimal to wait until $\frac{1}{e}$ (about 37%) of the applicants have been interviewed (for large n) and then select the first candidate that is better than all of the previous candidates (for details, see [5]). If no such candidate exists, then the last candidate is chosen. The probability of finding the best secretary is also about 0.37.

An *Excel* worksheet can be set up to simulate the process of employing this strategy and to work out the experimental probability of success. As an example, let $n = 10$ and assume we reject the first, say, three candidates. In the first part of the simulation, we set up the candidates by randomly assigning values to each of the 10 candidates. In order to reduce the chance of having ties, the values assigned are randomly chosen integers from the interval $[1, 10000]$. Figure 5 below shows how this part can be set up.

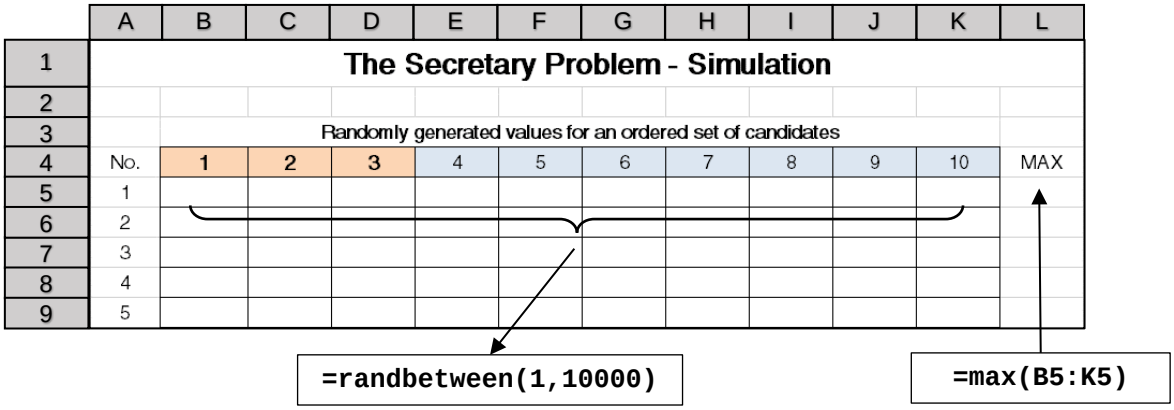

Figure 5: Setting up the Secretary Problem simulation

In the example shown, in Row 5, Cells B5 to K5 store the values of the candidates obtained randomly through the use of the formula **`randbetween(1,10000)`** in the first trial of our simulation. The maximum value of these values is found using the formula **`max(B5:K5)`**, and stored in Cell L5. This represents the value of the best candidate in this instance. These formulae can be copied down the respective columns up to the number of simulation trials desired.

Next, we set up a corresponding table to find the best candidate after rejecting the first, say, three candidates. This is illustrated in Figure 6 below.

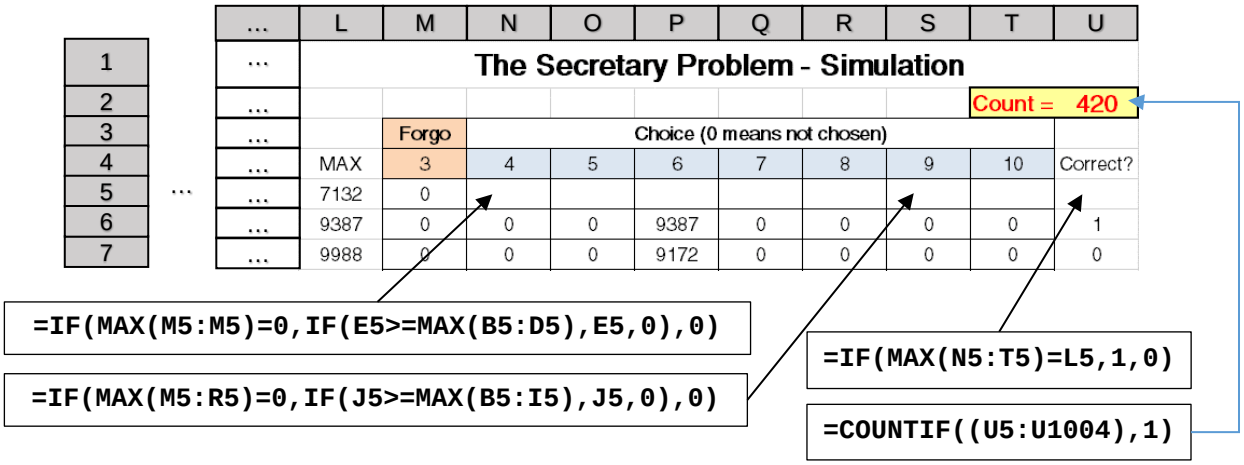

Figure 6: Implementing simulated runs of the strategy for solving the Secretary Problem

In the worksheet shown in Figure 6, the entire Column M indicates that Candidate 3 and those before are rejected. The rows are assigned formulae in the following manner.

Cell N5 contains the formula, "**=IF(MAX(M5:M5)=0,IF(E5>=MAX(B5:D5),E5,0),0)**", which means that if the value of Candidate 4 (in Cell E5) is higher than the maximum value of the previous 3 candidates, then the value is placed in Cell E5; otherwise, the value 0 is entered. This would mean that Candidate 4 is chosen. As another example, for Candidate 9 (whose value is stored in Cell J5), we enter formula "**=IF(MAX(M5:R5)=0,IF(J5>=MAX(B5:I5),J5,0),0)**". This means that all the candidates before Candidate 9 are not chosen, and if the value of Candidate 9 is larger than the highest value of the previous 8 candidates, then the value is placed in Cell S5; otherwise enter 0.

In Column U, from Row 5 onwards, for each row, we check if the maximum of the values stored in Columns N to T is the same as the maximum for that simulated run. If so, a value "1" is entered and if not, "0" is entered. We can then count the number of "1", which will be the number of times the best candidate was successfully picked. We store this value in Cell U2, as shown.

Figure 7 depicts part of a worksheet with 1000 simulated runs.

The Secretary Problem - Simulation

Count = 377

		Randomly generated values for an ordered set of candidates											Forgo	Choice (0 means not chosen)							
		1	2	3	4	5	6	7	8	9	10	MAX	3	4	5	6	7	8	9	10	Correct?
5	1	3412	200	5459	3183	1277	8054	9359	4531	8494	3049	9359	0	0	0	8054	0	0	0	0	0
6	2	4124	6123	3998	2760	3624	2528	8022	8226	6552	9730	9730	0	0	0	0	8022	0	0	0	0
7	3	2849	8013	464	6544	9420	8877	323	3739	1672	5042	9420	0	0	9420	0	0	0	0	0	1
8	4	8196	5511	1705	6920	2675	5843	925	7250	8584	2809	8584	0	0	0	0	0	0	8584	0	1
9	5	1915	8771	8516	907	9238	7146	7493	409	2579	3841	9238	0	0	9238	0	0	0	0	0	1
10	6	5119	2325	9735	1797	3219	430	3422	1416	2325	6608	9735	0	0	0	0	0	0	0	6608	0
11	7	6939	3560	9619	1067	7974	3037	486	7692	8951	8377	9619	0	0	0	0	0	0	0	8377	0
12	8	1581	2953	2390	7824	6085	3320	6610	41	7192	7135	7824	0	7824	0	0	0	0	0	0	1
13	9	8646	9235	7264	9565	9110	7375	2431	2715	1913	2879	9565	0	9565	0	0	0	0	0	0	1
14	10	892	8448	4739	6172	645	3063	5973	7689	6819	7073	8448	0	0	0	0	0	0	0	7073	0
15	11	3654	464	5071	7280	7742	7563	8042	3610	8801	6592	8801	0	7280	0	0	0	0	0	0	0
16	12	133	3110	8117	3032	7210	9264	4882	9863	7410	1421	9863	0	0	0	9264	0	0	0	0	0

Figure 7: Sample worksheet with 377 counts of success in 1000 trials

Example 3: Modelling Swarm Behaviour (Particle Swarm Optimisation)

In an attempt to construct a model to simulate the behaviour of bird flocking or fish schooling, James Kennedy and Russell Eberhart developed an optimisation technique called *Particle Swarm Optimisation* or PSO in short [8]. PSO is similar to other computational techniqus such as Genetic Algorithm (GA), and models behaviour of swarms of "particles" flying through the space of potential solutions in search of some optimal solution.

The concept of the swarm behaviour can be explained as follows. We imagine a flock of birds flying over a certain area looking for food. We assume, quite reasonably, that the one closest to the food source, in its excitement, would chirp the loudest and in doing so, would attract the other birds' attention. The other birds would then tend to move towards the direction where they hear the loudest chirp, and would in turn chirp louder as they get nearer (or think they get near) the food source. This pattern of movement iterates and continues until the food source is reached.

To simulate this behaviour, each bird is assumed to be a "particle" with the following attributes.

- a position (in the form of coordinates) in the problem space
- a velocity (as a vector)
- a personal best value, "pbest" (e.g. in terms of distance to the global best value)

In addition, globally, variables are set up to keep track of the following.

- a target value
- global best value, "gbest" (i.e. best values of personal best values of all particles)
- iteration count and condition to halt iterations

Therefore, each particle i has a position at time t denoted by $x_i(t)$, and a velocity denoted by $v_i(t)$. Each particle i would also have its own "personal best value" denoted by $p_i(t)$, and the entire swarm has a global best value denoted by $g(t)$. We note except for t, all the variables are vector quantities. For computational purpose, we shall assume discrete time steps, and therefore, the next position of particle i is denoted by $x_i(t+1)$. In PSO, $x_i(t+1)$ is obtained by taking into account the current velocity of the particle (inertia component), the personal best value (cognitive component) and the global best value (social component). The diagram in Figure 8 illustrates how $x_i(t+1)$ is obtained.

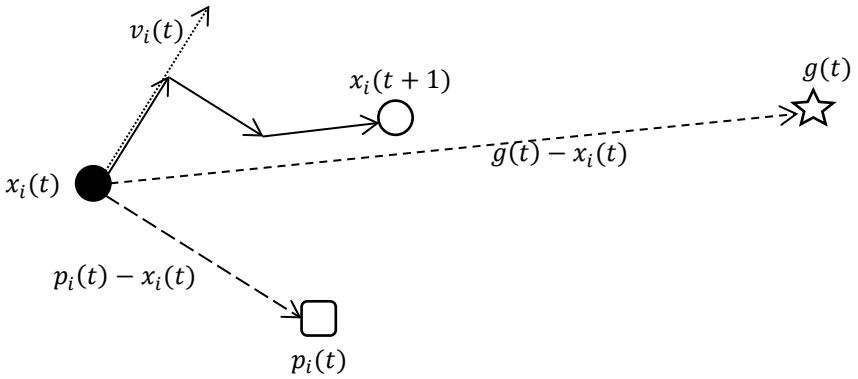

Figure 8: Diagram illustrating how $x_i(t+1)$ is obtained.

In other words, the position of particle i at the $(t+1)$ time step is computed by adding to the position vector $x_i(t)$ a velocity vector $v_i(t+1)$, which in turn is obtained by a weighted sum of

the vectors $v_i(t)$, $p_i(t) - x_i(t)$ and $g(t) - x_i(t)$. For simplicity, as a model for PSO, for each particle i, following equations govern its position over the problem space.

$$v_i(t+1) = av_i(t) + b\big(p_i(t) - x_i(t)\big) + c\big(g(t) - x_i(t)\big) \quad \ldots (1)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad \ldots (2)$$

The PSO algorithm can be summarised as a flowchart shown in Figure 9 below.

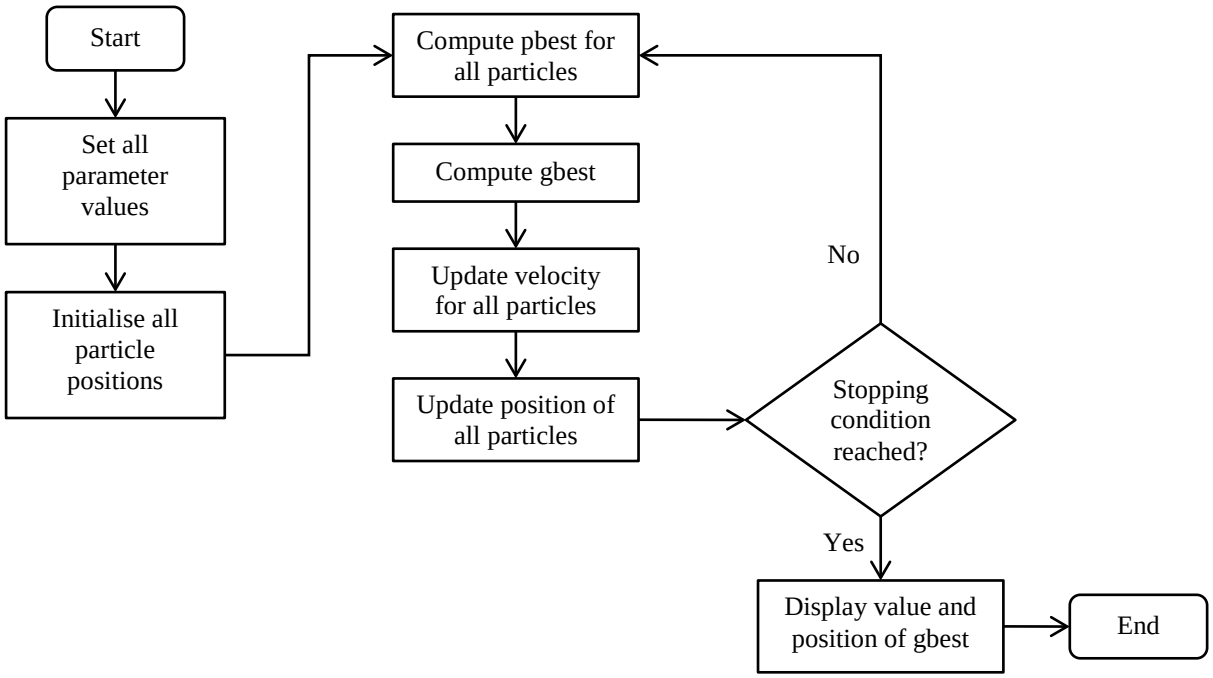

Figure 9: Flowchart showing the PSO algorithm

To illustrate this, consider the problem of locating the minimum (or maximum) of a single-variate function, such as $f(x) = 0.1x^2 - 0.5x + 2$. In this case, the minimum occurs at $x = 2.5$, which is the target position, and the target value is $f(2.5) = 1.375$.

We set up an *Excel* spreadsheet as shown in Figure 10 below.

	A	B	C	D	E	F	G	H	I	J	K
1											
2	n =	10									
3	a =	0.05									
4	b =	0.05									
5	c =	0.05	p0	p1	v	f(p0)	f(p1)	pbest	f(pbest)		
6			4.3225	4.6333	0.3108	1.7072	1.8301	4.3225	1.7072		
7		Start	-3.5953	-3.2462	0.3492	5.0903	4.6768	-3.2462	4.6768		
8			5.5592	5.4415	-0.1176	2.3108	2.2402	5.4415	2.2402		
9		Step	1.6848	1.5556	-0.1291	1.4415	1.4642	1.6848	1.4415		
10			0.4336	0.4082	-0.0253	1.8020	1.8125	0.4336	1.8020		
11			1.7361	2.0106	0.2745	1.4334	1.3989	2.0106	1.3989		
12	gbest	1.3989	-3.4700	-3.7074	-0.2374	4.9391	5.2282	-3.4700	4.9391		
13	pgbest	2.0106	7.3166	7.0756	-0.2410	3.6950	3.4686	7.0756	3.4686		
14			-6.8291	-7.1688	-0.3397	10.0782	10.7236	-6.8291	10.0782		
15			-5.1710	-5.4947	-0.3237	7.2594	7.7665	-5.1710	7.2594		

Figure 10: Worksheet for implementing PSO on *Excel*

This is a simple example where the problem space is the set, $\{x: x \in \mathbb{R}\}$. Geometrically, this means the swarm of particles could search along x-axis for the minimum value of $f(x)$. In this case, we consider a swarm of 10 particles, and the values of the parameters a, b and c are arbitrarily assigned and stored in Cells B3, B4 and B5 as shown. The value of the global best and its position at each time step are stored in Cells B12 and B13 respectively. The current and next positions of each particle are stored in Columns C and D, and their function values in Columns F and G respectively. For each particle, we calculate the velocity and record the value in Column E, and work out its personal best position and value, and store these in Columns H and I, as shown.

In this set-up, we need two buttons, "Start" and "Step" which are controlled by VBA code. Button "Start" initialises the swarm and sets up the worksheet, while "Step" calculates and updates the values one time step at a time. The positions of the particles can be visualised using the usual chart function in *Excel*. Details of the VBA code for the two buttons are found in Appendix A.

Figure 11 shows a sample output of the simulation. As can be seen, the swarm of 10 particles are initially randomly distributed at time step, $t = 0$. At $t = 10, 20, \ldots, 50$, a pattern in the swarm distribution seems to emerge. After about 100 steps, the swarm of 10 particles congregate around the position $x = 2.50$, which is the global best position.

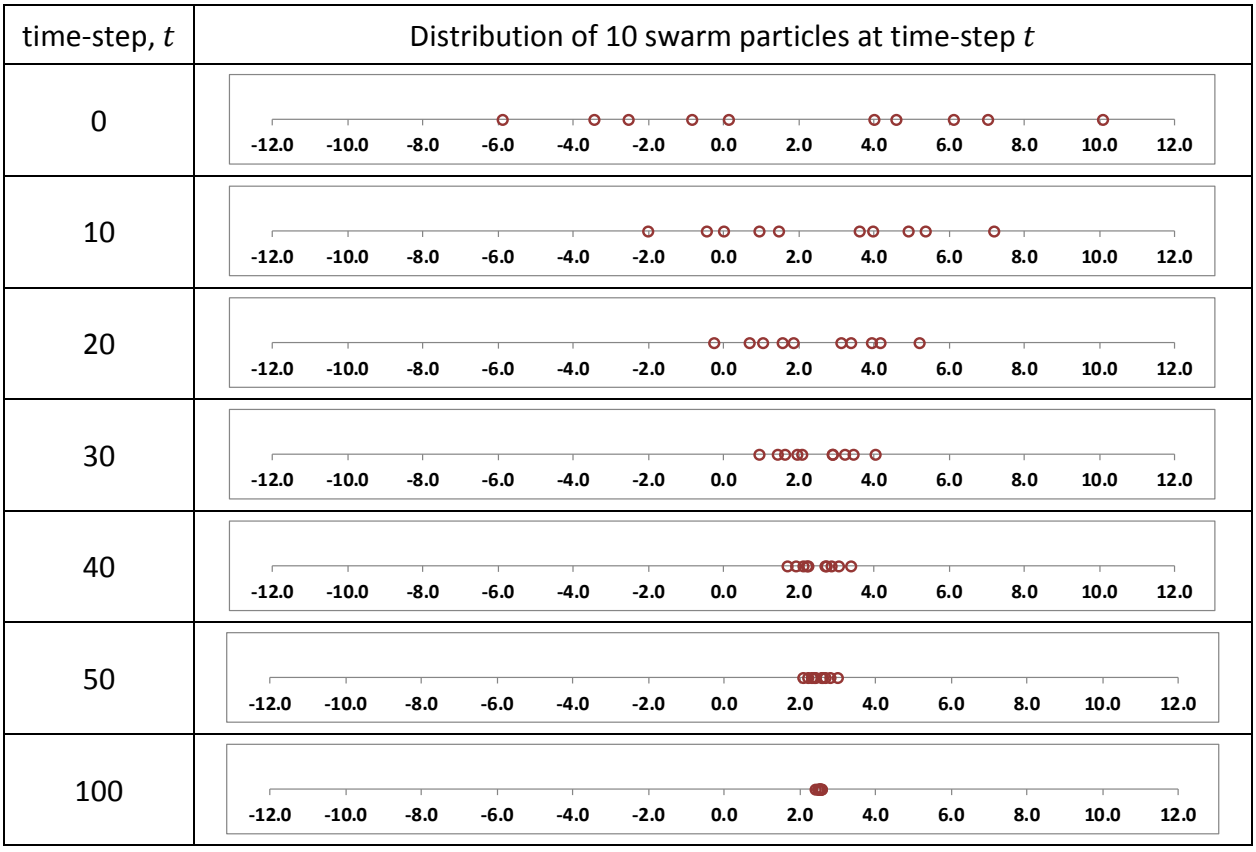

Figure 11: Sample simulation output of PSO technique, showing the swarm particles converging towards the optimal point $x = 2.50$ as the iteration progresses.

Discussion

The examples discussed above demonstrate the usefulness and power of simulation as a way to model or solve problems in mathematics. In addition, these simulations can be conveniently implemented on a commonly used and readily available electronic spreadsheet. While it is possible to code these programs in other computing languages or platforms such as Python or MATLAB, the fact remains many mathematics teachers and learners may not possess these computing skills. On the other hand, for mathematics teachers, an electronic spreadsheet such as *Excel* has become a ubiquitous tool for mathematical investigation in the past few decades, and many teachers and mathematics learners have become very familiar with the use of spreadsheets.

In all the three examples above, further extensions and modifications to automate simulation process can be made through the use of *Excel* controls and VBA code. In the Chaos Game, we could create a button control so that when clicked, it runs a simulation of, say, 500 rolls of the dice. This would eliminate the need to press the function key F9 each time for one roll, and make the simulation more efficient. Similarly, for the Secretary Problem, the entire implementation can be made simpler and more efficient if one uses a control button powered by VBA code to set up the model, and calculate the experimental probability of success based on the strategy discussed. The PSO implementation discussed has already made use of VBA code. However, it can be further modified, for instance, to run 100 steps at a time, or even to run until a certain condition is reached.

Although it appears to make sense to improve efficiency through the use of VBA codes in these examples, doing so may sometimes turn the simulation model into a "blackbox". While this could be what industrial and applied mathematicians would do in practice, it may not be instructive to a mathematics learner. In a mathematics classroom, it is sometimes necessary to break things down into their rudimentary components so that learners of mathematical modelling may be given the opportunity to visualise the entire process of constructing the simulation model.

As can be seen from these examples, in order to construct and implement a simulation model that is suitable for use in the classroom, a teacher will necessarily need to have some basic technological skills. One of the more useful technological skills is the ability to write computer code. The key concern here is not so much about learning a new skill but in developing the computational thinking required in constructing a simulation model. The importance of coding and link between coding and computational thinking has grown in relevance in recent years, and emphasised by various researchers [7]. It is therefore to one's advantage to embrace coding, for both practical and pedagogical reasons, if one wishes to be successful in designing and developing simulation models for the classroom.

Finally, professional development (PD) becomes a critical element of support for teachers who intend to plan and develop such activities in the classroom. It has been shown that a school-based PD for mathematical modelling, with some form of mentoring for novice teachers, generally has a higher chance of success [10], in comparison with the typical one-time workshops or training courses. This is especially so for the case of constructing simulation models, which may require a fair amount of knowledge and skills in coding or programming.

Conclusion

The affordances provided in a technology-enabled environment makes it possible for a classroom teacher to design and plan mathematical modelling lessons using specific tools. In this paper, the simulation approach in mathematical modelling is discussed within the context of teaching

simulation models in the classroom. Three simple simulations of problem situations are used as examples of simulation models, and an electronic spreadsheet, *Excel*, is used as the technological tool to implement the simulations. The examples have demonstrated that the proper and appropriate use of even a simple technological tool can lead to meaningful mathematical modelling activities. Nonetheless, technology cannot do everything; it is important to acknowledge that the teacher needs to be familiar with not just the technology but also the pedagogy involved in capitalising on the technology to be truly successful in developing mathematical modelling activities and lessons.

Acknowledgements

The work reported in this paper was partially supported by a Research Grant (No. *RS 1/16 AKC*) provided to the author by the National Institute of Education, Nanyang Technological University, Singapore.

References

[1] Ang, K.C. (2006). Mathematical Modelling, Technology and H3 Mathematics, *The Mathematics Educator*, 9(2). 33-47.

[2] Ang, K.C. (2010). Teaching and Learning Mathematical Modelling with Technology, in *Proceedings of the 15th Asian Technology Conference in Mathematics*, Kuala Lumpur, Malaysia, pp. 19-29.

[3] Blum, W. and Niss, M. (1991). Applied Mathematical Problem Solving, Modelling, Applications, and links to Other Subjects: State, Trends and Issues in Mathematics Instruction, *Educational Studies in Mathematics*, 22(1), pp. 37-68.

[4] Devaney, R. (2004). Chaos Rules!, *Math Horizon*, November, 11-14.

[5] Ferguson, T. S. (1989). Who Solved the Secretary Problem? *Statistical Science*, 4(3), 282-296.

[6] Ferrucci, B.J. and Carter, J.A. (2003). Technology-active mathematical modelling, *International Journal of Mathematical Education in Science and Technology*, 34(5), 663-670.

[7] Ho, W. K., and Ang, K. C. (2015). Developing Computational Thinking Through Coding, In *Proceedings of the 20th Asian Technology Conference in Mathematics* (pp. 7387). Leshan, China: Mathematics and Technology, LCC.

[8] Kennedy, J. and Eberhart, R. (1995). Particle Swarm Optimization, in *Proceedings of IEEE International Conference on Neural Networks*, IV, 1942–1948.

[9] Niss, M. (1987). Applications and modelling in the mathematics curriculum – state and trends, *International Journal for Mathematics Education in Science and Technology*, 18, pp. 487-505.

[10] Tan, L. S. and Ang, K. C. (2016). A School-Based Professional Development Programme for Teachers of Mathematical Modelling in Singapore, *Journal of Mathematics Teacher Education*, 19, 399-432.

Appendix A

```
Option Explicit
Dim i, n, r  As Integer
Dim a, b, c, gbest, pgbest As Double
```

```
Sub Start()
Dim x As Double
Dim mg As Range

n = Sheet1.Cells(2, 2)
a = Sheet1.Cells(3, 2)
b = Sheet1.Cells(4, 2)
c = Sheet1.Cells(5, 2)

Columns("C").ClearContents
Columns("D").ClearContents
Columns("E").ClearContents
Columns("F").ClearContents
Columns("G").ClearContents
Columns("H").ClearContents
Columns("I").ClearContents

Cells(5, 3) = "p0"
Cells(5, 4) = "p1"
Cells(5, 5) = "v"
Cells(5, 6) = "f(p0)"
Cells(5, 7) = "f(p1)"
Cells(5, 8) = "pbest"
Cells(5, 9) = "f(pbest)"


For i = 1 To n
    r = 5 + i
    Sheet1.Cells(r, 12) = 0
    Sheet1.Cells(r, 3) = Rnd() * 20 - 10
    Sheet1.Cells(r, 5) = Rnd() * 1 - 0.5
    Sheet1.Cells(r, 4) = Sheet1.Cells(r, 3) + Sheet1.Cells(r, 5)
    Sheet1.Cells(r, 6) = f(Cells(r, 3))
    Sheet1.Cells(r, 7) = f(Cells(r, 4))

    Rem find pbest
    If Sheet1.Cells(r, 6) < Sheet1.Cells(r, 7) Then
        Sheet1.Cells(r, 8) = Sheet1.Cells(r, 3)
        Sheet1.Cells(r, 9) = f(Sheet1.Cells(r, 3))
        Else
        Sheet1.Cells(r, 8) = Sheet1.Cells(r, 4)
        Sheet1.Cells(r, 9) = f(Sheet1.Cells(r, 4))
    End If
Next i

Rem initialise
    pgbest = Sheet1.Cells(6, 8)
    gbest = Sheet1.Cells(6, 9)
For i = 1 To n - 1
    r = 5 + i
    If Sheet1.Cells(r + 1, 9) < gbest Then
        gbest = Sheet1.Cells(r + 1, 9)
        pgbest = Sheet1.Cells(r + 1, 8)
    End If
Next i

Sheet1.Cells(12, 2) = gbest
Sheet1.Cells(13, 2) = pgbest
End Sub
```

```
Function f(x As Double) As Double
f = 0.1 * x * x - 0.5 * x + 2
End Function
```

```
Sub Step()
Dim mg As Range
Dim r, i, ibest As Integer
Dim v, dv, dp, dg As Double

For i = 1 To n
    r = 5 + i
    Rem find differences
    dp = Sheet1.Cells(r, 4) - Sheet1.Cells(r, 3)
    dg = pgbest - Sheet1.Cells(r, 3)
     v = Rnd() * 1 - 0.5
    Rem update last position
    Sheet1.Cells(r, 3) = Sheet1.Cells(r, 4)
    Rem generate next velocity component
    dv = a * v + b * dp + c * dg
    Sheet1.Cells(r, 5) = dv
    Sheet1.Cells(r, 4) = Sheet1.Cells(r, 3) + Sheet1.Cells(r, 5)

    Sheet1.Cells(r, 6) = f(Cells(r, 3))
    Sheet1.Cells(r, 7) = f(Cells(r, 4))

    If Sheet1.Cells(r, 6) < Sheet1.Cells(r, 7) Then
        Sheet1.Cells(r, 8) = Sheet1.Cells(r, 3)
        Sheet1.Cells(r, 9) = f(Sheet1.Cells(r, 3))
        Else
        Sheet1.Cells(r, 8) = Sheet1.Cells(r, 4)
        Sheet1.Cells(r, 9) = f(Sheet1.Cells(r, 4))
    End If

Next i

Rem initialise
    pgbest = Sheet1.Cells(6, 8)
    gbest = Sheet1.Cells(6, 9)
For i = 1 To n - 1
    r = 5 + i
    If Sheet1.Cells(r + 1, 9) < gbest Then
        gbest = Sheet1.Cells(r + 1, 9)
        pgbest = Sheet1.Cells(r + 1, 8)
    End If
Next i

Sheet1.Cells(12, 2) = gbest
Sheet1.Cells(13, 2) = pgbest
End Sub
```