International Journal of Science, Technology and Society
2013; 1(1): 9-18
Published online June 10, 2013 (http://www.sciencepublishinggroup.com/j/ijsts)
doi: 10.11648/j.ijsts.20130101.12

Teaching science through computation

Osman Yaşar

State University of New York, The College at Brockport, Brockport, New York, USA

Email address:
oyasar@brockport.edu

To cite this article:
Osman Yaşar. Teaching Science through Computation. *International Journal of Science, Technology and Society.* Vol. 1, No. 1, 2013, pp. 9-18. doi: 10.11648/j.ijsts.20130101.12

Abstract: We present a computational pedagogy approach to teaching an interdisciplinary science course. Modeling and simulation tools allow us to introduce a science topic from a simplistic framework and then move into details after learners gain a level of interest to help them endure the hardships and frustration of deeper learning. More than 90% of students in course surveys state that modeling improved their understanding of science concepts. Students appear to appreciate learning not only the use of simulation tools to design and conduct science experiments, but also basic programming skills to simulate a science experiment using a simple algebraic equation, *new = old + change*. A strong link is established between computational and natural sciences. Students learn in a simplistic framework how laws of nature act as the source of *change*.

Keywords: Computational Modeling, Computer Simulations, Pedagogy

1. Introduction

One of the emerging results from recent reform efforts in computing is the push for *contextualization of computing education* [1-2]. This presents an opportunity to teach computing in the context of natural sciences (or vice versa). Basically, this also points out to bridging the boundaries between computer science and natural sciences, as so suggested by market forces.

The employment data from the National Science Board [3], the Bureau of Labor Statistics [4], and other organizations such as American Physics Institute (AIP) [5] indicate that majority of science and engineering jobs have been increasingly demanding knowledge of computing resources and use of large databases and visualization tools. AIP surveys of physics majors taken in regular intervals after graduation indicate that the most important job skills continue to be scientific problem solving, computer programming, design and development, simulation and modeling, math skills, teamwork, and technical writing.

All in all, we can say that today's STEM jobs require multidisciplinary content in math, computing and sciences. Only a 3rd of undergraduates tend to get a job directly related to their education [3]. As a general conclusion, we can also say that obtaining a broad education continues to be the key to one's employment chances and career advancement in computing. Since only a fraction of physics bachelors in the job sector practice their knowledge of physics, some educators concluded that achieving deep mastery of physics at the undergraduate level might not be as important to students' careers as problem solving and computational skills [6]. Many science departments have, therefore, created new interdisciplinary courses, concentrations, and tracks to prepare their majors for computing jobs [7].

While the demand for computationally competent STEM workers is an unprecedented opportunity, the pipeline between institutions of higher education (IHE) and K-12 seems to be broken in terms of the quality and quantity of students entering college [8-9]. The pipeline problem is so deep that it cannot be fixed easily, at least not without colleges' participation in pre-college preparation [10-11]. The low interest and achievement in STEM have been chronically present for some time. A recent study suggests that student attitudes towards STEM become increasingly negative as a country advances economically, which suggests this phenomenon to be deeply cultural [12]. Learning science is demanding and it requires application, discipline and delayed gratification; values that contemporary culture does not seem to encourage. So, innovative and engaging ways of teaching introductory science and computing are necessary at both IHE and K-12 levels. Here, we describe an interdisciplinary course, namely Introduction to Computational Math, Science and Technology (C-MST), which meets the latest curricular recommendations in both computing, mathematics, and sciences to teach computational thinking skills to new generations [13-15].

2. Computational Pedagogy

Computational modeling and simulations provide us with a deductive pedagogical approach by enabling us to introduce a topic from a simplistic framework and then move deeper into details after learners gain a level of interest to help them endure the hardships and frustration of deeper learning. Once the learner grasps important facts surrounding the topic, a reverse (inductive) process can be facilitated through hypothetical and investigative simulations that enable discovery and honing of relevant principles and skills. Such a stepwise progression is consistent with the pedagogical framework *Flow* [16] and scaffolding strategy to balance skills with challenges (see Fig. 1). Deductive and inductive learning approaches are structurally illustrated in Fig. 2. Computational pedagogy carries both strategies as part of its nature. It puts the learner at the center of a constructivist experience that utilizes both bottom-up (abstraction) and top-down approaches to teaching.

Figure 1. *Illustration of Optimal Flow in learning [16].*

We know that 'attention to details' is important to master a skill, but we all have a limited memory to store information. The most pervasive strategy to improve memory performance (and information retrieval for problem solving) is organizing disparate pieces of information into meaningful units [17]. *Abstraction* skills can help with that by simplifying, categorizing, and registering key information and knowledge for quicker retrieval and processing. The act of abstraction is an *inductive* process by which we sort out details and connect the dots to arrive at more general patterns and conclusions. Abstraction is essential for cognition – in learning new concepts, acquiring language, and making inferences from data. We still do not know how humans make strong generalizations and construct powerful abstractions from such a sparse, noisy, and ambiguous data that is in every way far too limited [18], but it is obvious that a survival instinct (or something else encoded into our genes) is propelling us to make practical decisions in the face of uncertainties and distractions. Shakespeare's *Hamlet* is an example of a mindset (character) exhausted into inaction from calculating all scenarios before action.

The practicality and tendency to filter things out is even built into the nature itself; its ambiguous quantum behavior at the microscopic level is translated (abstracted) into concrete outcomes at our macroscopic level. In a way, the nature is helping us realize the benefits of abstraction by hiding the atomic-level motion so we can see the macro picture (forest) rather than being bogged down by the micro picture (trees).

Computational modeling uses abstraction intensively, by its simplification of the reality, to help eliminate unnecessary details and focus on what is being researched or taught. In research, it uses an *inductive* approach that employs abstraction skills for simplifications. In teaching, modeling supports a *deductive* approach that enables the learner to grasp important facts surrounding a topic before being exposed to the underlying details. Simulation adds another level of benefit by providing a dynamic medium for the researcher or the learner to conduct scientific experiments in a friendly, playful, predictive, eventful, and interactive way to test hypothetical scenarios without having to initially know the underlying science concepts.

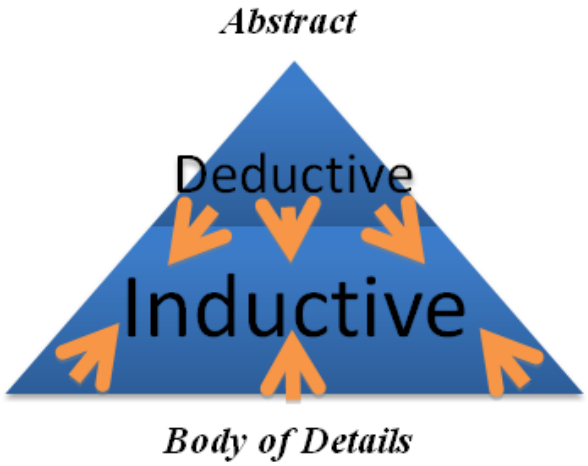

Figure 2. *Structure and directional flow of 'teaching and learning' process during deductive and inductive instructional pedagogies.*

Modeling and simulation meet the specifications of national reports such as [19-20] to teach computational thinking skills. The Task Force in [19] suggests that at early stages computational thinking education should involve *easy experimentation* (learners must be able to quickly set up and run a model using an intuitive user interface, with no knowledge of programming or system commands) and *high interactivity* (models need to evolve quickly and include smooth visualizations for providing interactions and feedback to users).

In our experience, tools such as Interactive Physics (IP) can be used to create many fun things that engage students into computing and sciences. After an initial experimentation with modeling in the context of a science simulation, students can be gradually introduced to the simple principle of mathematical modeling (*new = old + change*) which eventually (and quickly) leads the learner into understanding several aspects of computational

thinking as outlined below. In the process, students get a chance to realize the virtue of decomposing a problem into smaller chunks and to correlate the cost of computation and the choice of tool to the needed accuracy. Students experience firsthand the need for computer programming in order to handle complexity and computational cost of increasing data points as a result of decomposing a problem into much smaller chunks. Linking computing to natural sciences, through the computation of *change*, provides a motivation for science majors to learn programming and for computing majors to learn more about natural sciences.

3. Principle of Computational Modeling

Nature's indirect communication with us seems to be a pattern we see all around us. Rarely revealing a direct relationship such as $y = x^2$, the nature often talks to us in the language of change; that is 'if you change a thing (x) by this much (dx), then another related thing (y) will change by that much (dy).' A simple example of such a relationship is $dy/dx = 2x$; what is often called a derivative, the rate of change, or a differential equation by mathematicians. We consider it as a 'behavioral relationship' because it reveals behavior.

We can often infer a direct relationship from a behavioral relationship. The act of finding a direct relationship, such as $y=f(x)$, from a behavioral relationship, such as $dy= 2x \cdot dx$, is called *integration*. Denoted symbolically by $y = \int 2x \cdot dx$, mathematical integration follows simple rules, but the problem is that finding an analytical answer is not always possible; especially when there are multiple variables and higher-order derivatives describing the behavioral relationship. Examples are many; including those describing climate change, hurricane dynamics, earthquake propagation, population dynamics, ignition in an internal combustion engine, spread of fire, and chemical and nuclear reactions, to name a few. What we do under such conditions, when we cannot infer a direct relationship mathematically, is to employ *numerical integration*. Numerical integration can be illustrated via a simple algebraic equation (*new = old + change*) as described below.

Numerical integration constitutes a major principle of computational modeling and simulations. To numerically solve the above integration problem ($dy= 2x \cdot dx$), let's construct a table of (x, y) data set starting from $(0, 0)$ and using expressions $y_{new} = y_{old} + dy$ and $x_{new} = x_{old} + dx$ where dx is an (independent) increment we get to choose. Table 1 illustrates the steps to construct such a table and Table 2 demonstrates the numerical steps involved for a case in which $dx=1$ and $0 \le x \le 5$. The key here is to first compute the change (dy) using data from the *old* step (x_{n-1}, y_{n-1}) and then move on to establish the *new* data set (x_n, y_n).

Table 1. *A simple algebraic scheme to build an (X, Y) table.*

X	$X_1 = 0$	$X_2 = X_1 + dx$	$X_n = X_{n-1} + dx$
Y	$Y_1 = 0$	$Y_2 = Y_1 + 2 \cdot X_1 \cdot dx$	$Y_n = Y_{n-1} + 2 \cdot X_{n-1} \cdot dx$

Table 2. *Hands-on illustration of the algebraic scheme: 1) compute dY using old data, and 2) get new data: Y=Y+dY and X=X+dX. Last column shows the analytical (exact) solution.*

X + dX => X	dY = 2 · x_{old} · dx	Y + dY => Y	$Y = X^2$
0		0	0
0 + 1 = 1	dY = 2 · 0 · 1 = 0	0 + 0 = 0	1
1 + 1 = 2	dY = 2 · 1 · 1 = 2	0 + 2 = 2	4
2 + 1 = 3	dY = 2 · 2 · 1 = 4	2 + 4 = 6	9
3 + 1 = 4	dY = 2 · 3 · 1 = 6	6 + 6 = 12	16
4 + 1 = 5	dY = 2 · 4 · 1 = 8	12 + 8 = 20	25

As seen in Fig. 3 (and Table 2), the numerical solution (dotted line) results in lower values (undershooting) in comparison to the exact solution (solid line). The behavior of y, in other words its rate of change (slope; as described by $dy/dx = 2x$), depends on x. As we move along the x-axis, this slope increases continuously, but in the numerical solution, we assume it to be rather *constant* throughout each dx interval; based either on the old value of x (slope=$2x_{old}$) or the new value of x (slope=$2x_{new}$). Because of this assumption, we end up inaccurately computing the area under the solid line ($y=x^2$). The brown area (Y-U) underestimates and the 'gray + brown' area (Y-O) overestimates the solution, depending on how we compute the slope. The dashed line (associated with the 'gray + brown' area) in Fig. 3 represents such an overshooting solution as a result of using a higher slope ($2x_{new}$) instead of $2x_{old}$ in the second column of Table 2. Although we instruct students to use data from the *old* step to compute the slope, some students end up using x_{new} and some end up using x_{old} unintentionally, perhaps as a result of how carefully they program it in Excel. The instructor, however, should be ready to explain the difference (undershooting vs. overshooting) in results.

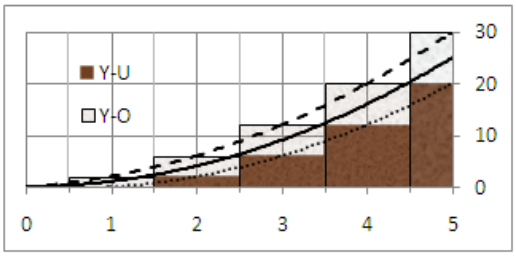

Figure 3. *The numerical solution (dotted line) for dx=1 and the analytical solution (solid line).*

Comparison of numerical results for varying step sizes (dx) could also help students understand the above point. More importantly, it could lead them to discover a correlation between accuracy and the integration size; that is, *the smaller the dx, the more accurate the answer*. This can be illustrated through plots and areas covered under the solid curve ($y=x^2$) in Figure 4. By using a smaller dx, one gets to update the slope more often to more accurately predict the

real solution. While the *dx=1.0* case (with green colored area and associated dashed curve) shows to be a gross approximation of the real solution, one could conclude that the *dx=0.1* case ('red + blue + green' area and dotted curve) is pretty close to the real solution (solid line). If we did not know the real solution, then, of course, we would not have known how close to the real solution we were getting by decreasing *dx* values. Here, testing our numerical method on a known case gave us a chance to calibrate our computational parameters.

We now know that we can use the above computational methodology for problems that cannot be solved analytically. We also know that the accuracy can be improved by employing smaller integration steps; however this comes with a price tag. While a human can calculate a few data points by hand when *dx* is *1*, or *0.5*, the need for automation (and accuracy) becomes obvious for smaller *dx* values such as *0.1* or *0.05*. Excel can be used to automate the calculation and graph the *y=f(x)* curves, but for much smaller step sizes (*dx*), such as *0.001*, *0.0001*, or *0.0000001*, students will discover that Excel cannot help process millions of data points in those computationally intensive cases. The need for finer and faster automation, via computer programming (example shown in later sections), becomes evident as the only way to obtain highly accurate results.

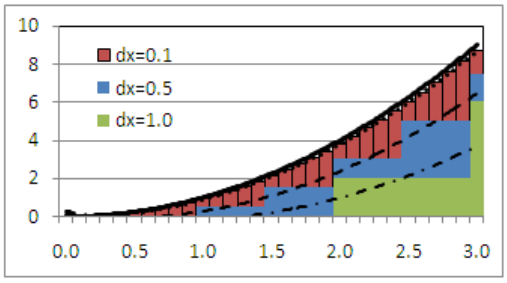

Figure 4. *Numerical results (dashed lines) are compared to the analytic solution (solid line) for several cases of different dx increments (dx= 1.0, 0.5, and 0.1).*

In an after-school project initiated by the author, several 9th graders from Brighton High School (NY) were able to replicate Interactive Physics simulations using the numerical integration method with Excel (and later with Python programming language) to compute the algebraic formulas for the position ($x_{new} = x_{old} + dx$) and velocity ($v_{new} = v_{old} + dv$) as a function of time, where $dx = v \cdot dt$ and $dv = a \cdot dt$, and *a=F/m (acceleration=Force/mass).* All that is needed, then, to conduct these computations is the knowledge (a formula) for the forces that cause change in *x* and *v*. Interactive Physics offer an easy-to-use graphical interface to define forces and their strengths without formulas or other prerequisite knowledge. However, when using Excel or Python, there is a need for a formula that describes the external forces. For example, the force applied by a spring onto an attached object is $F = -k \cdot x$, where *k* is the stiffness coefficient of the spring and *x* is the displacement of the object from the equilibrium position. Another example is the interplanetary gravitational force ($F=G \cdot M \cdot m/x^2$; where *G* is a Universal Constant, *M* and *m* are masses of the objects separated by distance *x*), which governs the orbital motion of a planet around the Sun. These two examples are given in more details in later sections

4. Contextualized Computing Education

The College at Brockport, State University of New York (SUNY Brockport), located in Upstate New York, is a comprehensive liberal arts college with about 7,300 undergraduate students (56% female) and 1,300 graduate students (67% female). In the past decade, the college has supported numerous efforts to increase STEM enrollments, enhance diversity, and promote the culture of research. In the fall of 1998, it started the nation's first undergraduate program in computational science [7, 21-23] and in 2003 it formed an institute with local school districts to promote an integrated (computational) approach to math, science, and technology (C-MST) education as a way to strengthen the IHE-K12 pipeline [24-27]. The computational science (CPS) faculty has developed more than thirty new courses, including three General-Education courses such as CPS 101 Introduction to Computational Math, Science and Technology, CPS 105 Games in Sciences, and CPS 302 Science, Technology, and Society.

CPS 101 teaches principles of computing in the context of modeling and scientific simulations as detailed in this paper. CPS 105 uses agent-based modeling to teach programming concepts in the context of culturally popular and low threshold applications such as games, while CPS 302 discusses scientific and computing concepts in the context of historical evolution and personal lives of their inventors, supported by demonstrations using again modeling tools. While motivational and educational aspects of these courses are important to draw students into computing and sciences, it is also important that students go beyond a sugarcoating of 'fun' to the full engagement these courses offer. An important question to answer would be if learners gain skills that are transferable to other domains or do they merely learn how to use specific computer programs to construct games and fun simulations.

Below, we offer details of one of these courses (CPS 101) that meet most of these desired outcomes, both in a fun and an educational way. Course materials include class notes, online modules posted at *www.brockport.edu/cmst*, and user manuals for tools Interactive Physics (IP) and Python. Its general structure below is followed by the weekly schedule in Table 3.

- Design and modeling tools: Learn tools such as Interactive Physics (IP) to create simulations.
- Principles of computational modeling: Learn numerical integration by hand; use Excel to establish a link between computational accuracy and cost; then discuss use of programming to obtain more accuracy, better control, and higher automation.
- Computer organization: Discuss factors impacting computational efficiency, including processing

speed, storage capacity, communication bandwidth, system software, and the hierarchy (and cost) of information flow between registers, cash, memory, and storage units.

- Programming: Learn programming and algorithmic concepts via a text-based language, such as Python.
- Computational science applications: Conduct science simulations by solving the same problem using multiple tools (IP, Excel, and Python); discuss trade-offs and why certain tools are utilized to solve particular problems.

Table 3. *Weekly schedule for CPS 101*

Week 1: Conduct surveys to examine math & computing skills. Discuss the role of modeling in scientific inquiry and industrial design (watch videos and show examples). HW #1: Search the web and write a short essay on the role of computational modeling.
Week 2: Discuss principles of computational modeling: *new=old + change*. Discuss functional (i.e., $y= x^2$) and behavioral (i.e., $dy= 2x \cdot dx$) relationships in tabular, formulated, and graphical forms.
Week 3: Discuss the rate of change and the difference between average & instantaneous rates of change. Test #1 (functions, rate of change, and forms of representation (tabular, formulated, and graphical)).
Week 4: Conduct numerical integration by hand and Excel. Discuss the role of integration step in reducing error & the role of computing power to afford smaller steps. HW #2: Numerical integration by hand & Excel.
Week 5: Computer Lab: Introduce Interactive Physics (IP) training. HW #3: Design a creative IP project to demonstrate comfort level with it.
Week 6: IP Labs: a) Harmonic motion: generate position and velocity plots of a spring-driven object; b) Falling objects: examine motion under gravity. HW #4: Create a swinging pendulum and report observations.
Week 7: Discuss the role of hardware (storage, processing, and communication) and software (data locality, memory usage, system software, and programming) on performance and accuracy. Introduce programming concepts using Python. Conduct a Midterm Exam.
Week 8: Discuss programming examples in Python. HW#5 and Test #2 on programming and on factors that affect computational performance.
Week 9: Break
Week 10: Lab: Review use of multiple tools (IP, Excel, and Python). Redo harmonic motion using *new = old + change* computations via Excel and Python. Learn about Newton's law of motion ($F=m \cdot dv/dt$) as Force being the cause of *change* in velocity and position. Compare IP, Excel and Python results.
Week 11: Lab: Trajectory of projectile: Use IP and Excel to study 2-D motion. HW #6: Write a Python program that computes the trajectory of a rock thrown upwards at an angle.
Week 12: Lab: Conservation of energy & momentum. Use IP to graph potential and kinetic energies of objects. Examine effects of friction.
Week 13: Lab: Orbital motion: Watch videos on orbital motion and space explorations. Learn about gravitational force $F=G \cdot M \cdot m/r^2$ as a cause of *change* in position. Simulate orbital motion in 2-D using Excel and then Python. HW #7 (Team project: Proof of Kepler's Laws).
Week 14: Lab: Team project: Design a project; discuss issues of computational efficiency and cost using multiple tools (IP, Excel, Python); and explain role and benefits of each tool.
Week 15: Re-visit HW #1 to improve the earlier essay.
Week 16: Conduct a review for Final Exam.

4.1. Programming with Interactive Physics

IP is used to model, simulate, and explore a wide range of physical phenomena, including harmonic motion (springs and pendulums), falling objects, trajectory of projectiles, energy conservation, orbital motion, Kepler's Laws, Newton's second law of motion, and electrostatic oscillator. Through IP, students are able to conduct experiments and investigate events without deeply knowing or memorizing the laws of physics. Users are allowed to set up their own physical world; add, remove, or modify external forces; monitor position, velocity, energy, and elapsed time; and also create control buttons to facilitate a simulation. Visual images and data can be transferred to Excel for analysis or to the Geometer's Sketchpad (GSP) to measure angles, distances, and areas needed for proofs or other calculations.

Screen shots of two separate IP simulations are shown in Fig. 5 and 6. Figure 5 shows a box subject to an external force by an attached spring. Control buttons allow the user to change initial velocity (v_0) and mass (m) of the box as well as the stiffness coefficient (k) of the spring. Figure 6 shows orbital tracks of three planets (Mercury, Venus, and Earth) around the Sun. The planets are represented by small circles, with appropriate masses and orbital velocities. For example, the earth (of mass 5.9×10^{24} *kg* and orbital velocity of 6.65×10^4 *mph*) is placed at 150×10^6 *km* from the Sun (mass of 1.89×10^{30} *kg*). Figure 6 is actually a screen dump from IP into the Geometer's SkecthPad (GSP) that measures distances for the proof of Kepler's laws, which states that for each planet the square of its period (T^2) is proportional to its semi-major R^3; or $(T_1/T_2)^2 = (R_1/R_2)^3$ for any two planets.

4.2. Programming with Excel

While IP is a good tool to expose students to many concepts, computing education needs to move beyond just using tools. Our experience indicates that students need to eventually understand the underlying mechanism of modeling and simulation and to flexibly master and apply acquired knowledge rather than practice rote memorization of scientific laws.

In CPS 101, students are required to model a natural phenomenon by computer simulation using IP, and then solve the same problem via Excel and later by writing a computer code using a language such as Python.

To use Excel for generating position and velocity values of an object subject to an external force, students need to designate several columns in an Excel worksheet to these variables as shown in Table 4 for the experiment in Fig. 5. The direction of computation in Table 4 is from left to right: first compute $dx=v_{old} \cdot dt$ using the velocity from the previous step; update $x=x+dx$; compute $dv=-(k/m) \cdot x_{old} \cdot dt$; and finally update $v=v+dv$. The first row in each column holds variable names and the 2nd holds initial values *(t=0 s, v = 1 meters/s* and *x= 0 meters)*. The 3rd row holds expressions computed in the order indicated, where *t, v,* and *x* are linked to their own values from the previous row. The computed expressions can be copied and pasted to the rest of

the rows below until *t* reaches maximum time *(T)* desired. This is where the limitations of Excel come into play. The visible and scrollable screen might not accommodate the desirable simulation range when the integration time step *(dt)* is very small. If one chooses *dt* to be *0.000001 s*, then one needs *1,000,000* rows to do an Excel computation for *1 s*.

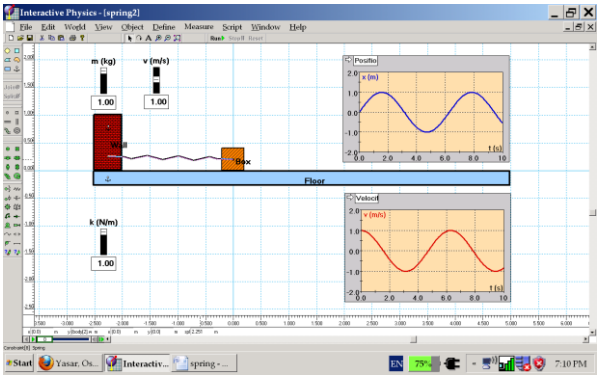

Figure 5. *Simple harmonic motion with Interactive Physics.*

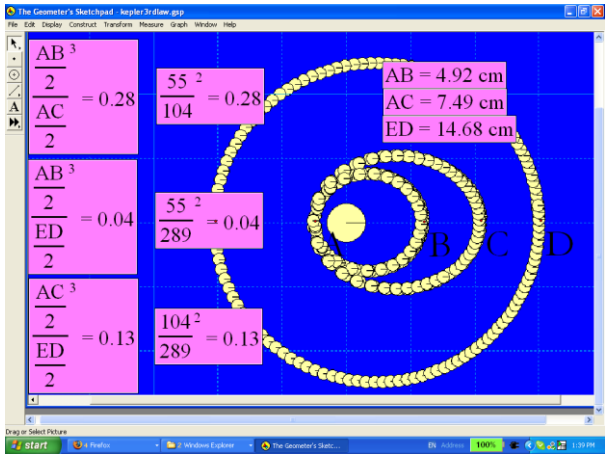

Figure 6. *Orbital tracks of several planets around the Sun. Orbits and periods are shown to prove Kepler's 3rd Law.*

In Fig. 5, we assume that there are no external forces (no friction or air resistance) on the box other than the force applied by a spring, which is characterized as $F = -k \cdot x$, where k is the stiffness coefficient of the spring and x is the displacement of the box from the equilibrium position ($x=0$). In this case, the acceleration becomes $a = -(k/m) \cdot x$. The chosen values for spring stiffness (k) and object's mass (m) are *1.0 Newton/meter* and *1.0 kg*. Figure 7 displays Excel computations for comparison with those seen in the IP window (Fig. 5). While the Excel results qualitatively match the IP results, it is clear that the amplitude of both position (x) and velocity (v) in Fig. 7 is growing at each cycle due to an error resulted from numerical solution.

There are ways to minimize the numerical error. In situations where change is very dramatic, we can either lower the step size *(dt)* to smoothen the transition as we did in the previous example or use newly calculated quantities (x_{new} and v_{new}) as soon as they become available in the computation of change. For example, in the harmonic motion, as soon as the box moves by x amount, the spring applies acceleration, proportional to $-(k/m) \cdot x$, to pull it back. At the first step, this acceleration could be quite large because of a highly stiff spring (large k), or a light box (small m), or a high initial velocity (v) that causes a big jump ($dx = v_{old} \cdot dt$) in position x. Despite such a large acceleration, its influence will not be felt until x is updated in the next time step. Therefore, much information will be missed (lost) if the time step is too large to capture a big change. As a result, not feeling the pulling force from the spring, the box will end up moving further away from it at each cycle. The growing amplitude in Figure 7 shows that. This growth is larger for bigger time steps.

Table 4. *Harmonic motion using Excel (dt= 0.1)*

t+dt (s)	dx= v_{old}·dt	x+dx (m)	dv= a·dt = - (k/m) · x_{old} · dt	v+dv (m/s)
0		0		1
0.1	0.10	0.10	0.00	1.00
0.2	0.10	0.20	-0.01	0.99
0.3	0.10	0.30	-0.02	0.97
0.4	0.10	0.40	-0.03	0.94
0.5	0.09	0.49	-0.04	0.90
0.6	0.09	0.58	-0.05	0.85
0.7	0.09	0.67	-0.06	0.79
0.8	0.08	0.74	-0.07	0.73
0.9	0.07	0.82	-0.07	0.65
1	0.07	0.88	-0.08	0.57

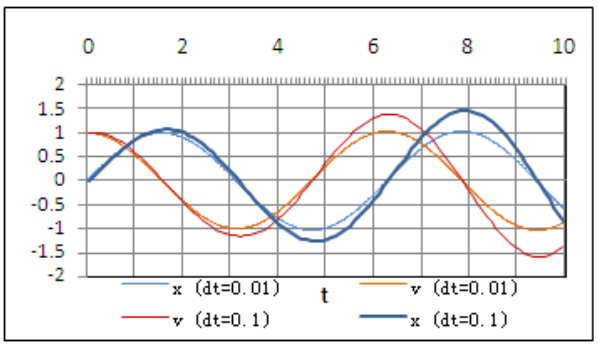

Figure 7. *Plot of position (x) and velocity (v) in Table 4*

Smaller step size leads to more accuracy as seen in Fig. 7 (in comparison to the graphs in Fig. 5). While lowering the step size (*dt*) by an order of magnitude has significantly dampen the growing amplitude, it has not stopped it. As mentioned before, another way of minimizing the numerical error is to put into use newly calculated/updated quantities in the formulas. For example, if we compute the acceleration based on the updated position, such as $a = -(k/m) \cdot x_{new}$, in the fourth column in Table 4, then we will have captured major changes in acceleration even during a step size as big as *0.1*. The orange curve in Fig. 8 illustrates that such a

technique accomplishes as much accuracy as a computation (light green curve) with a step size 10 times smaller (and ten times more costly).

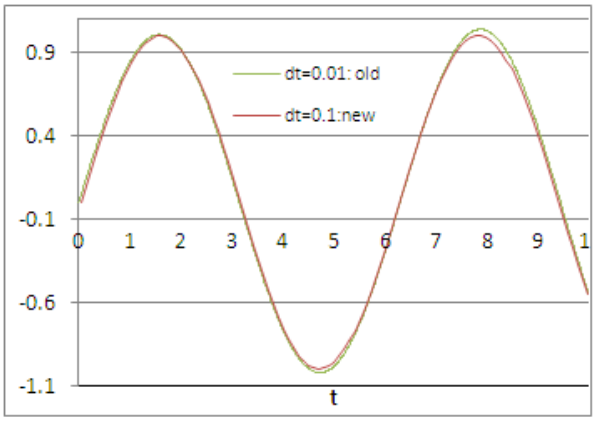

Figure 8. *Comparison of the numerical solution for position (x) using two different step sizes (dt=0.1 and 0.01) and two slightly different ways of computing the change in velocity*

In two-dimension, the above equations can be expanded as:

$$x_{new} = x_{old} + v_x \cdot dt; v_{x,new} = v_{x,old} + a_x \cdot dt;\ a_x = (x/r) \cdot a,$$

$$y_{new} = y_{old} + v_y \cdot dt; v_{y,new} = v_{y,old} + a_y \cdot dt;\ a_y = (y/r) \cdot a,$$

$$r^2 = x^2 + y^2;\ \text{and} v^2 = (v_x)^2 + (v_y)^2.$$

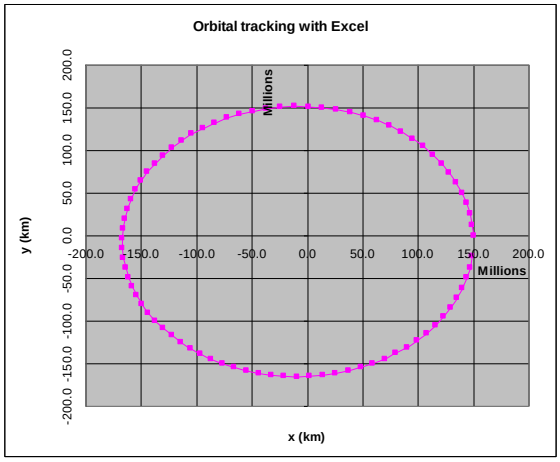

Figure 9. *Orbital tracking of the Earth using Excel.*

Using these equations along with interplanetary acceleration between Earth and the Sun ($a = F/m = G \cdot M/r^2 = 1.26x10^{14}\ N \cdot km^2/kg \cdot 1/r^2$; where G is a Universal Constant and M and m are masses of Sun and Earth), one could get the orbital track for the Earth as seen in Fig. 9. At $t=0$, we assume the Earth's orbital velocity to be $v_x=0$ and $v_y= 29.79\ km/s$ and its position to be $y=0$ and $x= 150x10^6\ km$. What is shown in Fig. 9 may not be the most accurate track, but qualitatively it is representative of a planet's orbit. Some planets have more elliptically looking orbits (see Fig. 6). The Excel calculations in Fig. 9 are given for dt = 5 days, and smaller time steps (*i.e.*, $dt=1$ day) could produce more accurate tracks. Again, that is where the limitations of Excel come into play, just like the million data points mentioned above. With computer programming, these limitations can be overcome. While doing these computations, students realize higher resolution and automation need use of a programming language.

4.3. Programming with Python

In the past we used Fortran and C++in CPS 101, but we have recently switched to Python. The switch to Python was based on three major reasons, including relative easiness and quickness with learning of Python as a computer language, its simple and short constructs, and less error-prone coding. Python is a general-purpose, object-oriented, high-level programming language, which comes with extensive standard libraries and it supports the integration with other languages and tools. It is increasingly used in scientific computing, web development, and database operations. Python can be learned in a couple of weeks for basic operations; it is open-source and platform-independent, and it can be installed on almost any computers free of charge. An introduction to basic syntax, input/output functions, repetition structures (loops), and algorithmic thinking is adequate to carry out programming assignments necessary for computing a mathematical or logical expression repetitively, recursively, and iteratively. Students can write simple loops to compute and generate data points for a number of problems listed in the course syllabus including falling objects, trajectory of projectile, harmonic motion, and orbital motion. A few lines of computer programming, as shown below without the exact syntax, can accomplish the above calculations with less effort and much higher accuracy.

```
while t <=T: # this sign preceeds comments
print x, v, t # position, velocity, time
a = F/m # compute acceleration
v = v + a × dt # update velocity
x= x + v × dt # update position
t = t + dt # update time & end loop if t >T
```

5. Results and Discussion

We used a mix-methods approach [28-29] to examine the student reaction to the interdisciplinary teaching of the CPS 101 course. Table 5 shows some of the findings from student surveys in consecutive years (2011, 2012, and half of 2013). Although it is difficult to infer meaningful results from our sample size (20-40 students per year), we can see some trends as a result of triangulating our survey results with classroom observations.

Launched in 1998, the CPS 101 was taught by the author to a classroom full of students until 2007 when a new faculty was hired to teach it. The new instructor's tendency to teach it merely as a programming course in C++, along with high level mathematics, brought enrollments in this introductory level course down to single digits in 2009. In 2010, through support from an NSF grant, the content was gradually

shifted back from 'differential equations and computer programming' to its original content of basic problem solving as described here. These changes, along with classroom instruction by a former K-12 teacher, brought the enrollments back again to more satisfactory levels, vindicating the computational thinking approach to introductory computing and science education that is being currently advocated at the national level [1, 13, 15, 19, 30].

Table 5. *Survey results from CPS 101*

Survey Questions (Q) Responses are in percentages (%).	**CPS 101**		
	2011	**2012**	**2013**
1. Would you recommend this course?	97	94	100
2. Do you like to take an advanced course?	44	63	92
3. Did modeling improve your learning?	68	82	92
4. Did you like project-based learning?	90	94	100
5. Did you have the necessary background?	60	75	85
6. Were your skills a match for challenges?	78	82	85
7. Did you ever feel frustrated?	32	25	15
8. Will new skills help you in later classes?	96	56	100
9. Did you change your major afterwards?	10	13	15

Currently, CPS 101 has a broad appeal, drawing students from many departments, including non-STEM majors. Majority of surveyed students liked project-based learning, which involved design of a game or a science experiment. A significant portion of students thought that modeling improved their understanding of science concepts. This portion grew every year as the shift from programming to modeling occurred. While the attendance rate in the lecture session was around 70%, it jumped to 90% in the computer lab. Students seemed highly engaged in lab activities and involved in practicing different computational tools. Classroom observations and attendance records indicate significant improvement from 2011 to the current year in student participation in hands-on lab activities.

A sizable number of students (40% in 2011, 25% in 2012, and 15% in 2013) said they did not initially have the necessary background and skills. The level of frustration (32% in 2011, 25% in 2012, and 15% in 2013) reflects this. When asked about how they overcame this deficiency, they cited professor's help, additional practice, project-based learning, and scaffolding. The gradual shift from heavy programming and differential equations in 2010 to the use of design-based tools (such as IP) and more applied mathematics in 2013 also seems to lead to more engagement and confidence. However, this gain comes with a price, at least for now until there is a better vertical integration between similar courses in terms of using the same tools.

When programming and differential equations were main tools of the instruction, despite their difficulties students saw more value in learning them to further their education than the design tools. The feeling that the tools they learned would be utilized in future classes went down from 96% in 2011 to 56% in 2012. Because of our concern about transfer of skills and vertical integration to more advanced courses, in 2013 we reached a more balanced use of design and programming tools. We also better explained the pros and cons of different tools used. Some of the answers to "What did you like (or dislike) most about this class?" included the following: "I most enjoyed the experience of using different modeling and programming software," "Learning and doing examples on a number of different programs," and "I like how you program something, using the code, and you can usually see the outcome." Additional time may need to pass before the same tools are used in other advanced courses to create a feeling of continuity. Also, because CPS 101 teaches the principles employed by computational modeling tools; its emphasis on fundamentals goes way beyond just using a particular tool. Students may realize the benefits of learning such fundamentals much later when they encounter new modeling tools in their courses or work environment. In the latest surveys, all respondents stated that newly learned skills would be beneficial in their education and work.

Another front that we examined our computational approach to STEM education was K-12. This was done in the form of a C-MST professional development (PD) for secondary school math, science, and technology (MST) teachers in two partnering school districts (Rochester City School District and Brighton Central School District). The content of a 3-tier (beginner, intermediate, and expert level) teacher training and its overall impact on teacher retention and student achievement is being published elsewhere in [27], but here we will briefly mention the surveys on student engagement as a result of technology-enhanced instruction in the classroom. Majority of the 200 trained MST teachers surveyed by external evaluators from 2005 to 2010 reported encouraging results on infusing technology into classroom instruction. While it initially involved use of laptops for presentations, graphing calculators for math instruction, and electronic smart boards for interactive lessons, these surveys indicated that it took 3 years (or 200 hours of training) for a teacher to feel comfortable with modeling tools such as IP. This is consistent with findings of a report by the Urban Institute in 2005 that a minimum of 160 PD hours are needed to effect changes in the classroom environment. 60% of the beginner-level teachers in 2003 reported occasional use of modeling in their classrooms. In a 2010 survey of 40 expert-level teachers (who had three years of prior training), 78% of them reported that they regularly used modeling software in their classrooms and 92% of them indicated use of smart boards. 90% of those who used modeling software, graphing calculators, and smart boards in their lessons agreed that use of technology increased student engagement and made math and science concepts more comprehensible. Student reaction to modeling (versus traditional techniques) was found to be quite favorable in both mathematics (97%) and science (77%) classes. While science classes utilized technology less due to limited access and lack of science modeling examples, in instances where it was utilized, it actually led to a deeper understanding of science topics than it did for mathematics topics (83% vs. 76%).

6. Conclusion

We believe that the practice of teaching introductory computing courses in the context of natural sciences (or vice versa) offers benefits to a wide group of students. While science majors get to use simulation tools and computer programming to solve science problems, math and computer science majors get to establish a link to natural laws through computation of change.

By using multiple tools (IP, Excel, and Python) to solve the same problem, students get a chance to weigh advantages of each tool and conclude firsthand that more accurate and faster computation of *new =old + change* for a large number of data points require computer programming. In the context of applications, it is easy for students to understand why they need to learn computer programming.

Steep learning curves, limited technology access, time constraints and rigid curriculum frameworks are hard to overcome, especially at the K-12 level. While integration of computational modeling and inquiry into mathematics and science courses has been slowed down by the above factors, the new K-12 math and science standards and the upcoming AP course to teach computational thinking skills might accelerate such integration in a systematic way.

Acknowledgements

This work was supported by the National Science Foundation (NSF) funds via Grants #0226962, #0942569, and #1136332. We would like to thank to faculty and teachers whose efforts contributed to the development, teaching, and assessment of the reported courses and materials. Special thanks to Pınar Yaşar who introduced the author to the Shakespearean world of human thought.

References

[1] Guzdial, Mark. (2009). Teaching computing to everyone. *Communications of the ACM,*Vol. 52, No. 5, 1-3.

[2] Computing Curricula.(2005). A Cooperative Project of the Association for Computing Machinery, the Association for Information Sciences, and the IEEE Computer Society. http://www.computer.org/portal/web/education/Curricula.

[3] S & E Indicators. National Science Board. 1996 and 2010. http://www.nsf.gov/statistics/.

[4] BLS Report. (2010). The Bureau of Labor Statistics. Occupational Employment Statistics. http://www.bls.gov/oes/.

[5] AIP Survey. (2010). Important Knowledge & Skills Used on the Job. American Institute of Physics. http://www.bls.gov/oes/2010/may/stem.htm.

[6] Landau, R. (2006). Computational Physics: A Better Model for Physics Education? *IEEE Comp. in Sci & Eng.*, 8 (5), 22-30.

[7] Swanson Survey. (2010). A Survey of Computational Science Education. By C. Swanson. The Krell Institute, http://www2.krellinst.org/services/technology/CSE_survey/.

[8] NAP Report. (2007). Rising Above The Gathering Storm. Washington, D.C.: The National Academy Press. http://www.nap.edu/.

[9] NAP Report. (2010). Rising Above The Gathering Storm, Revisited: Washington, D.C.: The National Academy Press. http://www.nap.edu/.

[10] National Science Foundation, Math and Science Partnership (MSP) Program. http://www.nsf.gov.

[11] Cuny, J. (2011). Transforming Computer Science Education in High School. *IEEE Computer*, 44 (6), 107-109.

[12] Sjøberg, S. and Schreiner, C. (2005). How do learners in different cultures relate to science and technology? Results and perspectives from the project ROSE.*Asia Pacific Forum on Science Learning &Teaching*, 6, 1-16.

[13] The College Board. (2011). AP CS Principles Course. http://www.csprinciples.org. Also see June 2012 issue of *ACM Inroads*.

[14] NGSS (Next Generation Science Standards). (2013). http://www.nextgenscience.org/.

[15] Wing, J. M. (2006). Computational Thinking, *Communications of the ACM*, Vol. 49, No. 3, 33-35.

[16] Csikszentmihalyi, M. (1990). *Flow: The Psychology of Optimal Experience*. New York: Harper Collins.

[17] NAP Report. (2000). How People Learn: Brain, Mind, and School. Washington. D.C: The National Academies Press. http://www.nap.edu/.

[18] Tenenbaum, J. B., Kemp, C., Griffiths, T. L. & Goodman, N. D. (2011). How to Grow a Mind: Statistics, Structure, and Abstraction. *Science*, 331, 1279-1285.

[19] NSF Report. (2008). Fostering Learning in the Networked World. National Science Foundation. http://www.nsf.gov/pubs/2008/nsf08204/nsf08204.pdf.

[20] NSTA Report (2008). Technology in the Secondary Science Classroom. National Science Teachers Association. (Eds) Bell, L. R., Gess-Newsome, J., and Luft, J. Washington, DC.

[21] Yaşar, O., Rajasethupathy, K., Tuzun, R., McCoy, A. and Harkin, J. (2000). A New Perspective on Computational Science Education, *IEEE Comp. in Sci & Eng*, **5** (2), 74-79.

[22] Yaşar, O. (2001). Computational Science Education: Standards, Learning Outcomes and Assessment. *Lecture Notes in Computer Science*, 2073, 1159-1169.

[23] Yaşar, O. and Landau, R. (2003). Elements of Computational Science & Eng. Education, *SIAM Review*, 45, 787-805.

[24] Yaşar, O. (2004). C-MST Pedagogical Approach to Math and Science Education. *Lecture Notes in Comp Sci*, 3045, 807-816.

[25] Yaşar, O., Little, L., Tuzun, R. Rajasethupathy, K., Maliekal, J. and Tahar, M. (2006). Computational Math, Science, and Technology, *Lecture Notes in Comp Science*, 3992, 169-176.

[26] Yaşar, O., Maliekal, J., Little, L. J. and Jones, D. (2006). Computational Technology Approach to Math and Science Education. *IEEE Comp. in Sci & Eng.*, 8 (3), 76-81.

[27] Yaşar, O., Maliekal, J., Little, L. and Veronesi, P. (2013). An interdisciplinary approach to professional development for secondary school math, science, and technology teachers. Submitted to *J. Computers in Mathematics and Science Teaching*.

[28] Creswell, J. W. (2012). Educational Research: Planning, Conducting and Evaluating Quantitative and Qualitative Research. 4th Ed. Pearson Education, Inc.

[29] Fincher, S. and Petre, M. (2005). *Computer Science Education Research*. Taylor&Francis e-Library: London and New York.

[30] Goode, J. and Margolis, J. (2011). Exploring computer science: A case study of school reform. *Transactions on Computing Education*. 11(2).