Computational Math, Science, and Technology (C-MST) Approach to General Education Courses

Osman Yaşar
The College at Brockport
State University of New York
Brockport, NY 14420
Tel: +1 (585) 395-2595
oyasar@brockport.edu

ABSTRACT
In this paper, we present a computational approach to teaching general education courses that expose students to science and computing principles in engaging contexts, including modeling and simulation, games, and history. The courses use scalable curriculum modules organized in layers of increasing difficulties in order to balance learning challenges and student abilities. We describe the computational pedagogy followed in these modules and courses, with particular attention to the simulation-based course, namely introduction to computational science, to present a case study for those considering similar initiatives.

General Terms
General Education, Pedagogy, Games, History, Natural Sciences

Keywords
Modeling and Simulation, Abstraction, Computational Thinking

1. INTRODUCTION
Recent increases in power, access and affordability of digital technology have impacted scientific research, industrial design, and education. Educators stated at the turn of the century that, when used in the context of applications, technology would support higher-order thinking by engaging students in authentic, complex tasks within collaborative learning contexts [26]. More recently, the National Science Teachers Association (NSTA) described computation as a "third pillar" of scientific inquiry, accompanying experiment and theory [19]. It cited a growing body of evidence that using models and simulations, students learn better since they are actively engaged in "doing," rather than passively engaged in "receiving" knowledge. Within the scientific computing community, the role of computation had long been recognized and brought to the classroom through training by many organizations such as Shodor Foundation, and through formal degrees and courses by other institutions [15, 23], including ours [28-33]. However, now that we have help from educators and pedagogy experts to promote computational science education in more fundamental ways, we get a second chance to address some of the challenges we have faced.

While recruitment challenges can be linked to a general lack of interest and preparation by nation's high school students [2, 5-6, 9, 11, 16-17], computational science education requires additional preparation in multiple domains (math, programming, and sciences) that not every college student is willing to undertake. A fundamental (pedagogical) approach, including a focus on computational thinking skills [27], could bring all stakeholders together in a way not only to reform the computing education but also push scientific thinking into mainstream to address underlying causes of the rising category-5 storm in nation's K-12 education [16-17].

The importance of math and computational skills for STEM workforce has been noted in many reports; including the projections by the Bureau of Labor Statistics [3], National Science Board statistics [21], and surveys by the American Institute of Physics [1]. AIP surveys taken at regular intervals (in 1999 and 2010) of physics majors, 5+ years after finishing an undergraduate degree, indicate that some of the important job skills continue to be scientific problem solving, teamwork, computer programming, design and development, simulation and modeling, math skills, and technical writing (See Fig. 1).

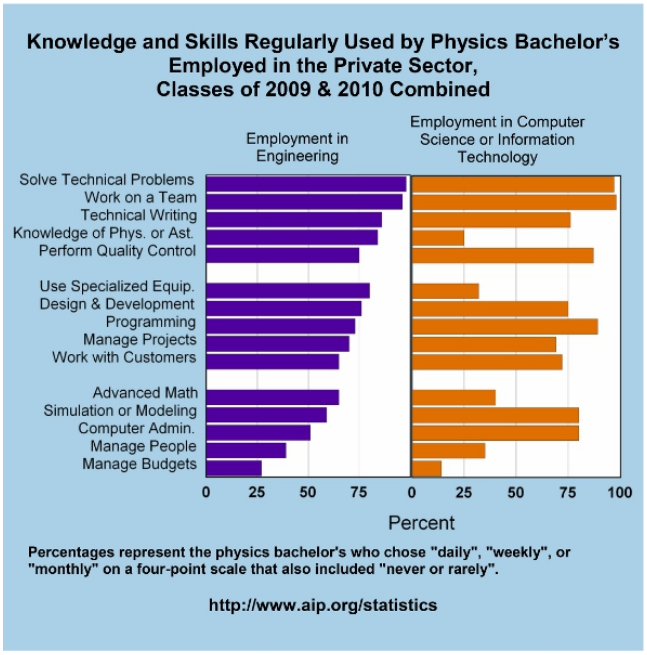

Figure 1: Results of the most recent AIP survey in 2010 [1].

While the demand for computationally competent science, technology, engineering, and mathematics (STEM) workers is an unprecedented opportunity, enrollments have gone down steadily in recent years. The pipeline between institutions of higher education (IHE) and K-12 seems to be broken [2, 5-6, 9, 11, 16-17]. The issue of why science is not as engaging as other subjects is complex, but according to the Relevance of Science Education (ROSE) study, student attitudes towards STEM become increasingly negative as a country advances economically, which suggests this phenomenon to be deeply cultural [22]. 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 science and computing are necessary.

2. COMPUTATIONAL SCIENCE EDUCATION AT BROCKPORT

Established in 1998, the computational science degree (BS and MS) program at Brockport attracted high parameter students and promoted research experience in an undergraduate institution [28-33]. Success stories from alumni hired by the software industry include multiple offers from the same company, offers for significant others as incentives, promotion to senior positions upon hiring, and many more. Those hired as teachers could teach multiple subjects (math, programming, and general science) due to their diverse background. These examples all point out to the benefits of a broad education to improve one's marketability and job orientation at a time when the likelihood of working at a job not related to one's field of study is greater than 50% [21].

While the computational science (CPS) program has attracted to Brockport students who normally would go to a higher tier school, it has graduated only a handful (~5) of students annually. Concerns over the number of freshmen entering the program led to an outreach effort by the program in 2003 to address the IHE-K12 pipeline issue as described earlier. An institute was formed in partnership with local school districts (Rochester City SD and Brighton Central SD) and national organizations (Shodor Foundation, Krell Institute, and Texas Instruments) to train secondary school teachers on the computational approach to math, science, and technology (C-MST). Improved teacher retention and student achievement reported by partnering districts drew national attention to this initiative, including testimony before the U.S. Congress. Over the past decade, institute staff and participants (faculty and teachers) created a large inventory of curriculum modules and lesson plans that are currently being used in the introductory-level general science courses described here.

While the computational approach to STEM education has been recognized a novel strategy to improve the technical workforce, curricular and recruitment challenges have slowed its growth. Brockport has revised its degree programs and courses several times to update and diversify its curriculum in several fronts, including: a) science of computing (simulation tools, programming, parallel computing, numerical and statistical methods, visualization, technical writing, and computing principles), b) science done computationally in application domains, and c) education done computationally (pedagogy, teacher training and K-12 student outreach). While some of the application courses such as *computational-x* (x: biology, physics, etc.) and teacher education courses cover deep (x) content to support STEM majors, others include service courses, such as those described here, under general education category to spread the benefits to all college freshmen across the spectrum. What follows in the next section is the computational pedagogy we used in these courses, particularly in CPS 101, to draw both STEM and non-STEM majors into learning about computing and sciences. We believe that this pedagogical approach is also relevant to a recent initiative by the College Board to implement a new AP course on computational thinking [25]. The impact on the nation's STEM education can be significant.

3. C-MST PEDAGOGY

While 'attention to details' is important to master a skill, 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 [14]. *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 [24]. While abstraction is essential for cognition, there are other benefits all around us. Since the nature itself employs abstraction by hiding the atomic-level motion and the cellular phenomena, we get the benefit of seeing the bigger picture. Computer scientists use abstraction to write large-scale complex codes (such as operating systems, compilers, and networking) where the complexity is distributed into seemingly independent layers and protocols of the code in such a way to hide the details of how each layer does the requested service. We all use abstraction in our daily lives. For example, when we go to a restaurant, we order our meal and not worry about how they cooked it in the kitchen. Those who worry and check it out once or twice cannot possibly afford doing it all the time. Abstraction skills can be improved beyond what was inherited, through training, education, additional knowledge and experience.

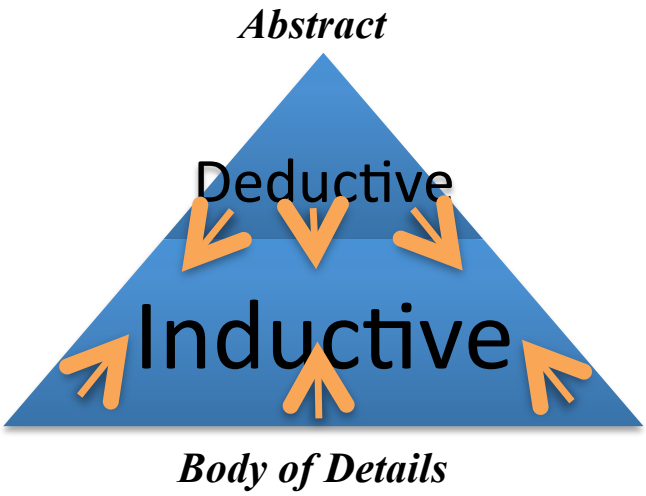

Figure 2: Illustration of the informational organization and the resulting deductive/inductive instructional pedagogies.

Computational modeling uses abstraction by its simplification of the reality. Such simplification helps scientists eliminate certain parameters and focus on what is being studied. Another benefit of computational ***modeling*** is that it supports *deductive* learning. Modeling enables the learner to grasp important facts surrounding a topic before revealing the underlying details. In a sense it helps to do a reverse engineering by gradually leading the learner to the details that support the abstracted knowledge (See Fig 2 for a schematic view of inductive and deductive processes). S***imulation***

adds another level of benefit by providing a dynamic medium for 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. Together, both computational modeling and simulation lead to a *deductive pedagogy* by first introducing a topic from a simplistic framework and then moving deeper into details after learners gain a level of interest to help them endure the hardships and frustration of deeper learning. Such a stepwise progression in learning is consistent with the pedagogical framework *Flow* [8] and scaffolding strategy to balance skills with challenges as illustrated in Fig. 3.

Figure 3: Illustration of Optimal Flow in learning [8].

According to a national report [18], at early stages computational modeling approach to STEM 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). Using existing computational models, instructors can start general science education via games and simulations without exposing them to STEM principles right away. Students can get to modify an existing model, or create one from scratch. Tools such as Interactive Physics (IP) and AgentSheets (AS) can be used to create many fun things that could engage students into science experimentation. They also provide easily discoverable links (i.e., buttons for controlling the run-time and accuracy) to underlying ***principles*** of computational and scientific modeling.

After initial experimentation with modeling in the context of a game or science topic, students can be introduced to a simple principle of mathematical modeling (*new = old + change*) which eventually (and quickly) leads the learner into understanding several aspects of computational thinking [27], including decomposing a problem into smaller chunks, computational cost for more accuracy, and the need to use a programming language in order to handle complexity and increasing number of data points (due to decomposing the problem into much smaller chunks). The mathematical foundation of modeling and simulation can be taught in terms of building functional relationships (such as $y=f(x)$) using the *rate of change* equations and the above algebraic equation ($y_{new} = y_{old} + dy$) where y_{new} and y_{old} are *new* and *old* values of *y,* and *dy* is the *change* from the *old* to the *new*.

As an example, consider finding a direct relationship, $y=f(x)$, based on the rate of change (derivative) $dy/dx= 2x$, and the initial condition $y=0$ when $x=0$. The analytic answer to this mathematical integration is $y = x^2$. However, students can be led to find an answer through *numerical integration* instead, by constructing a table of *(x, y)* data points starting from *(0, 0)* for different choices of increment in *x* values (*dx= 1, 0.5, 0.1,* and so on). When the numerical results are compared to the analytic solution ($y= x^2$) for these cases, students could be led to discover the correlation between the step size (*dx*) and the accuracy of the numerical results: such as *the smaller the dx, the more accurate the answer*. 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 might discover that even Excel cannot be of help 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.

In an after-school project, several 9th graders from Brighton High School (NY) were able to replicate IP results for the harmonic motion (Fig. 4) by first applying Excel (Table 1) and then Python (programming language) to algebraic formulas for the position ($x_{new} = x_{old} + dx$) and velocity ($v_{new}= v_{old} + dv$) of the spring-driven object at times ($t_{new} = t_{old} + dt$) separated by interval *dt*. Here, time (*t*) is an independent variable and *change* in *x* and *v* are: $dx= v \cdot dt$ and $dv= a \cdot dt$, where acceleration *(a)* is *Force/mass*. The force applied by a spring unto an attached box is $F= - k \cdot x$, where *k* is the stiffness coefficient of the spring and *x* is the displacement of the box from the equilibrium position (x=0). The following year, these students modeled orbital motion of planets when the formula for the force, causing the *change*, was given to them ($F=G \cdot M \cdot m/x^2$; where *G* is a Universal Constant, *M* and *m* are masses of the Sun and planet separated by distance *x*).

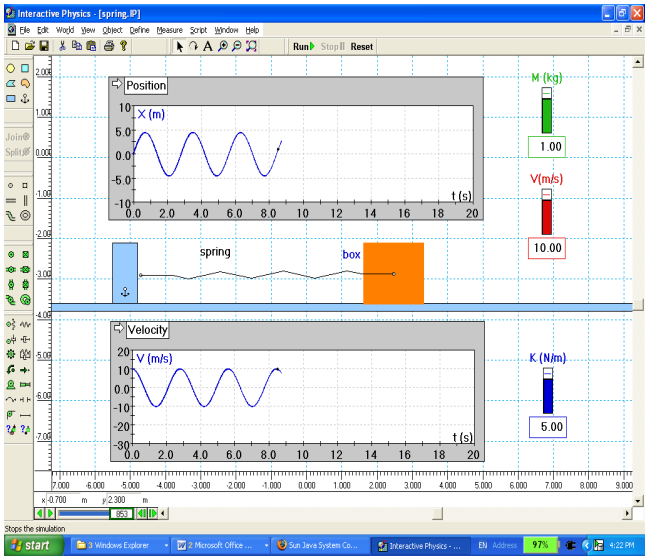

Figure 4: Simple harmonic motion using Interactive Physics.

Table 1: Simple harmonic motion using Excel (dt= 0.125).

t(s)	v(m/s)	x(m)	t(s)	v(m/s)	x(m)
0.000	10.00	0.0	1.250	-8.97	1.49
0.125	10.00	1.25	1.375	-9.90	0.26
0.250	9.22	2.40	1.500	-10.06	-1.00
0.375	7.72	3.37	1.625	-9.44	-2.18
0.500	5.61	4.07	1.750	-8.08	-3.19
0.625	3.07	4.45	1.875	-6.08	-3.95
0.750	0.29	4.49	2.000	-3.61	-4.40
0.875	-2.52	4.17	2.125	-0.86	-4.51
1.000	-5.13	3.53	2.250	1.96	-4.26
1.125	-7.33	2.62	2.375	4.62	-3.69

While these high school students were exceptions, this (deductive + inductive) pedagogical approach [13] does show a ***path*** that can be promoted in both IHE and K-12 classrooms. It starts with a deductive approach by using modeling to introduce the learner to important facts surrounding a topic. Then, by running hypothetical scenarios and investigative projects through simulations, they are encouraged to discover relevant principles of computing and sciences in an inductive fashion. We call the above approach 'C-MST pedagogy' because of the local context; others may call it differently (i.e., Project-First, then Principles just-in-time [11, 20]). Pros and cons of deductive and inductive learning have been a topic for many years in language training. Many contemporary programs use a combination to get double benefit [13].

The computational approach to STEM education encourages inquiry, involves projects, and facilitates team-based instruction. It puts the learner at the center of a constructivist experience. While it uses a deductive approach to engage learners into a STEM topic, it emphasizes the importance of abstraction skills to support inductive learning. By linking computing to science through the computation of ***change***, it provides a motivation for science majors to learn programming and for computing majors to learn more about science. Its motivational and deductive/inductive cycle can be used to broaden participation in computing and sciences among female students [11, 20].

Table 2: Gen-Ed Computing courses and their enrollments

	2010	2011	2012	Total
101 Intro to Computational Sci.	12	20	38	70
105 Games in Sciences	18	52	55	125
302 History of Sci. & Tech.	6	22	25	53

4. GEN-ED COMPUTING COURSES

We have developed and taught three computing-based general science courses, including CPS 101 Introduction to Computational Science, CPS 105 Games in Sciences, and CPS 302 History of Science and Technology. While the primary topic of this article is CPS 101, we want to give a brief overview of all these 3 courses. Launched in 1998, CPS 101 was taught by the author in full capacity (25 students per semester) until 2007 when a new faculty was assigned to teach it. The new instructor's tendency to teach it merely as a programming course with high level mathematics brought the course down to extinction. Through support from an NSF Course Development grant in 2010, the content of CPS 101 was shifted back from 'differential equations and computer programming' to its original content of problem solving at a more fundamental level as described later. This modification brought the course back to life again (see Table 2 for enrollments), vindicating the fundamental computational thinking approach to introductory computing [27] and computational science education [29]. Currently, CPS 101 uses simulation tools such as IP to teach students basic science concepts without having to require a deep level of mathematics and knowledge of the natural laws. The CPS 105 uses AgentSheets (AS; an agent-based modeling tool) to demonstrate science applications in the context of games and environmental issues. The CPS 302 uses an introduction to science and computing in the context of history, again supported by demonstrations using tools such as IP and AS.

Table 3: External/Internal factors affecting enrollments.

How did you hear about this course?								
CPS ↓	Friend		Advisor		Department		Other	
	2011	2012	2011	2012	2011	2012	2011	2012
101	7%	10%	40%	25%	7%	0%	46%	65%
105	34%	34%	18%	4%	2%	0%	46%	60%
302	17%	12%	44%	20%	16%	0%	23%	66%

To meet the needs of students with various backgrounds, a contextual learning is critical for students' success and it improves interest in technology while generating enthusiasm towards sciences [12]. Surveys reveal interesting observations on *student attitude vs. the context* (in which science topics were taught in these courses). They all had a broad appeal; drawing students from 28 departments, including non-science majors. In fall 2010, the College approved these as General Education courses in natural sciences. To meet the Gen-Ed designation in natural sciences, more general science content had to be added to the original syllabus. These modifications seem to have triggered an increase in the enrollment, as shown in Table 2. Interest by non-STEM majors increased. More than 60% of currently enrolled students in these courses are non-STEM majors, while this ratio was about 35% before these modifications when the main focus was on computing and mathematics.

According to the course surveys, the games-based course (CPS 105) was the most popular; 34% of enrollments in this course came from the word of mouth among friends. The evolving self-interest also seems to be high as about 80% of students in this course wanted to take another course on the same topic (See section on Results and Discussion). The simulation-based course (101) and the history-based course (302) were often recommended by student advisors. The Gen-Ed status contributed significantly to enrollments as can be seen from the 'Other' category in Table 3. As we moved from 2011 to 2012, even more students enrolled under 'Other' category as a consequence of either easiness of online scheduling or popularity of Gen-Ed designation; or both. The 2012 data shows a significant increase in students' self interest and a decrease in advisement by faculty and department.

4.1 Course Description: CPS 101

The weekly schedule for CPS 101 is shown in Table 4. Course materials include class notes and user manuals for software tools (such as IP) and the online C-MST curriculum modules open to public at *www.brockport.edu/cmst*. IP is used to model, simulate, and explore a wide range of physical phenomena, including harmonic motion (springs and pendulum), falling objects, trajectory of projectile, energy conservation, orbital motion, Kepler's Laws, Newton's second law of motion, and electrostatic oscillator. Through IP, students are able to build projects, conduct digital experiments, and investigate physical events without deeply knowing or memorizing the laws of physics. Users are allowed to set up their own physical world, choose physical

parameters, monitor the position, velocity, energy, and elapsed time and also create control buttons to facilitate simulation. Visuals images and data from IP can be transferred to geometrical software (such as Geometer's Sketchpad, GSP) to measure angles, distances, and areas needed for proofs or other calculations. IP simulation data can also be transferred in numerical format to Excel for further analysis.

A big advantage of the IP is what you see is what you get. It is interactive and it greatly enhances instruction and helps students build their confidence and success in learning. The use of IP tool is straightforward, and students are able to build their own projects after a couple of weeks training. A screen shot of simulating orbital motion is shown in Fig. 5, where the planets are represented with small circles and corresponding masses. For example, the earth (of mass 5.9 x 10^{24} kg and orbital velocity of 6.65x10^4 mph) is placed at 150x10^6 km from the Sun (mass of 1.89 x 10^{30} kg). The IP images can be transferred to GSP to measure the distances and areas to prove Kepler's laws. For example, Fig. 5 shows the proof of the 3rd law, 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.

Table 4. Weekly schedule for CPS 101

Week 1: Conduct surveys to examine math and CS skills. Discuss survey results. Show videos on the role of computational modeling & simulations in science & industry.
Week 2: Discuss the role of modeling in scientific inquiry and industrial design (show examples). HW #1: Short essay on the 'role of modeling' in research, industry, and education. Computer Lab: Introduce Interactive Physics with examples
Week 3: Computer Lab: Continue Interactive Physics (IP) training. HW #2: Design an IP project that demonstrates students' comfort level with IP.
Week 4: IP Labs: a) Simple harmonic motion: Generate position and velocity plots of a moving object attached to a spring; b) Pendulum: Examine the time it takes for a complete swing vs. initial velocity. c) Falling objects: Examine motion under different gravitational forces on the Earth and the Moon. HW #3: Report effects of elasticity, friction, and air resistance on motion in one of these labs.
Week 5: Discuss principle of mathematical and computational modeling: *new=old+change*. Discuss functional (i.e., y= x^2) and behavioral (i.e., dy= 2x · dx) relationships in tabular, formulated, and graphical forms. Discuss the Rate of Change (ROC) and the difference between Average & Instantaneous ROC. Test #1 (functions, ROC, and forms of representation).
Week 6: Continue numerical integration with examples by hand first and later with Excel. Discuss the role of integration step in reducing error and the role of computational power to afford smaller steps. HW #4: numerical integration by hand.
Week 7: Discuss the role of hardware (storage, processing, and communication) and software (data locality, memory usage, system software, and programming style) on performance and accuracy needed for problem solving. Introduce programming concepts using Python language. Conduct a midterm exam.
Week 8: Break
Week 9: Continue programming discussion with examples in Python. Hands-on experience on programming in a computer lab. Test #2 (factors affecting computational performance). HW #5 (on computer programming).
Week 10: Review use of multiple tools (IP, Excel, and Python) for modeling. Lab: 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 a cause of *change* in velocity & position. Compare IP, Excel & Python results.
Week 11: Lab: Trajectory of Projectile: Use IP & Excel to study 2-dimensional motion. HW #6: Write a Python program that computes the trajectory of a rock thrown up at an angle.
Week 12: Lab: Conservation of Energy & Momentum. Discuss potential & kinetic energy of earlier examples. Use IP to graph potential and kinetic energies of objects. Examine effects of friction and air resistance.
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 of planets. Simulate orbital motion in 2-D using Excel and then Python.
Week 14: Discuss Kepler's laws and use IP to simulate multiple planets around the Sun. Lab: HW #7 (team project: proof of Kepler's Laws). Introduce agent-based modeling using AgentSheets (AS).
Week 15: AS Lab: Model collective behavior of agents. HW#8: Design an AS project. Re-visit HW #1 to improve the essay on 'role of computation'. Review for Final Exam: Discuss and review scientific concepts learned.

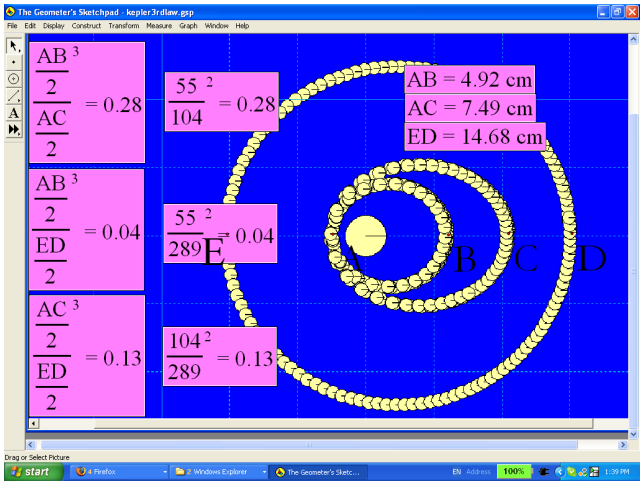

Figure 5: Orbital motion of several planets around the Sun. Orbits and periods are shown to prove Kepler's 3rd Law.

4.1.1 Programming with Excel

While IP is a good tool to expose students to many physical concepts, computational STEM education needs to move beyond just using tools. Our previous experience indicates that students need to eventually understand the underlying mechanism of simulation and modeling 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 physics 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 that is subject to an external force, students need to designate three columns in an Excel worksheet to these variables as shown in Table 1. The columns were cut into half and put side by side for the purpose of fitting the data into a frame for this

article. The first row in each column holds variable names and the 2nd holds initial values *(t=0 sec, v = 10 meters/s* and *x= 0 meters)* and constants *(m= 1 kg* and *k= 5 Newton/meters)*. The 3rd row holds expressions computed in the following order: $t + dt \rightarrow v + (-k{\cdot}x/m){\cdot}dt \rightarrow x + v{\cdot}dt$ where *t, v,* and *x* are linked to their own values on the previous row; except that the value for "*v"* in the $x + v{\cdot}dt$ expression is linked to the newly computed value of *v* (on the same row) in order to move forward updated information. The expressions in the 3rd row 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 whole simulation range when the integration time step *(dt)* is very small. If one chooses *dt* to be 0.000001 sec, then one needs 1,000,000 rows to do an Excel computation for just only 1 second.

In two-dimensions, the above equations need to be expanded to include position, velocity, and acceleration in additional dimensions:

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

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

and

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

Using these algebraic 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), we get the orbital track seen in Fig. 6. At t=0, we assumed that the Earth's orbital velocity was given by v_x=0 and v_y= 29.79 km/s and its position was given by *y*=0 and *x*= 1.50x10^8 km. What is shown in Fig.6 may not be the most accurate track, but qualitatively it is representative of a planet's orbit. Some planets have more elliptically looking orbits. The above calculations are given for *dt* = 5 days, however 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. Computations with higher resolution and automation need use of programming.

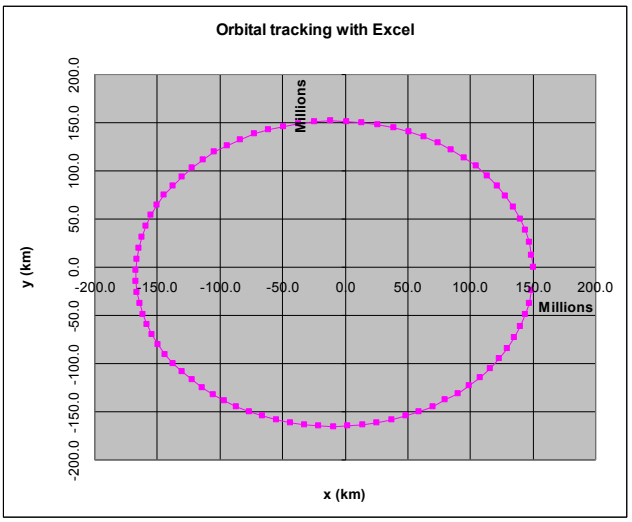

Figure 6: Orbital tracking of the Earth using Excel.

4.1.2 Programming with Python

In the past we used Fortran and C 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 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, or 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. Below is a sample algorithm for one-dimensional Newtonian motion that can be easily extended to two- and three-dimensions.

Input initial position (x), velocity (v), and time (t)
Input time step (dt), maximum time (T), mass (m)
While t <=T:
    Output position (x), velocity (v), and time (t)
    Compute force F & acceleration a = F/m
    Compute velocity in x direction v = v + a × dt
    Compute position in x direction x = x + v × dt
    Update the time t = t + dt
End of While Loop

5. RESULTS AND DISCUSSION

We have used a mix-methods approach [7] to examine the context, pedagogy, and the tool set used in computational gen-education courses. Table 5 and 6 show some of the statistics from student surveys. The tables show a multi-year data for each course. To examine whether there is any statistical difference between 2011 and 2012 responses (due to different instructors, pedagogies, or software tools), we computed z-scores assuming a normal distribution approximation to these binomial surveys [4]. The column p indicates the confidence level that results in each row may be different due to a nonrandom effect. Normally, any confidence level below 90% is less than significant. Plus, it is difficult to infer meaningful results from our research due to small sample sizes (20-25 students per sample). However, by triangulating these survey results with instructors' classroom observations and student grades, and with experimentation of a similar approach with other audiences (K-12 teachers and students), we would like to make a few preliminary conclusions.

Almost all students in the three gen-education courses liked project-based learning, which involved design of a game or a science experiment. More than 95% of them recommend others to take these courses. A significant portion of students (60%-80%) thought that modeling improved their understanding of science concepts and motivated them to pursue additional courses in computing and sciences. Although there are challenges of learning multiple fields in a single course, student skills can grow along with challenges to provide them an optimum *Flow* experience (Fig. 3). While a sizable number of students thought they did not initially have necessary background and skills, they eventually

overcame these difficulties through professor's help, practice, project-based learning, and scaffolding. The level of frustration has gone down in all courses and this may be due to several factors, including the use of more friendly tools, change of instructor in CPS 101, and improvements in the way we teach them. In 2011, the level of frustration was as high as 32% in CPS 101 (due to mathematics and programming), yet the percentage of students who thought that the new skills and knowledge would help them in future courses was also very high. With the change from heavy programming and mathematics to the use of more friendly tools such as IP and AS, students felt more engaged and confident but they did not see a high prospect of using the new toolset and interdisciplinary approach in other classes (down from 96% to 56%). This may be cultural and time needs to pass before it settles like the other two courses where the level of frustration is low (<10%), desire to take another course on the topic is high (70%-80%), and confidence in the later usage of newly learned skills is also high (74%-80%).

Table 5: Survey results from CPS 101. The last column (p) shows statistically the confidence level that there is any difference between responses in 2011 and 2012 for each row. The z scores are calculated based on two proportions [4].

Survey Questions (Q)	Y E S		Difference?	
Responses are in percentages (%).	2011	2012	z	p (%)
1. Recommend this course?	97	94	0.46	35
2. Like to take another course?	44	63	1.20	78
3.Modeling improve science learning?	68	82	1.02	70
4. Like project-based learning?	90	94	0.47	36
5. Had necessary background?	60	75	1.01	70
6. Your skills a match for challenges?	78	82	0.32	26
7. Ever felt frustrated?	32	25	0.49	38
8.Skills may help you in later classes?	96	56	2.96	99
9. Changed your major after?	10	13	0.30	24

Table 6: Survey results for CPS 105 and 302.

Q	CPS105				CPS302			
	Y E S (%)		Difference?		Y E S (%)		Difference?	
	2011	2012	z	p(%)	2011	2012	z	p (%)
Q1	100	100	0	0	100	96	1.01	68
Q2	85	80	0.47	36	50	70	1.44	85
Q3	78	70	0.64	48	61	83	1.73	92
Q4	98	100	0.71	52	95	90	0.67	50
Q5	69	91	1.94	95	88	80	0.77	56
Q6	80	91	1.10	73	100	92	1.44	85
Q7	26	7	1.81	93	20	5	1.60	90
Q8	85	74	0.96	66	89	80	0.88	62
Q9	6	9	0.40	31	0	4	1.01	69

Classroom observations and attendance records from instructors indicate a significant improvement in student behavior and participation in hands-on lab activities. While the attendance rate in the lecture session in classroom was around 70%, it jumped to 90% in the computer lab. They seem highly engaged in lab activities and involved in practicing different computational tools. A different instructor taught the CPS 101 in 2012, and this may have impacted the course dynamics. Other more systematic changes mentioned before (shift away from programming and differential equations) took place before the change of instructors, however, we should note that the new instructor (Sounthone Vattana; MS in Computational Science-Brockport; and MS in Educational Technology-Robert Wesleyan) is a former K-12 teacher and appears to have pedagogical skills with deep content knowledge in mathematics, programming and general science. He also teaches the other two gen-education courses.

Beyond its college-level use as reported above, the C-MST approach has been introduced, through teacher training, into secondary school classrooms in two partnering school districts (Rochester City SD and Brighton Central SD). The content of teacher training and its overall impact on teacher retention and student achievement are being documented in other publications, but here we will briefly mention the surveys on student engagement. Majority of the 200 trained teachers agreed that using modeling tools (IP, AS, Excel, and GSP) in their classrooms increased student engagement. 100% of science and 97% of math teachers agreed that C-MST initiative made math and science concepts more comprehensible to students. Student reaction to modeling (versus traditional techniques) was found to be 97% favorable in math and 77% in science classes. 100% of technology, 72% of math, and 31% of science teachers reported observed improvement in problem solving skills. This order may be linked to low reliance and utilization of mathematical and computational skills in science courses as a result of limited access to computers and possibly lack of available science-related modeling examples. However, while science classes utilized technology less, in instances where it was utilized, it led to a deeper understanding of science topics than it did for math topics (83% in science and 76% in math).

While computational modeling has been shown as an effective pedagogy to expose students to science concepts in an incremental fashion, by using tools that hide the underlying mathematics and science involved in the simulations, it can also motivate them to learn computer programming. By using multiple tools (IP, Excel, and Python) to solve the same problem, students had a chance to weigh advantages of each tool and conclude first-hand that more accurate and faster computation of *new =old + change* for a large number of data points will require computer programming. Additionally, various programming concepts (variables, loops, memory hierarchy, and data types, etc.) are learned in the process. For example, to simulate the orbital motion of an object with Python, a number of variables are needed to store elapsed time, time increment, acceleration, velocity, and position. To predict the velocity and position at the next time step, mathematical operations are used based on the relationships among acceleration, velocity, and position. To find the relations of velocity or position with respect to time, a loop is used to perform repeated calculations. To ensure correctness, the calculations of acceleration, velocity, and position have to be put in sequential while logically right order. Finally, in the context of applications, it is easy for students to understand why and how to learn computer programming. They show more willingness to learn computer programming in order to tackle real-world applications.

A strong link is established between computing and natural sciences through the computation of *change*. For example, change in position and in velocity requires computation of acceleration, which requires knowledge of the Force. This not only links mathematical, computational, and scientific inquiry, it also reinforces in an inductive way the 'learning the fundamentals of laws of nature' and it simplifies the great complexity of the universe into a handful of natural laws (gravity, electromagnetism, and nuclear interactions) that one can learn in a general science course. At the same time, a link between mathematics (numerical integration) and science applications is established, which can be used both in math courses (in the context of *rate of change*, *building functions,* and *modeling*) and in science courses (in the

context of *computational thinking* (CT) as a method of science inquiry). The new K-12 learning standards support teaching of CT skills as early as the 5th grade (See http://www.corestandards.org/ for math and http://www.nextgenscience.org/ for science standards).

6. CONCLUSION

The practice of teaching introductory computing courses in the context of natural sciences (or vice versa) is a promising means of enhancing both General Education and the STEM education. When computational tools are used, students seem more engaged in the class, and their attitude toward learning is more active. 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 sciences at an abstract level, through computation of change caused by natural laws, which projects science in a universal and simplistic framework. Those with curiosity could then acquire a deeper knowledge and pursue additional courses or even a career either in mathematics, computing, or natural sciences. About 50% of non-STEM majors in these reported courses have shown an orientation to add STEM to their education; some (40%) through additional courses and some (10%) through a degree. We believe that solving real world problems by writing computer programs urges students to acquire knowledge through scientific inquiry beyond memorizing laws of science, encourages them to spawn ideas of computational thinking, and fosters them to develop habits of scientific thinking.

A major contribution of computational approach to STEM education is that it integrates math, science and computing in a single unit, exposes and trains students with multiple skills, which are useful in their future careers. While we suggest inclusion of computationally oriented general education science courses for all college freshmen [11-12], attention to a more fundamental concept, computational thinking (CT) at secondary school level is also needed as a long-term strategy [25]. A focus on CT could not only help improve science and computing education at college level but it might also push scientific thinking into mainstream to address underlying causes of the rising Category-5 storm in nation's K-12 education [25, 16-17]. Since abstraction is part of a CT skillset, through modeling and simulations, students can learn not only abstraction skills but also a whole set of other CT skills such as algorithmic thinking, decomposing a problem into smaller chunks, understanding the computational cost for more accuracy, and realizing the need to use a programming language in order to handle complexity and increasing number of data points.

The findings presented here cannot be generalized due to small sample size, limited audience, and lack of reliable and valid instrumentation to assess knowledge, attitudes, and skills. The education research in computing is very new, unlike physics, mathematics, and statistics [10]. We need to pay more attention to findings of learning science. As outlined in [14], we need to: a) draw out and work with the preconceptions and misconceptions of learners; b) help them take control of their learning in a constructivist environment; c) teach subject matter in depth with many examples, and d) employ pedagogical approaches such as metacognition, scaffolding, and project-based learning. The roles for assessment need to be expanded beyond traditional testing to use frequent formative assessment that would help make students' thinking visible to themselves, their peers, and instructors.

7. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (NSF) funds via Grant #0942569. We would like to thank to faculty and teachers whose efforts contributed to the development, teaching, and assessment of the reported courses and materials.

8. REFERENCES

[1] AIP Survey. Important Knowledge & Skills Used on the Job. American Institute of Physics. http://www.aip.org/statistics/trends/highlite/; 1999: /bachplus5/figure2.htm. 2010: /emp3/figure4b.htm.

[2] Augustine, N. 2007. *Is America Falling Off the Flat Earth?* Washington, D.C.: The National Academic Press.

[3] BLS Report. 2010. The Bureau of Labor Statistics. Occupational Employment Statistics. http://www.bls.gov/oes/2010/may/stem.htm.

[4] Brase, C. H. and Brase, C. P. 2012. *Understandable Statistics*. 10th Edition. ISBN: 0840048386. Also, see Stat Trek web site on Hypothesis Test: Difference Between Proportions. http://stattrek.com/hypothesis-test/difference-in-proportions.aspx.

[5] Congressional Research Service (CRS) Report. 2008. STEM education: Background, Federal Policy, and Legislative Action. http://fas.org/sgp/crs/misc/RL33434.pdf.

[6] Computing Curricula 2005: The Overview Report. A Cooperative Project of the Association for Computing Machinery (ACM), the Association for Information Sciences (AIS), and the IEEE Computer Society (IEEE-CS).

[7] Creswell, J. W. 2012 *Educational Research: Planning, Conducting and Evaluating Quantitative and Qualitative Research*. 4th Edition. Pearson Education, Inc.

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

[9] Cuny, J. 2011. Transforming Computer Science Education in High School. *IEEE Computer*, June 2011, 107-109.

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

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

[12] Guzdial, Mark. 2009. Teaching computing to everyone. *Communications of the ACM* 52: 31.

[13] Haight, C. E., Herron, C., and Cole, S. P. 2007. The Effects of Deductive and Guided Inductive Instructional Approaches on the Learning of Grammar in the Elementary Foreign Language College Classroom. *Foreign Language Annals*, 40 (20), 288-301.

[14] How People Learn: Brain, Mind, and School. 2000. The National Academies Press. Wash., D.C. http://www.nap.edu.

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

[16] NAP Report. 2007. Rising Above The Gathering Storm. Washington, D.C.: The National Academy Press. http://www.nap.edu/catalog/11463.html.

[17] NAP Report. 2010. Rising Above The Gathering Storm, Revisited: Washington, D.C.: The National Academy Press. http://www.uic.edu/home/Chancellor/risingabove.pdf.

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

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

[20] Repenning, A. 2012. Programming Goes Back to School. *Communications of the ACM*, 55 (5), 35-37.

[21] Science and Engineering Indicators. 2010. National Science Board. http://www.nsf.gov/statistics/seind10/c2/c2s2.htm.

[22] Sjøberg, S. and Schreiner, C. 2005. How do learners in different cultures relate to science and technology? Results and perspectives from the project ROSE (http://roseproject.no). *Asia Pacific Forum on Science Learning & Teaching*, 6, 1-16.

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

[24] 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.

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

[26] Wenglinsky, H. 2005. *Using Technology Wisely: The Keys to Success in Schools*. New York: Teachers College Press.

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

[28] 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.

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

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

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

[32] 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.

[33] 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.