Incorporating Computational Exercises into Introductory Physics Courses

G. Kortemeyer

Lyman Briggs College and Department of Physics and Astronomy, Michigan State University, East Lansing, MI 48825-1107, USA

kortemey@msu.edu

Abstract. The paper describes the incorporation of computational exercises into introductory physics courses: mechanics, electricity and magnetism, and modern physics. While these courses traditionally emphasize symbolic and numerical calculations, as well as experimental methods, over the last decade, increasingly computational methods are incorporated into the curricula. The talk discusses opportunities for and examples of incorporating VPython projects, as well as logistics, learner collaboration, learning outcomes, and student feedback.

1. Introduction

In the 21st century, Computational physics is emerging as a branch of physics parallel to the formation of theoretical and experimental physics in the early part of the 20th century; consequently, it has found entrance into the mandatory curricula of physics majors, frequently in the form of dedicated (but single and oftentimes isolated) courses [1-3]. There are, however, increasing efforts to also incorporate such dedicated courses into liberal arts curricula, which would reach non-physics majors [4].

Computation and computational modeling, however, are currently not an integrated part of the majority of introductory physics courses. This is regrettable, since for many non-physics majors, these courses may be their one and only exposure to physics – these students will have an incomplete picture of physics, which may easily seem outdated and irrelevant in an information society. Various projects are underway to embed computation into introductory courses [5-10], where a commonly used language is the easy-to-learn and intuitive VPython [11] language. These curricular efforts usually consist of small-scale, low-stakes programming exercises, which are aligned with the physics concepts taught [12].

2. VPython

VPython is a visual extension of the Python language [11]. Most notably, any time "visual objects" are created, they immediately appear in the graphics display. The environment is three-dimensional and can be turned and zoomed as part of the built-in functionality. VPython also implements some basic lighting and shading to add depth to the scene. Figure 1 shows an example of a VPython script; the script launches a ball and lets it bounce off the floor a few times before the program ends.

```
from visual import *
dt=0.01

floor=box(pos=(0,0,0),size=(20,0.1,4))

ball=sphere(pos=(-10,5,0),radius=0.3)
ball.velocity=vector(2,0,0)

g=vector(0,-1,0)

while ball.pos.x<10:
    rate(100)

    ball.velocity+=g*dt
    ball.pos+=ball.velocity*dt

    if ball.pos.y<=0:
        ball.velocity.y=-ball.velocity.y
```

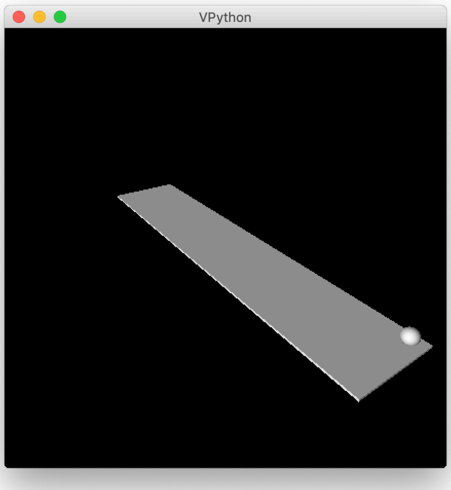


Figure 1. Example of a simple VPython program. The left panel shows the code, the right panel the graphical program output

The first lines initialize a variable `dt`, which will serve as the time-step length, and the objects `floor` and `ball`. The parameters determine the initial position and size of these objects. The object `ball` is assigned an additional property `velocity`, which is set to be a vector in the positive *x*-direction. A vector `g` is initialized to represent gravitational acceleration in negative *y*-direction (VPython programs are oftentimes not "to scale," thus the negative one as gravitational acceleration).

The following loop implements the time steps (in Python, program blocks are indicated by indentation instead of curly brackets like in many other languages – Python is whitespace sensitive). After a synchronization statement (`rate`), in every time step, the ball's velocity and position are updated according to straightforward kinematics. If the ball's *y*-position is smaller than zero (it hits the floor), the *y*-component of the ball's velocity is turned around to make the ball bounce.

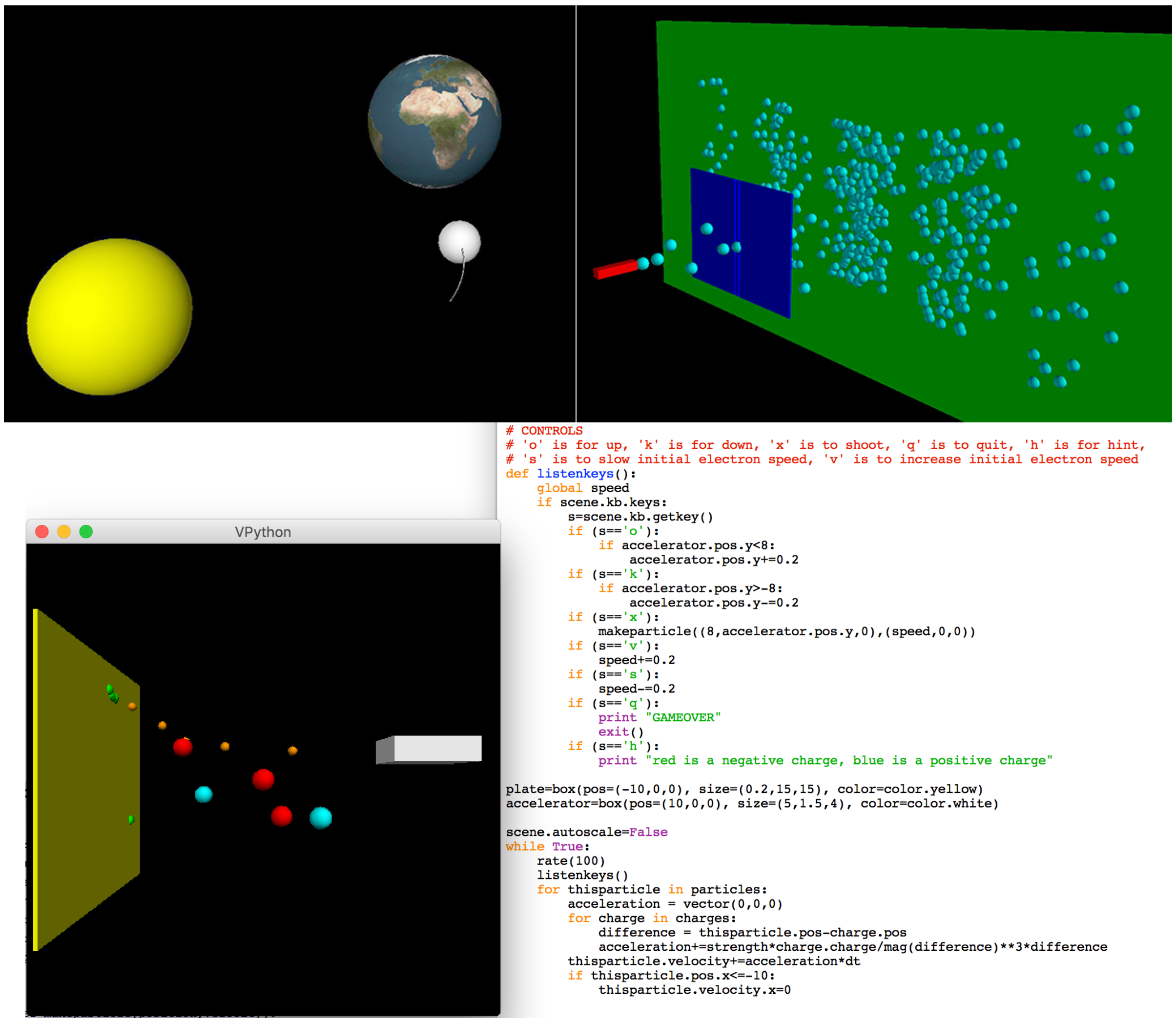
Students could derive many physics insights from the program. For example, the gravitational acceleration never changes, regardless of whether the ball is on the way up or on the way down (or at the highest point of the trajectory). They could also derive the idea of a normal force from the floor from the fact that only the *y*-component of the velocity changes. It may be surprising to them how some calculations that would be prohibitively complicated to carry out analytically can be simulated using nothing but first principles in a few lines of code.

VPython traditionally had to be installed locally as a standalone Integrated Development Environment (IDE); while students find software installation surprisingly challenging, in many respects, this is still the most predictable method. However, VPython can also be run without the need to install any software within browsers using GlowScript [13], and it can be run within Jupyter Notebooks [14], which allow for making worksheets for the learners. Unfortunately, different versions of Python are not necessarily backward compatible, including fundamental functionality such as integer algebra; this can pose challenges in grading assignments.

We are teaching our courses in Studio Physics format, where the programming exercises comprise one of many activities, which also include more traditional problem-solving or laboratories activities. Oftentimes we provide some starter or template code, which the students can subsequently modify and expand. We collect the programs in a drop box in our course management system LON-CAPA [15], which also allows for specifying collaborators at the time of submission.

3. Examples of Exercises

VPython can be used across the whole introductory course sequence, which in the United States usually encompasses Mechanics in the first semester, Electricity and Magnetism in the second semester, and Modern Physics in the third. Figure 2 shows examples of exercises we used in those courses.



```
# CONTROLS
# 'o' is for up, 'k' is for down, 'x' is to shoot, 'q' is to quit, 'h' is for hint,
# 's' is to slow initial electron speed, 'v' is to increase initial electron speed
def listenkeys():
    global speed
    if scene.kb.keys:
        s=scene.kb.getkey()
        if (s=='o'):
            if accelerator.pos.y<8:
                accelerator.pos.y+=0.2
        if (s=='k'):
            if accelerator.pos.y>-8:
                accelerator.pos.y-=0.2
        if (s=='x'):
            makeparticle((8,accelerator.pos.y,0),(speed,0,0))
        if (s=='v'):
            speed+=0.2
        if (s=='s'):
            speed-=0.2
        if (s=='q'):
            print "GAMEOVER"
            exit()
        if (s=='h'):
            print "red is a negative charge, blue is a positive charge"

plate=box(pos=(-10,0,0), size=(0.2,15,15), color=color.yellow)
accelerator=box(pos=(10,0,0), size=(5,1.5,4), color=color.white)

scene.autoscale=False
while True:
    rate(100)
    listenkeys()
    for thisparticle in particles:
        acceleration = vector(0,0,0)
        for charge in charges:
            difference = thisparticle.pos-charge.pos
            acceleration+=strength*charge.charge/mag(difference)**3*difference
        thisparticle.velocity+=acceleration*dt
        if thisparticle.pos.x<=-10:
            thisparticle.velocity.x=0
```

Figure 2. Examples of other exercises.

The top left panel of Figure 2 shows the simulation of a spacecraft moving on a closed trajectory between the Earth and the Moon. The simulation is based solely on gravitational forces and is appropriate for the first semester course. The top right panel shows a double slit experiment for a third semester course; this simulation also incorporates some more advanced numerical methods such as

Monte Carlo. The bottom panel shows a second-semester "pinball game," where charges need to be shot through a random charge distribution to reach a target screen.

4. Collaborations

A concern with programming exercises, just like with any other assignments, is "the problem that won't go away" [16], namely unauthorized collaboration and plagiarism. This problem is arguably aggravated by the ease of copying digital information, and lines are blurred by practices such as the widely accepted and oftentimes desirable "borrowing" of code segments from sites like *Stack Overflow* [17]. Understanding the nature of this problem requires careful analysis of the spectrum from desired and authorized peer teaching and collaboration on the one end to blatant plagiarism on the other end [18]; this has been extensively investigated for physics homework assignments [19-22]. However, particularly for coding assignments, there might be a "fine line" between expected code similarities and the onset of plagiarism [23,24]. Why and when students overstep this boundary has been researched for decades [25–27]. For one of our programming exercises, we analysed the collaboration structure in detail [28].

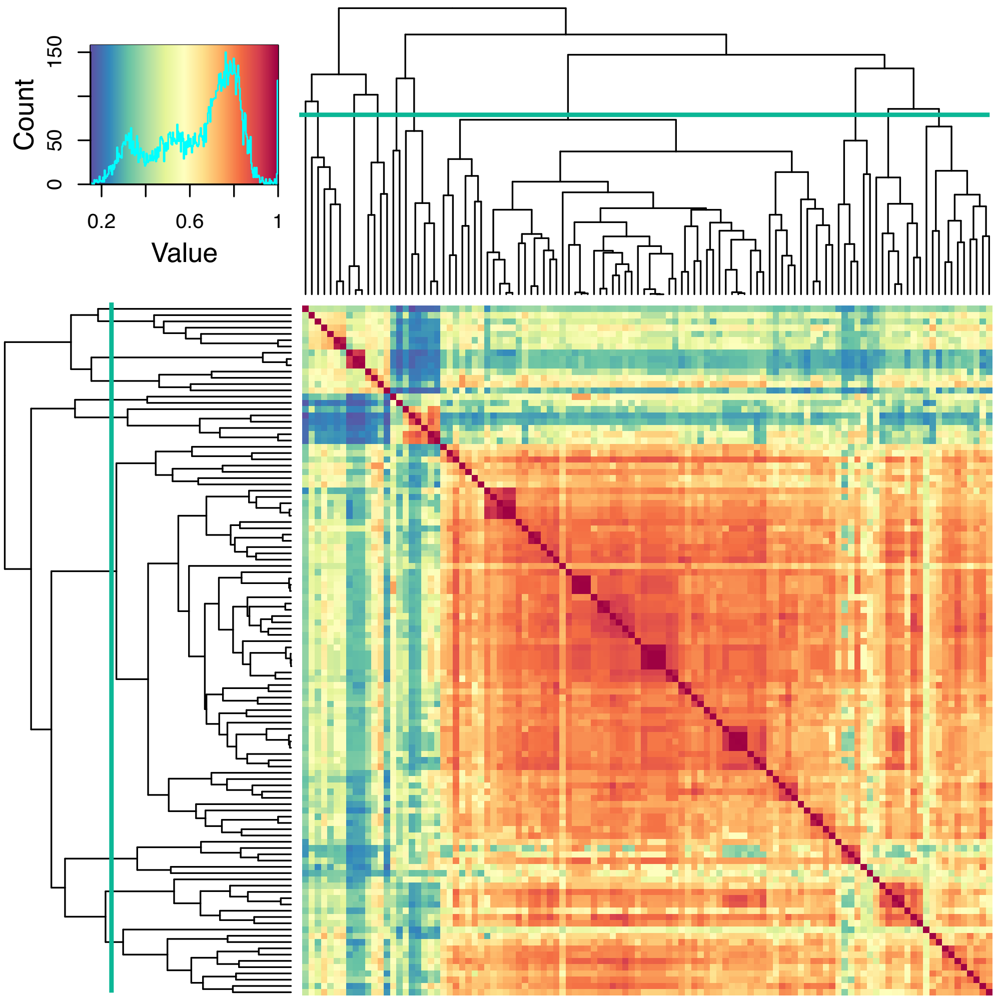


Figure 3. Code similarity heat map [28]. Dark red indicates high similarity, blue low similarity. The blocks are the result of a clustering algorithm, as shown in the dendrograms on top and on the left.

Figure 3 shows a heat map of similarity between student code submissions [28], based on the number of edits to turn one submission into another. Each row and each column represent one code submission. The dark red blocks outside the main diagonal indicate extreme similarity, which indicates blatant plagiarism. There are clearly clusters of more moderate collaboration, which can be visualized using network analysis; Figure 4 shows the result, where each vertex represents one program submission and the edges represent the similarity (thicker edges indicate higher similarity). The colors distinguish identified similarity clusters.

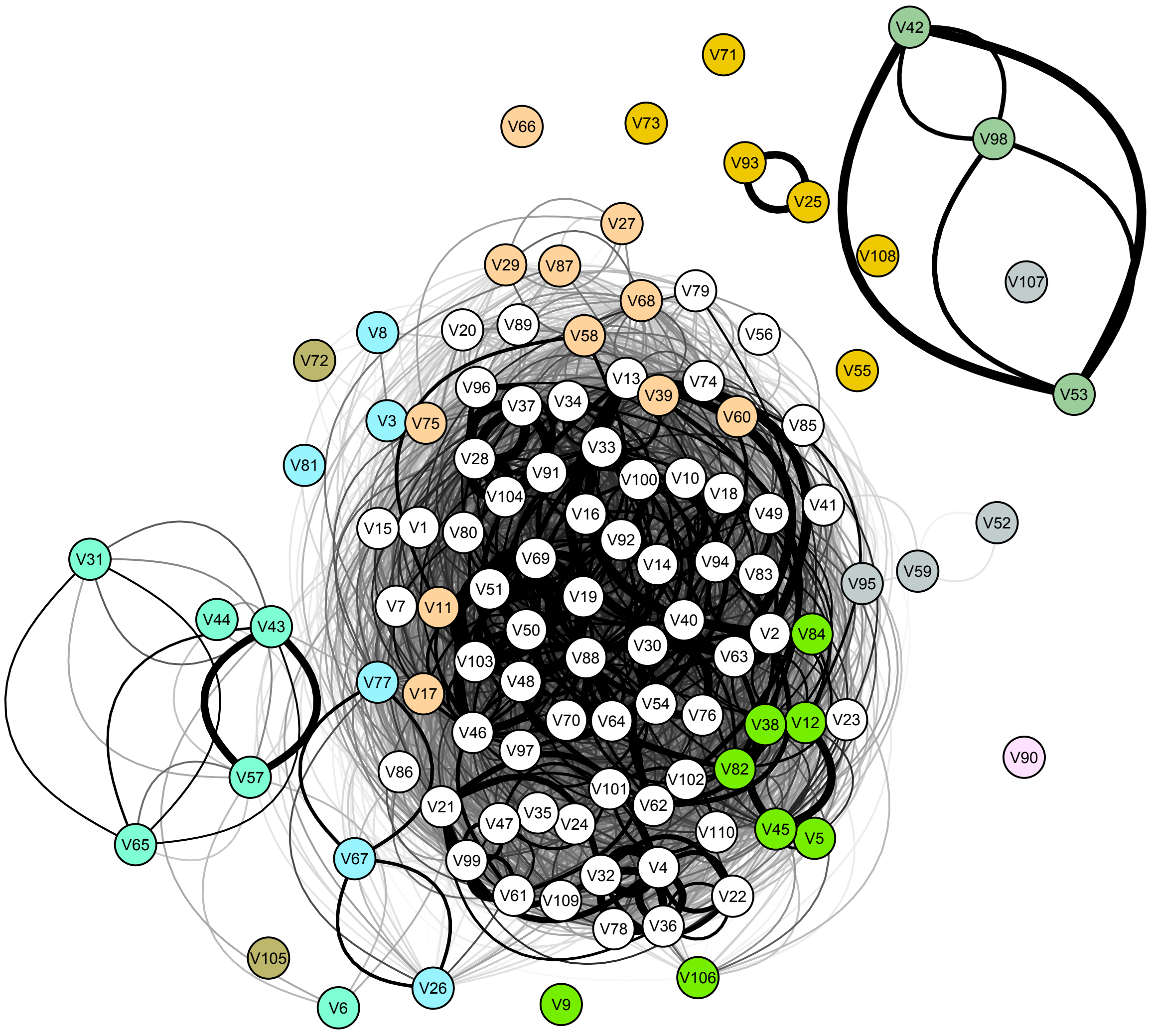


Figure 4. Force-directed network graph using the Fruchterman-Reingold algorithm [29] (graph reproduced from Ref. [28]).

Based in these structures, inheritance relationships can be identified using Minimum Spanning Tree algorithms [30,31]. Figure 5 shows the outcome of this analysis, where the left panel is the full inheritance tree and the right panel is a pruned version that eliminates unstable branches.

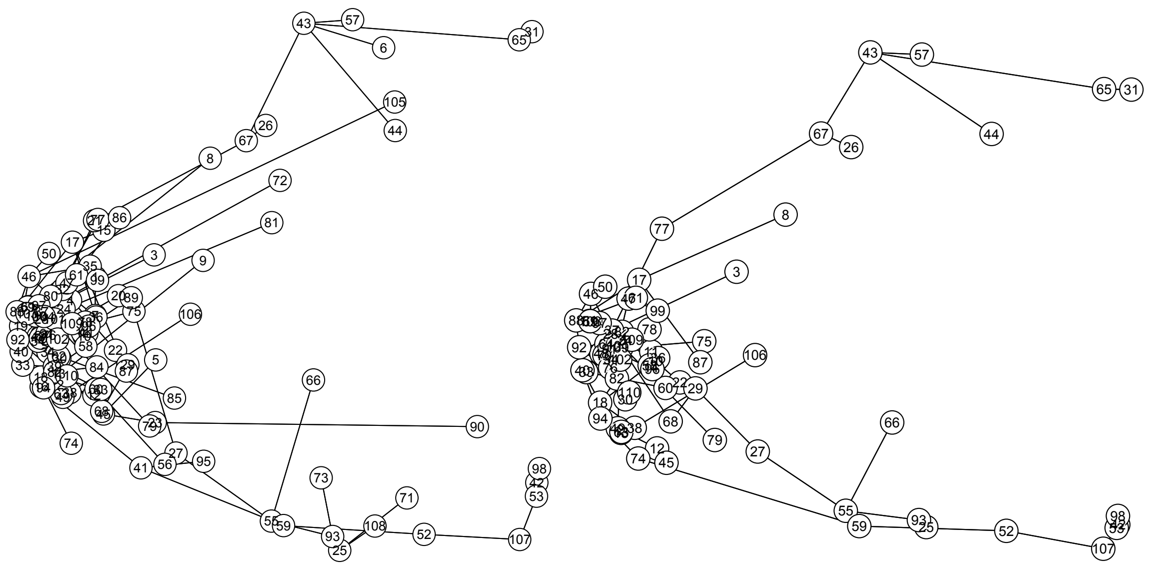


Figure 5. Minimum Spanning Tree [30,31] of the code similarity. The vertices represent student code submissions, where the numbers correspond to the vertices in Figure 4. The left panel shows the full tree, and the right panel a pruned version that eliminates unstable branches (graphs reproduced from Ref. [28])

Students were also asked in a survey how large their collaboration groups were, and most students stated that they worked with two or three peers [28]. Based on Figure 4, actual collaboration groups are typically larger, having six, eight, or even ten members. As it turns out, this may be due to "messenger" students, who provide links between groups, see Figure 5 – single students may carry approaches or code segments from one group to another. Students reflect on their collaborations in free-form survey answers:

- The programming projects are very difficult if I'm alone.
- I usually only collaborate with others if I have a question about a portion of the code.
- It's mostly one person who knows how to do it showing everyone else how to do it.
- I'm usually teaching them how to program and fixing all of their errors.

As a result of our study, in our current semester, we decided to turn the programing exercises into group projects. Preliminary experience shows that formalizing the group collaborative process was a step in the right direction, as student satisfaction appears to have increased.

5. Learning Outcomes

No formal assessment of coding proficiency was carried out in our classes, however, students self-reported on perceived learning outcomes on surveys [28]. A large number of students appreciate the opportunity to learn about simulations and coding. Typical examples are:

- I think programming is a super relevant, useful and exciting skill.
- It is a useful skill that will be helpful in many careers.
- [The exercises] teach a valuable skill set that would not be attained unless taking a computer programming class.

However, in its current implementation, students do not believe they learn physics from the exercises:

- They do not help me with my understanding of physics.
- I don't understand a thing from [the exercises], the visual aid is nice, but it does not benefit my learning at all.
- Maybe try and tie them in more with the lessons.

6. Conclusion

Integrating computational exercises into introductory physics courses is possible, and VPython provides a tool that puts visualization of complex three-dimensional scenarios within reach of non-physics majors. However, many students are overwhelmed by these projects when facing them alone, so they seek out collaborators. Turning the programming exercises into group projects appears to be a step in the right direction for this student population.

While students appreciate the knowledge gained about programming, as well as the visual aspects of the simulations, a more explicit effort needs to be made to analyse the physics within the simulations and help students with the transfer. In particular, more time needs to be spent in discussing how code is frequently based on simple first principles (kinematics and pairwise forces), yet allows to simulate complex scenarios (multi-body systems).

References

[1] A Spencer R L 2005 *American Journal of Physics* **73** 151
[2] Martin R F 2016 *Journal of Physics: Conference Series* **759** 012005
[3] Burke C J and Atherton T J 2017 *American Journal of Phys*ics **85** 301
[4] Dominguez R and Huff B 2015 *Journal of Physics: Conference Series* **640** 012061
[5] Cook D M 2008 *American Journal of Phys*ics **76** 321
[6] Sands D 2010 *New Directions in Teaching Physical Science* **6** 47–50
[7] Serbanescu R M, Kushner P J and Stanley S 2011 *American Journal of Physics* **79** 919
[8] Caballero M D, Kohlmyer M A and Schatz M F 2012 *AIP Conference Proceedings* **8** 020106
[9] Aiken J M, Caballero M D, Douglas S S, Burk J B, Scanlon E M, Thoms B D and Schatz M F 2013 *AIP Conference Proceedings* **1513** 46
[10] Chabay R W and Sherwood B 2015 *Matter and Interactions* (New York: Wiley)
[11] Scherer D, Dubois P and Sherwood B 2000 *Computational Science Engineering* **2** 56
[12] Caballero M D, Kohlmyer M A and Schatz M F 2012 *Phys. Rev. ST Phys. Educ. Res.* **8** 020106
[13] http://www.glowscript.org/ (accessed Nov. 2018)
[14] http://jupyter.org/ (accessed Nov. 2018)
[15] http://lon-capa.org/ (accessed Nov. 2018)
[16] Paldy L G 1996 *J. Coll. Sci. Teach.* **26** 4
[17] https://stackoverflow.com/ (accessed Nov. 2018)
[18] Park C 2003 *Assess. Eval. Higher Educ.* **28** 471
[19] Palazzo D J, Lee Y-J, Warnakulasooriya R and Pritchard D E 2010 *Phys. Rev. ST Phys. Educ. Res.* **6** 010104
[20] Kortemeyer G 2014 *Phys. Rev. ST Phys. Educ. Res.* **10** 010118
[21] Kontur F, de La Harpe K and Terry N 2015 Phys. Rev. ST Phys. Educ. Res. **11** 010105
[22] Busch H 2017 *The Physics Teacher* **55** 422
[23] Joy M and Luck M 1999 *IEEE Trans. Educ.* **42** 129
[24] Mann S and Frew Z 2006 *Proc. 8th Australasian Conf. on Computing Education* **52** 143
[25] Drake C A 1941 *J. Higher Educ.* **12** 418

[26] Nuss E M 1984 *Coll. Univ. Teach.* **32** 140
[27] Simkin M G and McLeod A 2010 *J. Bus. Ethics* **94** 441
[28] Kortemeyer G and Kortemeyer A F 2018 *European Journal of Physics* **39** 055705
[29] Fruchterman T M and Reingold E M 1991 *Software: Practice and Experience* **21** 1129
[30] Kurskal J B 1956 *Proc. American Mathematical Society* **7** 48
[31] Paradis E, Claude J, Strimmer K 2004 *Bioinformatics* **20** 289