3-14-2023

Foregrounding the Code: Computational Chemistry Instructional Activities Using a Highly Readable Fluid Simulation Code

Gianmarc Grazioli
San Jose State University, gianmarc.grazioli@sjsu.edu

Adam Ingwerson
San Jose State University

David Santiago
San Jose State University, david.santiago@sjsu.edu

Patrick Regan
San Jose State University

Hee Kun Cho
San Jose State University, heekun.cho@sjsu.edu

Recommended Citation

Gianmarc Grazioli, Adam Ingwerson, David Santiago, Patrick Regan, and Hee Kun Cho. "Foregrounding the Code: Computational Chemistry Instructional Activities Using a Highly Readable Fluid Simulation Code" *Journal of Chemical Education* (2023): 1155-1163. https://doi.org/10.1021/acs.jchemed.2c00838

JOURNAL OF CHEMICAL EDUCATION

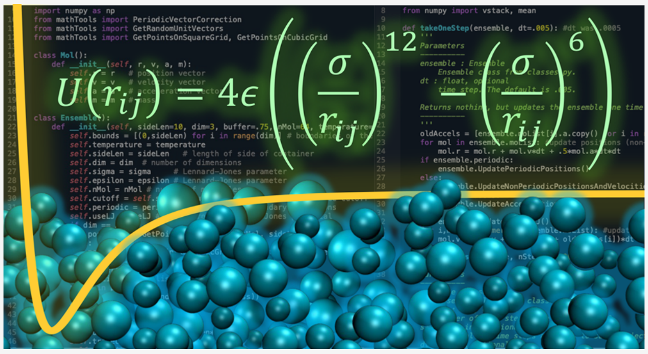
pubs.acs.org/jchemeduc Article

Foregrounding the Code: Computational Chemistry Instructional Activities Using a Highly Readable Fluid Simulation Code

Gianmarc Grazioli,* Adam Ingwerson, David Santiago, Jr., Patrick Regan, and Heekun Cho

Cite This: *J. Chem. Educ.* 2023, 100, 1155−1163 Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

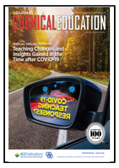

$U(r_{ij}) = 4\epsilon\left(\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right)$

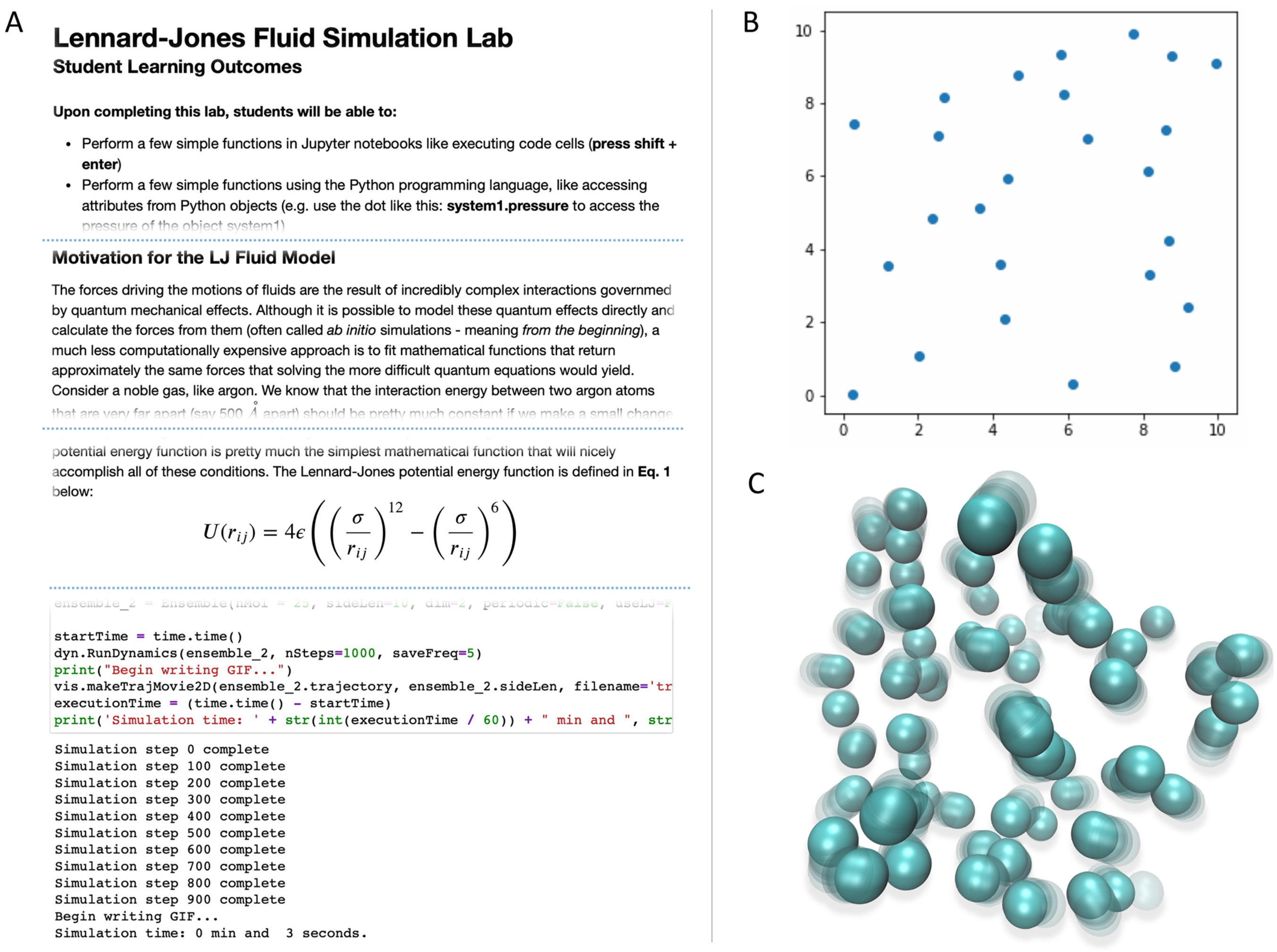
ABSTRACT: Computational chemistry instructional activities are often based around students running chemical simulations via a graphical user interface (GUI). GUI-based activities offer many advantages, as they enable students to run chemical simulations with a few mouse clicks. Although these activities are excellent for introducing students to the capabilities of chemical simulations, the disadvantage is that the students' experience is not representative of how professional computational chemists work. Just as it is important that students in an organic chemistry instructional lab gain hands-on experience with equipment commonly used by professional organic chemists, students of computational chemistry must gain hands-on experience with coding, as professional computational chemists do not rely on GUIs; we write code. Motivated by the need for instructional activities that provide hands-on experience with computer code, a pair of activities were created around a free lightweight (runs on standard laptops) open-source Lennard-Jones (LJ) fluid simulation code written in Python, a programming language that prioritizes readability. The first activity, aimed at undergraduate physical chemistry lab courses, involves students writing Python code in a Jupyter Notebook that is used to run LJ simulations and fit a van der Waals gas model to data produced by the LJ fluid simulations. The second is a jigsaw activity, aimed at advanced undergraduate or graduate students, where students are assigned different sections of the LJ fluid simulation code, and must demonstrate the functionality of their section to the class by both giving an oral presentation and sharing a Jupyter Notebook demonstration of their own design.

KEYWORDS: *Upper Division Undergraduate, Graduate Education, Physical Chemistry, Computer-Based Learning, Distance Learning, Hands-On Learning, Internet/Web-Based Learning, Computational Chemistry, Statistical Mechanics, Theoretical Chemistry, Thermodynamics*

Advancements in computing power, high throughput laboratory equipment, and other technological developments have led to computational methodologies taking on a central role in the way that chemists work in areas ranging from drug development to materials design.[1,2] This paradigm shift has made it paramount that chemistry students develop the skills necessary to efficiently and effectively process, analyze, visualize, and report findings from chemical data using modern computational toolchains. Although software packages like Microsoft Excel can be used for some data processing tasks, these graphical user interface-based (GUI-based) programs quickly become unwieldy for processing large data sets, creating custom data visualizations, and multidimensional analysis.[3] Working with the massive data sets and simulation tools that have become commonplace in the field of chemistry requires proficiency with more powerful coding-based software tools for data analysis and data management, such as the wide variety of software libraries available for Python and R. Python is one of the most popular programming languages in the world, is the perennial favorite as the programming language most frequently mentioned in job descriptions on LinkedIn,[4] and has seen a particularly strong recent uptick in the scientific community.[3] The Jupyter notebook is a powerful interactive coding environment which has also gained popularity in the scientific community for sharing experimental data,[5] as demonstrated by the 2017 Nobel Prize winning observation of gravitational waves from the LIGO and Virgo Collaboration.[6] The development of free open-source chemical education materials coded in Python that leverage the framework of Jupyter Notebooks and JupyterLab is gaining momentum,[3,7−12] and we aim to contribute to that momentum with the educational tools presented herein.[13−15]

Received: August 29, 2022
Revised: January 23, 2023
Published: February 7, 2023





Lennard-Jones Fluid Simulation Lab

Student Learning Outcomes

Upon completing this lab, students will be able to:

- Perform a few simple functions in Jupyter notebooks like executing code cells (**press shift + enter**)
- Perform a few simple functions using the Python programming language, like accessing attributes from Python objects (e.g. use the dot like this: **system1.pressure** to access the pressure of the object system1)

Motivation for the LJ Fluid Model

The forces driving the motions of fluids are the result of incredibly complex interactions governmed by quantum mechanical effects. Although it is possible to model these quantum effects directly and calculate the forces from them (often called *ab initio* simulations - meaning *from the beginning*), a much less computationally expensive approach is to fit mathematical functions that return approximately the same forces that solving the more difficult quantum equations would yield. Consider a noble gas, like argon. We know that the interaction energy between two argon atoms that are very far apart (say 500 Å apart) should be pretty much constant if we make a small change

potential energy function is pretty much the simplest mathematical function that will nicely accomplish all of these conditions. The Lennard-Jones potential energy function is defined in **Eq. 1** below:

$$U(r_{ij}) = 4\epsilon\left(\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right)$$

```
ensemble_2 = Ensemble(nMol = 25, sideLen=10, dim=2, periodic=False, useLJ=

startTime = time.time()
dyn.RunDynamics(ensemble_2, nSteps=1000, saveFreq=5)
print("Begin writing GIF...")
vis.makeTrajMovie2D(ensemble_2.trajectory, ensemble_2.sideLen, filename='tr
executionTime = (time.time() - startTime)
print('Simulation time: ' + str(int(executionTime / 60)) + " min and ", str
```

```
Simulation step 0 complete
Simulation step 100 complete
Simulation step 200 complete
Simulation step 300 complete
Simulation step 400 complete
Simulation step 500 complete
Simulation step 600 complete
Simulation step 700 complete
Simulation step 800 complete
Simulation step 900 complete
Begin writing GIF...
Simulation time: 0 min and  3 seconds.
```

Figure 1. (A) Excerpts from the Lennard-Jones Fluid Simulation Lab Jupyter Notebook. This image displays the beginning of the notebook, describing the student objectives for the lab, including programming skills in python that will be learned throughout the lab, followed by some of the motivations behind the model. The Lennard-Jones potential energy equation is displayed neatly in the Jupyter Notebook (LaTeX typesetting is available for Jupyter) and explained within. Also displayed under the equation is a code cell from the Jupyter Notebook, showing how students can get experience writing and executing code directly in the Jupyter Notebook containing the assignment. (B) A single frame from the animated GIF file outputted by the LJ simulation code, which can be viewed as a movie by opening it in any web browser (see Supporting Information for a movie showing the animated GIF *2D_trajectory_from_GIF.mov*). (C) A 3D rendering of data created by using VMD[28] to visualize trajectories created by the three-dimensional LJ fluid simulation (see Supporting Information for the trajectory file *sample_3D_trajectory.xyz* and a movie of the trajectory *LJ_3D_periodic_boundaries.mov*). Notebook and 2D renderings were generated using Jupyter Notebook version 6.0.3, and Matplotlib version 3.2.2.

Here we introduce two student-centered exercises for teaching computational chemistry: the first is an introductory undergraduate level activity titled "Lennard-Jones Fluid Simulation Lab in a Jupyter Notebook," and the second activity, which is at the advanced undergraduate/introductory graduate level, is titled "Lennard-Jones Fluid Simulation Code Dissection." Both activities use Jupyter notebooks and Python to explore molecular simulations using the Lennard-Jones fluid model. All materials for both activities are freely available online,[13,15] including a roughly 2 h complete video guide on YouTube for the introductory activity,[14] which enables instructors to assign the activity as an asynchronous fully online assignment, greatly reducing any barriers that may be present due to an instructor's personal comfort with computer programming. The LJ fluid simulations are used to demonstrate the application of important physical chemistry concepts such as Newtonian mechanics, the van der Waals equation of state, and the kinetic theory of gases, while minimizing complexity and computing hardware requirements (any laptop is sufficient) thanks to the venerable, simple but effective, Lennard-Jones interaction potential.[16,17] This simplified fluid model is centered around a two parameter intermolecular pair potential that, despite decades of research since its introduction, is still a subject of current interest,[18,19] as is the development of coarse-grained models in general.[20−23] In addition to standalone LJ fluid simulations, the LJ potential is also utilized in many of the most widely used molecular dynamics (MD) force fields used to simulate proteins, nucleic acids, and other complex systems.[24,25] Additionally, many of the concepts covered in the advanced activity, such as periodic boundary conditions and numerical integration of equations of motion, are general to a wide variety of more complex molecular simulation methods, and thus can be useful for aiding students' understanding of how other molecular simulation methodologies function. The Lennard-Jones simulation code introduced in the present work offers an opportunity for instructors to foreground coding for students of computational chemistry not only as it pertains to running and interpreting molecular simulations, but also with respect to the molecular simulation code itself.

In the following sections, we begin with an overview of the materials common to both instructional activities, then give a description of the more introductory LJ Fluid Simulation Lab assignment, followed by a description of the more advanced LJ Fluid Simulation Code Dissection assignment, then describe the results of our assessment of effectiveness for the assignments, and finally offer some concluding remarks to summarize some of the key points and findings from this work.

■ STRUCTURE AND IMPLEMENTATION

Overview

The instructional activities described in the present work were constructed using two software frameworks: the Python programming language and Jupyter Notebook. Python offers several advantages leading to a widely available and easy to learn language, including large community-based tutorials and help forums (e.g., *Github*, *Stackoverflow*), free online courses, and custom coding packages and libraries tailored to specific uses, such as *ChemPy* for writing chemical applications in Python.[11] The coding syntax was created with readability as its main emphasis, and is an object-oriented language, allowing for efficient coding and fewer lines of code compared to other programming languages.[9] Jupyter Notebook is a multifaceted interactive development environment which uses a web browser to run the IDE. Jupyter Notebook supports over 40 coding languages, has a modular and intuitive interface, and is incredibly flexible.[26] Similarly to Python, Jupyter has a large community with easy access to online tutorials and help. Jupyter uses expandable text blocks known as "cells" which can be used to write code or notes in a piecewise fashion. Each cell can be executed individually to test code blocks independently, or cells can be run all at once. The Lennard-Jones Fluid Simulation Lab activity, discussed in the following section, is carried out entirely within a Jupyter Notebook, including but not limited to communicating student learning objectives, background information on the activity, running chemical simulations, data visualizations, and a few basic coding exercises. In keeping with best practices in writing flexible, modular, and reusable code,[27] the LJ simulation code created for the instructional activities in the present work is split between different files, within which the functionality and data storage is organized into classes of objects, such as molecules (Mol()), which store data like the positions and velocities of a particular LJ particle, and ensembles (Ensemble()) that store information about entire systems of LJ particles, like the number of particles, size of the container, whether the container includes periodic boundary conditions, etc. For an in-depth look at the Lennard-Jones Fluid Simulation Lab activity, the reader is encouraged to view a PDF of the entire Jupyter notebook (included in the Supporting Information as *LennardJonesFluidLabJupyterNotebook.pdf*), as well as download the Jupyter notebook and supporting Python code from GitHub.[13,15] Figure 1 is also helpful for giving a sense of how the activity is structured within a Jupyter Notebook.

Lennard-Jones Fluid Simulation Lab in a Jupyter Notebook

The Lennard-Jones Fluid Simulation Lab exercise is a guided, student-centered, hands-on experience aimed at giving undergraduate students an introduction to running and interpreting molecular simulations. The assignment is fully contained within a Jupyter Notebook, runs locally on any laptop, and requires the students to learn to write some simple Python code to both run the molecular simulations and interpret them. Another option, which is especially helpful for students with limited RAM on their devices, is to run the Google Colab version of the Jupyter Notebook (a link is included on the GitHub repository[13]), which enables users to utilize free computing resources hosted by Google to run the simulations. Video tutorials for both installing the requisite free software (Anaconda—a popular distribution of Python among scientists), and carrying out the entire lab are freely available on YouTube.[14,29] Although the video demonstration accompanying this activity is roughly 2 h long, instructors can typically expect this activity to take students between 2.5 and 3.5 h to complete, as students will need to pause the video frequently to work through the different steps of the activity. The pacing of the activity makes it ideal as a computational lab activity for undergraduate physical chemistry lab courses. It is recommended that instructors assign the 4 prelab questions in advance of completing the lab activity and dedicate about 10 min of classroom time to going over the students' answers before beginning the computational lab activity. In particular, instructors should ensure that students have completed prelab question 4 correctly, as the volume of the 3D Lennard-Jones particle is needed for fitting the van der Waals fluid model to the Lennard-Jones simulation data. The video demonstration makes it possible for instructors to assign the activity fully asynchronously, although instructors will ideally want to make themselves accessible for answering questions. One in-person teaching approach is to have students work in pairs using the video as their guide, with the instructor moving from group to group answering questions as they arise. Student learning outcomes (SLOs) for this activity are that students will be able to

1. Write and execute basic Python codes inside of a Jupyter Notebook.
2. Explain the effects of the LJ potential and periodic boundary conditions on the interactions between particles.
3. Optimize the *a* parameter of a van der Waals fluid to fit data from a set of Lennard-Jones fluid simulations.

The first outcome is measured throughout the activity, the second is measured in lab question 3, and the last is measured by the last 7 steps of the activity.

Section 1 of the assignment consists of instructions on basic python programming such as importing the necessary libraries using the "import" command, as well as four prelab questions. The prelab questions are designed to demonstrate some of the basic plotting capabilities within a Jupyter Notebook, which the students must interpret, as well as to test the students' comprehension of the background information provided. The prelab questions also involve using fundamental Jupyter notebook functionality such as markdown cells and code cells. The second prelab question has students create a new code cell and write a print statement to display the ϵ and σ parameters for an Ensemble object. Constant reinforcement of the physical chemistry and mathematical concepts behind the simulation code is maintained, and in the second prelab question students are required to write out the LJ potential function on paper and plug in those sigma and epsilon values to solve for the equilibrium bond length r_{ij} between particles i and j after some basic calculus (the minimum energy length is

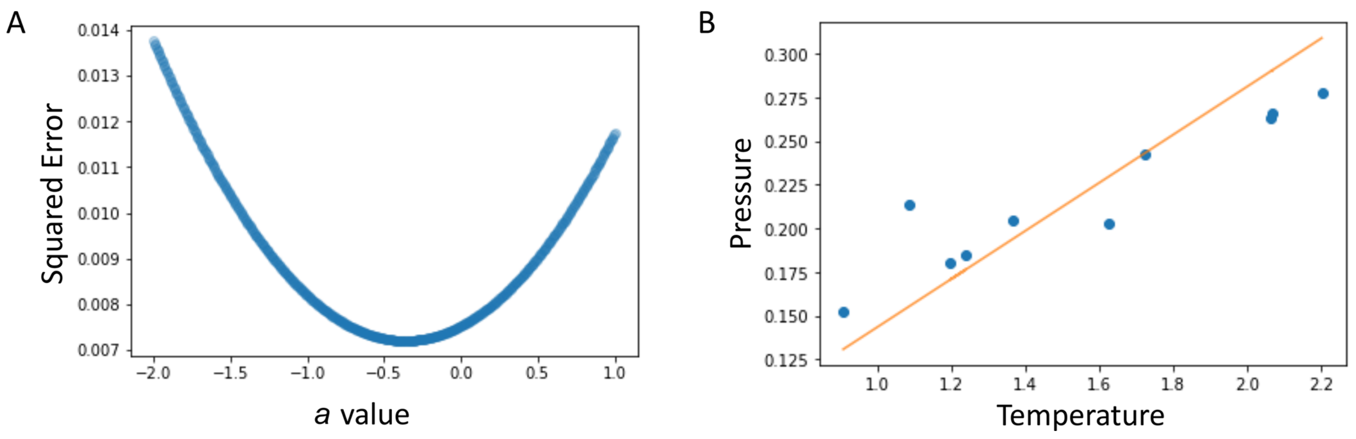


Figure 2. A) Plot showing the squared error between the pressures predicted by the van der Waals equation for ten different temperatures and the pressure measured from Lennard-Jones fluid simulations, at the same ten different temperatures, as a function of the van der Waals parameter *a*. It is easy to see that the minimum error is within the bounds of the highest and lowest *a* values that were plugged into the van der Waals equation. The students are then instructed to write a short Python code to extract the *a* value that produced the smallest squared error. (B) Points indicate pressure as a function of temperature as measured in the LJ fluid simulation, and the line indicates pressure values predicted by the optimal van der Waals model. Plots were generated using Jupyter Notebook version 6.0.3, and Matplotlib version 3.2.2.

where the derivative of the potential energy function $V(r_{ij})$ equals zero). The students are then encouraged to develop their own physical interpretation of this behavior, as the particles can be thought of as deformable spheres (or circles in a 2D LJ fluid) that are weakly attracted to each other. Prelab question 3 demonstrates how changing the parameters of the LJ potential affects the behavior of the potential energy function. This helps reinforce the concept of finding the best possible set of parameters that can accurately model the behavior of physical particles with van der Waals force-like interactions, without the computational cost of explicitly modeling the quantum mechanical effects on electronic structure that would produce similar behavior.

As is the case with nearly every programming language, Python allows users to add comments to code that are not interpreted by the Python interpreter as code, which also allows programmers to temporarily disable lines of code by "commenting them out." Section 2 begins with commented-out code that students will read line by line, along with an explanation in the provided video lesson. Once they have restored the commented out code, and understand what the code is doing, the simulation is run. This code creates an animated GIF file that displays 2D LJ particles moving in a simulation that can be viewed by opening the GIF in any web browser. Following this simulation, two lab questions are asked. The first lab question tests the students' understanding of periodic boundary conditions based on the behavior they observed in the animated GIF file. The second lab question has students plot the potential and kinetic energies of the system, along with the total energy of the system. This demonstrates that the conservation of energy is maintained in the system in perfect balance as energy transfers between the total kinetic energy and total potential energy of the particles (the system is a microcanonical, or NVE ensemble). Another simulation is then run which is set without periodic boundary conditions, and again produces a GIF file as an animated data visualization. By creating similar simulations that differ only in the turning off of the periodic boundary conditions and the forces due to the LJ potential, students can dynamically visualize the difference that these conditions make on the simulation and the resulting data. Lab question 3 reinforces this by asking students to explain why the second simulation's particles behave like an ideal gas, rather than an LJ fluid. Lab question 4 requires students to uncomment a line of code that outputs all the keys from the python dictionary, "ensemble_2," and is intended to both introduce students to an important class of objects in Python and to encourage students to explore the simulation code further. The Lennard-Jones fluid lab activity culminates with the students running a set of 10 LJ fluid simulations at different temperatures, and then using the temperatures and pressures of those simulations to fit a van der Waals fluid model to the LJ fluid data. This portion of the activity can provide a valuable entry point for instructors to cover topics ranging from the development of coarse-grained models to explaining how different levels of theory that can be used to describe the same system, depending on the use case for the model. The approach leverages the fact that the van der Waals equation

$$\left(p + \frac{a}{v^2}\right)(v - b) = k_B T \tag{1}$$

where $v = V/N$ (N is the number of particles and V is volume), p is reduced pressure (P/N), k_B is Boltzmann's constant (here, k_b is set to unity, a commonly used approach for simplifying simulations of this type), a and b are the van der Waals corrections to pressure and volume due to van der Walls type forces and the volume occupied by the particles themselves, respectively, and T is temperature, can be rearranged to the form

$$p = \frac{-ab + av - Tv^2}{bv^2 - v^3} \tag{2}$$

This form allows one to generate plots of pressure as a function of volume, assuming, of course, that one knows the values of a and b! Since the students calculated the volume of a Lennard-Jones particle with both LJ parameters set to 1 in the beginning of the lab, the value of b has already been determined; thus, all that remains is to find the optimal value for a that yields the van der Waals model that best fits their LJ simulation data. The students then use functions provided in the Jupyter Notebook to plug 500 different values for a into the van der Waals model and measure the squared error between each van der Waals model and the simulation data to show that the best van der Waals model is the one with the smallest squared error (Figure 2). Although optimization of the model could have been carried out by using an optimization function from an existing library, this would also result in an abstraction of the optimization process away from the students. Since one of the primary goals of the exercise is transparency, having students write their own simple optimization code, that systematically tests a grid of potential parameters and selects the value of minimal squared error does more to reinforce the students' understanding of what it means to fit an optimal model to a data set.

Thus, by taking an opposite and complementary approach to most computational chemistry classroom exercises, where the process of initializing and interpreting complex chemical simulations is simplified for the sake of accessibility, we have instead constructed an assignment around a more accessible model of a chemical system so that students are able to work more closely with the coding aspects of the simulation, and build better intuition for what goes into constructing a chemical simulation. In fact, we suggest that instructors encourage students to explore the simulation code itself. This could mean simply reading the code to try and understand how it works, or even probing the code by making modifications to it and observing the effects. Given that there are zero safety concerns with a computational lab, the code is lightweight and self-contained, and a fresh version of the code can always be redownloaded from GitHub, this lab offers a unique opportunity for inexperienced students to be bold in their experimentation! Instructors interested in adopting the LJ simulation lab activity are also invited to adjust the difficulty of the assignment by editing the Jupyter Notebook prior to sharing it with the students (e.g., adding additional code snippets to make the activity easier, or taking away some of the code snippets and hints to increase the difficulty).

Lennard-Jones Fluid Simulation Code Dissection

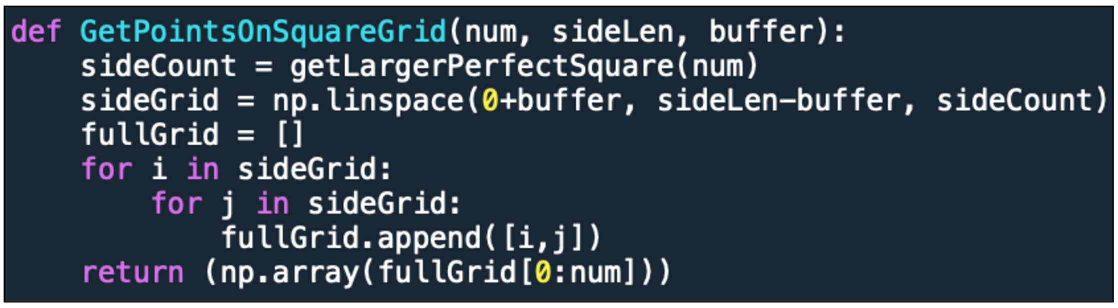
This assignment derives its name from dissection activities common in biology lab courses, in that, just as removing and examining organs from an organism can aid in understanding how each contributes vital functionality to the organism, importing individual modules from the LJ simulation code into a Jupyter notebook allows students to explore their functionality in an isolated environment, where they can experiment with aspects of the functions that they would not even be aware of while using the LJ simulation code to simply carry out simulations. The approach for the LJ fluid simulation code dissection activity is straightforward: divide the LJ simulation code into chunks, where each chunk encapsulates some functionality within the LJ code that contributes to its overall functionality, and then assign the different chunks to individual students or pairs of students to present to the rest of the class. This practice of dividing the subject matter into separate parts to be understood in-depth by individual subgroups and later collaboratively recombined to produce a synthesized understanding of the subject matter as a whole by the class is a well-established technique in education research called a "jigsaw."[30,31] In addition to being an excellent vehicle for active learning, jigsaw activities have an added benefit of fostering a sense of community and cohort building in the classroom. In fact, the motivation behind their creation in the late 1970s was to foster unity in classrooms following the end of racial segregation in American public schools.[32]

It should be noted that the code dissection activity is not an introductory activity, or one that can be completed during one or two class meetings. The code dissection activity is best implemented as a multiweek project, and assigned only after Python programming basics have been introduced. In the Introduction to Computational Chemistry course taught by the lead author, after spending the first 5 weeks of the semester covering some of the basics of Python programming and molecular simulation fundamentals, the Lennard-Jones Fluid Simulation Lab was assigned and completed during week 6. During week 7, a brief overview of the different functions that make up the Lennard-Jones fluid simulation code was given by the instructor, and students were given the opportunity to express interest in presenting portions of the code they found particularly interesting. The remainder of the class time during week 7 and all of week 8 was used for individual meetings between the instructor and students so that the instructor could provide guidance. The students gave oral presentations with accompanying Jupyter Notebook-based demonstrations during week 9. In the interest of providing context, the Course learning outcomes (CLOs) for the Introduction to Computational Chemistry course, where the first author originally introduced the code dissection activity, included that students:

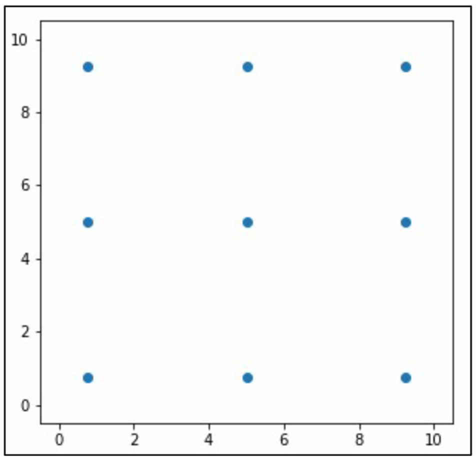
1. Become conversant in methodologies from computational chemistry used in research and industry for analyzing the dynamic structure and function of molecules.
2. Develop the skill of teaching one's self-to write computer code and use new software tools to solve chemical problems.
3. Improve written and verbal communication skills as applied to topics in computational chemistry.

This assignment effectively served all three of these objectives by coupling the students each carrying out an exploratory "dissection" of different portions of the LJ simulation code with a class presentation given by each of the students. In giving their presentations, students were responsible for explaining to their peers how their assigned portion of the code works, both within the context of computer programming, i.e., what each line of the code is doing programmatically, as well as how their portion of the code contributes to the overall functionality of the LJ simulation. In addition to giving a 15 min presentation using slides, the students assigned to each component were also required to create a new Jupyter Notebook specifically designed to demonstrate how their portion of the code works and distribute them to the rest of the class. Students were given the option of either giving a more interactive presentation, where their classmates would be interacting with the Jupyter Notebook they created for the demonstration, or giving a more slide-based presentation and providing the Jupyter Notebook for their classmates to review later at their own pace.

Carrying out a case study in programming molecular simulations in this manner serves several purposes. For one, there is no substitute for first hand experience working directly with a molecular simulation code. Although there is value in discussing with students how molecular simulations work and even having them answer exam questions about them, this can limit student learning outcomes (SLOs) to the lower levels of Bloom's taxonomy.[33] By requiring students to go beyond understanding the code, and moving on to evaluating how their portion of the code functions under different scenarios

A)


```
def GetPointsOnSquareGrid(num, sideLen, buffer):
    sideCount = getLargerPerfectSquare(num)
    sideGrid = np.linspace(0+buffer, sideLen-buffer, sideCount)
    fullGrid = []
    for i in sideGrid:
        for j in sideGrid:
            fullGrid.append([i,j])
    return (np.array(fullGrid[0:num]))
```

B) GetPointsOnSquareGrid(9, 10, .75)


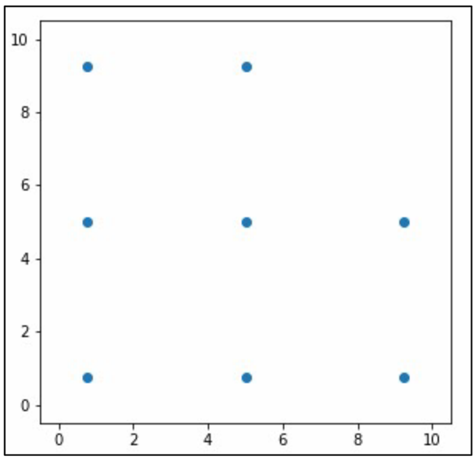
C) GetPointsOnSquareGrid(8, 10, .75)


Figure 3. (A) Sample code showing the function "GetPointsOnSquareGrid()," from the Lennard-Jones fluid simulation code. This function assigns initial positions of all particles in a 2D LJ fluid simulation. (B) This nine-point cubic grid was produced from the function in panel A using the inputs 9, 10, and 0.75 (number of particles, side length of the container, and distance from the walls for the particles on the edge). (C) This eight-point incomplete grid was produced from the function in panel A using the inputs 8, 10, and 0.75. Note how the function determines the positions for the nearest perfect square number of points (9) and returns all but the last to give 8 positions. Plots were generated using Jupyter Notebook version 6.0.3, and Matplotlib version 3.2.2.

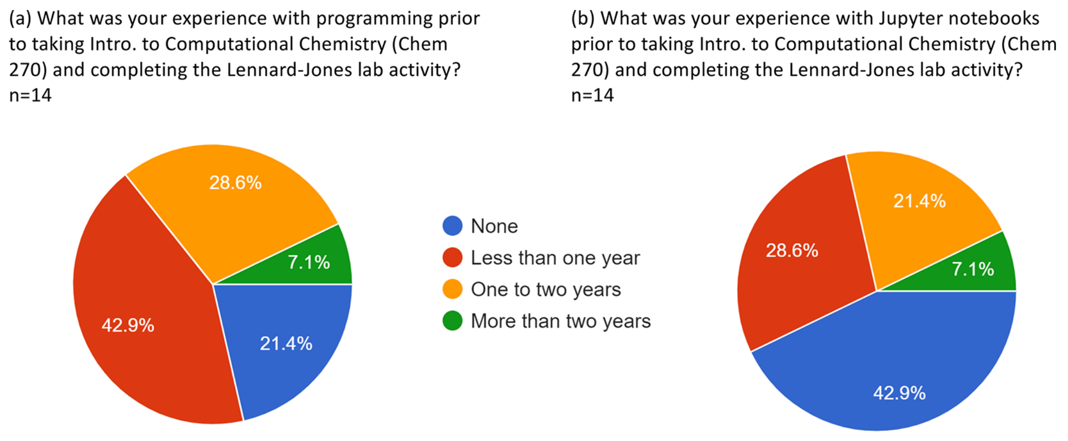


Figure 4. Survey results indicating students' experience with programming prior to the course where the students completed the activities.

that they create to test the code, and to even create a Jupyter Notebook of their own design for the purpose of demonstrating the functionality of their portion of the code, motivated students will reach the highest levels of learning upon completion of this activity. Further benefits include: a better appreciation for structuring readable, modular, reusable code that can be developed in a team environment (of key importance in industry), practice using integrated development environments (IDEs) like *Spyder* (recommended, as it is included in any Anaconda installation) to develop code intended for collaborative projects, the well-established benefits of group work for student learning,[34] and even improved efficiency in the use of class time, leaving more time for questions and discussion.[35] In order to get a sense for the high degree of readability of the LJ code, and how a student might demonstrate the functionality of their assigned function, the reader is directed to Figure 3. The function *GetPointsOnSquareGrid()* assigns the initial positions of 2D LJ particles in an ensemble on a square grid, a task likely taken

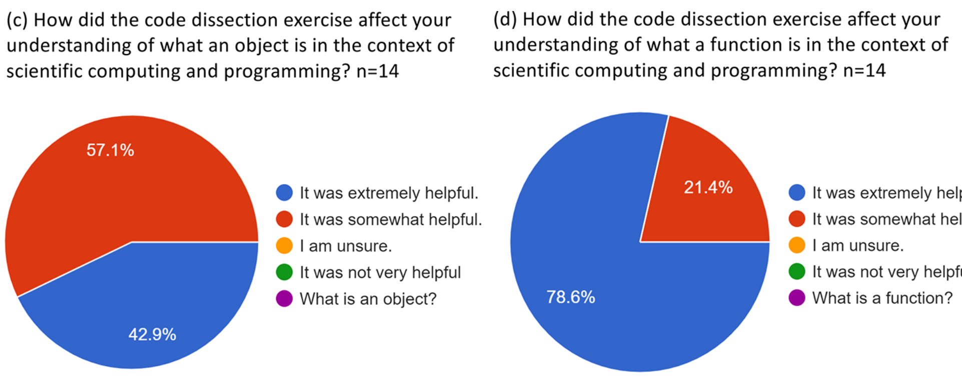
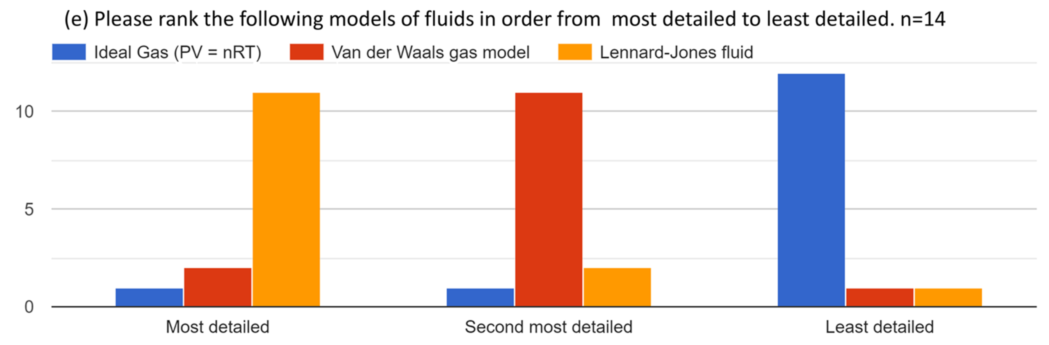


Figure 5. Survey results indicating students' feelings on how the code dissection aided their understanding of scientific computing concepts.



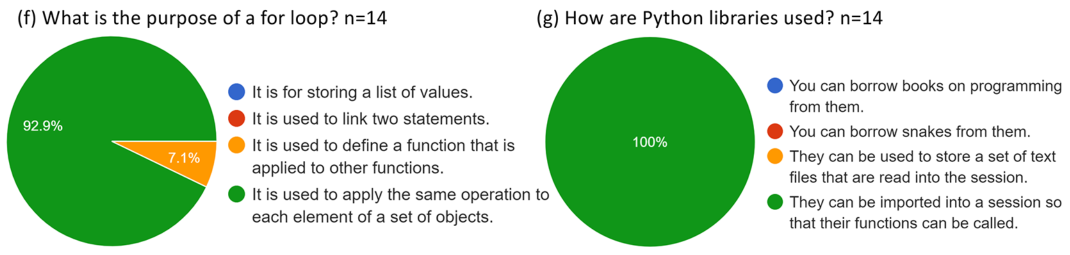


Figure 6. Survey results testing students' comprehension of computational chemistry topics.

for granted by students who have never been directly exposed to molecular simulation code.

The authors strongly suggest that any instructor planning to assign the code dissection should first assign the Lennard-Jones simulation lab activity, as the students should experience the code as a user before they experience it from a development perspective.

■ ASSESSMENT OF STUDENT EXPERIENCE

In order to assess the student experience for both activities, a survey was administered to the students in the Introduction to Computational Chemistry graduate course, which was taught in the online modality. All survey results were submitted anonymously and voluntarily at the end of the semester by students who had completed both activities as graded assignments while enrolled in the Introduction to Computational Chemistry course. Students were informed that the data collected anonymously in the survey would be used in the present study, and that consenting to participate in the study by completing the survey was optional and would not impact their grade in any way. The response rate was 82%. Although the course was listed as a graduate course, it was also open to interested undergraduate students, resulting in a number of undergraduates enrolling in the course as well. Of the students who responded to the assessment survey, 35% were undergraduate students and 64% were graduate students.

The programming experience of the students going into the course varied from none at all to two or more years of prior programming experience to taking the course (Survey Results (a), Figure 4). Additionally, prior experience with using Jupyter Notebooks varied similarly to programming experience (Survey Results (b), Figure 4). After the completion of the Lennard-Jones lab and code dissection activity, the students' perception of their understanding of concepts associated with the assignments was improved. The use of functions and object-oriented programming can be challenging topics for students new to programming, and the survey indicates that the students' perceived understanding of the use of objects and functions in the context of scientific computing and programming improved as a result of completing the two assignments (Survey Results (c) and (d), Figure 5). Additionally, conceptual questions related to Lennard-Jones fluids and the Python language were included in the survey to gauge students' comprehension of these topics (Survey Results (e), (f), (g), and (h) in Figure 5 and Figure 6). The professor for the course in which these activities were introduced also observed a marked improvement in the quality of students' code when comparing coding assignments prior to this assignment with the students' final projects immediately following the code dissection activity. One of the most exciting survey results is that the majority of the class plans to continue developing their newfound skills as scientific programmers (Survey Results (h), Figure 7).

(h) How likely are you to continue to explore the capabilities offered by computational tools that require some level of facility with computer programming (e.g. the Python language, Jupyter notebooks, Python libraries like rdkit Chem, etc.)? n=14

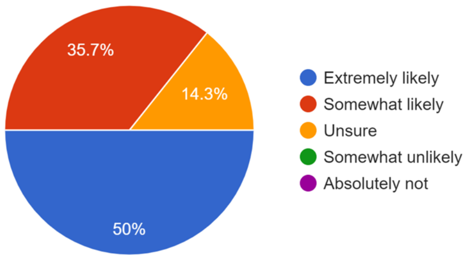


Figure 7. Survey results indicating students' plans for continuing to develop the skills they learned while completing the activities.

Survey Results

CONCLUSION

Two new instructional activities in computational chemistry were introduced, which are centered around a Lennard-Jones fluid simulation Python code that is designed for maximal readability and to encourage student curiosity and exploration into how molecular simulations work. The Jupyter-based lab activity is ideal for undergraduate physical chemistry lab courses, while the code dissection activity is ideal for either an advanced upper-level undergraduate course in computational chemistry or an introductory graduate course on computational chemistry. Both activities are focused on exposing the students to computer code used to simulate chemical systems, rather than insulating them from it through the use of graphical user interfaces (GUIs). The activities discussed herein offer an opposite yet complementary approach to GUI-based activities, where instead of simplifying the user interface to make the complex software more accessible, the simplified Lennard-Jones model of a fluid is implemented in a lightweight highly readable Python code. This approach allows students more opportunities to interact directly with chemical simulation code, offering an experience more representative of how professional computational chemists work. Both the simulation code and the Jupyter notebook containing the guided LJ fluid simulation activity are available as free open-source downloads on GitHub,[13,15] and a video tutorial for the LJ fluid simulation activity is available on YouTube.[14]

ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available at https://pubs.acs.org/doi/10.1021/acs.jchemed.2c00838.

xyz trajectory file of a 3D LJ fluid simulation(XYZ)
2D Lennard-Jones (LJ) fluid simulation (2D_trajectory_from_GIF.mov) (MOV)
D LJ fluid simulation (LJ_3D_periodic_boundaries.mov) (MOV)
PDF of the Jupyter Notebook used in the laboratory (PDF)

AUTHOR INFORMATION

Corresponding Author

Gianmarc Grazioli − *Department of Chemistry, San Jose State University, San Jose, California 95192, United States;* orcid.org/0000-0003-2559-5103; Email: gianmarc.grazioli@sjsu.edu

Authors

Adam Ingwerson − *Department of Chemistry, San Jose State University, San Jose, California 95192, United States*
David Santiago, Jr. − *Department of Chemistry, San Jose State University, San Jose, California 95192, United States;* orcid.org/0000-0002-3088-3444
Patrick Regan − *Department of Chemistry, San Jose State University, San Jose, California 95192, United States*
Heekun Cho − *Department of Chemistry, San Jose State University, San Jose, California 95192, United States*

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jchemed.2c00838

Author Contributions

GG wrote the Lennard-Jones fluid simulation code, designed and wrote the Jupyter Notebook lab activity, designed the code dissection activity, created all instructional and assessment materials for both activities, and contributed to writing the manuscript. AI, DS, HC, and PR carried out key beta testing of both instructional activities, provided feedback that was used to improve the simulation code and activities, and contributed to writing the manuscript.

Notes

The authors declare no competing financial interest.
The Python code and Jupyter Notebook needed to complete both of the activities presented herein are available for download at https://github.com/ggrazioli/LennardJonesFluidLab. Also available at that same GitHub link is a Google Colab version of the Jupyter Notebook, which utilizes Google's computing resources to run the simulation, enabling students to carry out the activity on devices with less available RAM. Finally, a video walkthrough of the entire LJ fluid simulation lab, which can be used by instructors for assigning the lab activity as an asynchronous online assignment, is available on YouTube at https://youtu.be/WgyuJYh1VaA.

ACKNOWLEDGMENTS

GG gratefully acknowledges helpful conversations with Prof. Nicholas Esker, as the LJ fluid simulation laboratory activity was assigned to students for the first time in his section of an undergraduate physical chemistry lab course at San José State University. This work was supported by San José State University and a California State University Program for Education and Research in Biotechnology (CSUPERB) award.

REFERENCES

(1) Sabe, V. T.; Ntombela, T.; Jhamba, L. A.; Maguire, G. E.; Govender, T.; Naicker, T.; Kruger, H. G. Current trends in computer aided drug design and a highlight of drugs discovered via

computational techniques: A review. *Eur. J. Med. Chem.* **2021**, *224*, 113705.

(2) Fish, J.; Wagner, G. J.; Keten, S. Mesoscopic and multiscale modelling in materials. *Nature materials* **2021**, *20*, 774−786.

(3) Menke, E. J. Series of Jupyter Notebooks Using Python for an Analytical Chemistry Course. *J. Chem. Educ.* **2020**, *97*, 3899−3903.

(4) Programming Languages Most in Demand Right Now. https://www.linkedin.com/pulse/programming-languages-most-demand-right-now-michael-spencer-?trk=articles_directory (accessed 2022−08−14).

(5) Wofford, M. F.; Boscoe, B. M.; Borgman, C. L.; Pasquetto, I. V.; Golshan, M. S. Jupyter notebooks as discovery mechanisms for open science: Citation practices in the astronomy community. *Computing in Science & Engineering* **2020**, *22*, 5−15.

(6) Abbott, B. P.; Abbott, R.; Abbott, T.; Abernathy, M.; Acernese, F.; Ackley, K.; Adams, C.; Adams, T.; Addesso, P.; Adhikari, R.; et al. Observation of gravitational waves from a binary black hole merger. *Physical review letters* **2016**, *116*, 061102.

(7) Weiss, C. J. A creative commons textbook for teaching scientific computing to chemistry students with python and Jupyter notebooks. *J. Chem. Educ.* **2021**, *98*, 489−494.

(8) Perri, M. Online Data generation in quantitative analysis: excel spreadsheets and an online HPLC simulator using a jupyter notebook on the chem compute web site. *J. Chem. Educ.* **2020**, *97*, 2950−2954.

(9) Lafuente, D.; Cohen, B.; Fiorini, G.; García, A.; Bringas, M.; Morzan, E.; Onna, D. Introduction to Machine Learning for chemists: An undergraduate course using Python notebooks for visualization, data processing, data analysis, and data modeling. *ChemRxiv*. 2021 *Preprint*, Ver. 1, https://chemrxiv.org/engage/chemrxiv/article-details/60c754d9337d6c22b0e28aec (accessed December 3, 2022).

(10) Srnec, M. N.; Upadhyay, S.; Madura, J. D. A Python Program for Solving Schrödinger's Equation in Undergraduate Physical Chemistry. *J. Chem. Educ.* **2017**, *94*, 813−815.

(11) Dahlgren, B. ChemPy: A package useful for chemistry written in Python. *Journal of Open Source Software* **2018**, *3*, 565.

(12) Weiss, C. J. Scientific computing for chemists: An undergraduate course in simulations, data processing, and visualization. *J. Chem. Educ.* **2017**, *94*, 592−597.

(13) Lennard-Jones Fluid Lab. https://github.com/ggrazioli/LennardJonesFluidLab (accessed 2022−08−14).

(14) Lennard-Jones Fluid Simulation Lab. https://youtu.be/WgyuJYh1VaA (accessed 2022−08−15).

(15) Grazioli, G. ggrazioli/LennardJonesFluidLab: Initial Release. 2022; DOI: 10.5281/zenodo.7242966 (accessed 2022−08−14).

(16) Jones, J. E. On the determination of molecular fields.—I. From the variation of the viscosity of a gas with temperature. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* **1924**, *106*, 441−462.

(17) Jones, J. E. On the determination of molecular fields.—II. From the equation of state of a gas. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* **1924**, *106*, 463−477.

(18) Stephan, S.; Thol, M.; Vrabec, J.; Hasse, H. Thermophysical properties of the Lennard-Jones fluid: Database and data assessment. *J. Chem. Inf. Model.* **2019**, *59*, 4248−4265.

(19) Wang, X.; Ramírez-Hinestrosa, S.; Dobnikar, J.; Frenkel, D. The Lennard-Jones potential: when (not) to use it. *Phys. Chem. Chem. Phys.* **2020**, *22*, 10624−10633.

(20) Souza, P. C.; Alessandri, R.; Barnoud, J.; Thallmair, S.; Faustino, I.; Grünewald, F.; Patmanidis, I.; Abdizadeh, H.; Bruininks, B. M.; Wassenaar, T. A.; et al. Martini 3: a general purpose force field for coarse-grained molecular dynamics. *Nat. Methods* **2021**, *18*, 382−388.

(21) Grazioli, G.; Yu, Y.; Unhelkar, M. H.; Martin, R. W.; Butts, C. T. Network-based classification and modeling of amyloid fibrils. *J. Phys. Chem. B* **2019**, *123*, 5452−5462.

(22) Roy, M.; Grazioli, G.; Andricioaei, I. Rate turnover in mechano-catalytic coupling: A model and its microscopic origin. *J. Chem. Phys.* **2015**, *143*, 045105.

(23) Duong, V. T.; Diessner, E. M.; Grazioli, G.; Martin, R. W.; Butts, C. T. Neural Upscaling from Residue-Level Protein Structure Networks to Atomistic Structures. *Biomolecules* **2021**, *11*, 1788.

(24) Wang, J.; Wolf, R. M.; Caldwell, J. W.; Kollman, P. A.; Case, D. A. Development and testing of a general amber force field. *Journal of computational chemistry* **2004**, *25*, 1157−1174.

(25) Vanommeslaeghe, K.; Hatcher, E.; Acharya, C.; Kundu, S.; Zhong, S.; Shim, J.; Darian, E.; Guvench, O.; Lopes, P.; Vorobyov, I.; et al. CHARMM general force field: A force field for drug-like molecules compatible with the CHARMM all-atom additive biological force fields. *Journal of computational chemistry* **2009**, *31*, 671−690.

(26) Domínguez, J.; Alonso, M.; González, E.; Guijarro, M.; Miranda, R.; Oliet, M.; Rigual, V.; Toledo, J.; Villar-Chavero, M.; Yustos, P. Teaching chemical engineering using Jupyter notebook: Problem generators and lecturing tools. *Education for Chemical Engineers* **2021**, *37*, 1−10.

(27) Barba, L. A. Engineers Code: reusable open learning modules for engineering computations. *Computing in Science & Engineering* **2020**, *22*, 26−35.

(28) Humphrey, W.; Dalke, A.; Schulten, K. VMD - Visual Molecular Dynamics. *J. Mol. Graphics* **1996**, *14*, 33−38.

(29) How to Install Anaconda—CHEM-101 at San José State University. https://youtu.be/Y-9Hlrq1kt8 (accessed 2022−08−15).

(30) Jones, T. N.; Graham, K. J.; Schaller, C. P. A jigsaw classroom activity for learning IR analysis in organic chemistry. *J. Chem. Educ.* **2012**, *89*, 1293−1294.

(31) Perkins, D. V.; Saris, R. N. A" jigsaw classroom" technique for undergraduate statistics courses. *Teaching of psychology* **2001**, *28*, 111−113.

(32) Aronson, E. The jigsaw route to learning and liking. *Psychology Today* **1975**, 43−59.

(33) Krathwohl, D. R. A revision of Bloom's taxonomy: An overview. *Theory into practice* **2002**, *41*, 212−218.

(34) Johnson, D. W.; Maruyama, G.; Johnson, R.; Nelson, D.; Skon, L. Effects of cooperative, competitive, and individualistic goal structures on achievement: A meta-analysis. *Psychological bulletin* **1981**, *89*, 47−62.

(35) Carroll, D. W. Use of the jigsaw technique in laboratory and discussion classes. *Teaching of Psychology* **1986**, *13*, 208−210.