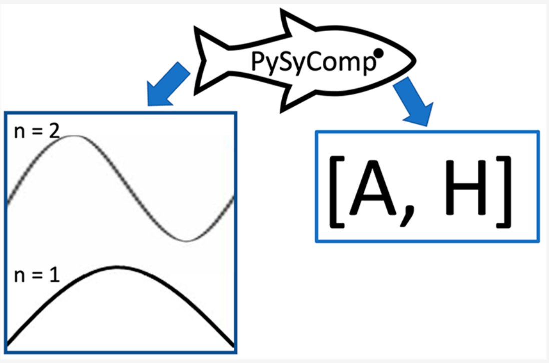
PySyComp: A Symbolic Python Library for the Undergraduate Quantum Chemistry Course

Elizabeth Stippell, Alexey V. Akimov,* and Oleg V. Prezhdo*

Cite This: J. Chem. Educ. 2023, 100, 4077−4084 Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

ABSTRACT: We report an educational tool for the upper level undergraduate quantum chemistry or quantum physics course that uses a symbolic approach via the PySyComp Python library. The tool covers both time-independent and time-dependent quantum chemistry, with the latter rarely considered in the foundations course due to topic complexity. We use quantized Hamiltonian dynamics (QHD) that provides a simple extension of classical dynamics and captures key quantum effects. The PySyComp library can compute various concepts regarding the fundamental postulates of quantum mechanics, including normalized wave functions, expectation values, and commutators, which are at the core of solving the Heisenberg equations of motion. It provides a tool for students to experiment with simple models and explore the key quantum concepts, such as zero-point energy, tunneling, and decoherence.



KEYWORDS: *Upper-Division Undergraduate, Graduate, Computer-Based Learning, Quantum Chemistry*

INTRODUCTION

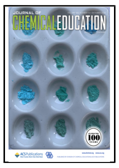
The rapid progress in computer science in the last 30 years has stimulated the creation of the computer algebra systems (CASs), which have been used for educational purposes in various fields of study, including chemistry.[1−11] Tools such as Wolfram Mathematica,[12] Mathcad,[13] Maple,[14] and Matlab[15] have been used in classroom environments as valuable teaching tools across a variety of subject areas.[2−9,16−20] The extensive databases of demonstration codes supported by these tools have been reported for quantum mechanics and physical chemistry education. Among them is the Wolfram Mathematica's Demonstrations Project,[9,21] containing simple demonstrations on topics standard in quantum mechanics course curricula, such as quantum harmonic oscillators, spherical harmonics, potential wells, and more.[21] More recently, a wide adoption and great success of the Python programming language has stimulated the development of Python-based teaching tools in a diverse array of chemical disciplines, such as in the general analytical chemistry course, nuclear magnetic resonance visualization, automated titrations, and gas chromatography.[22−25] Python has been used in various computational and quantum chemistry teaching settings as well, e.g., for solving Schrödinger's equation, understanding Boltzmann distributions and reciprocal space.[26−29]

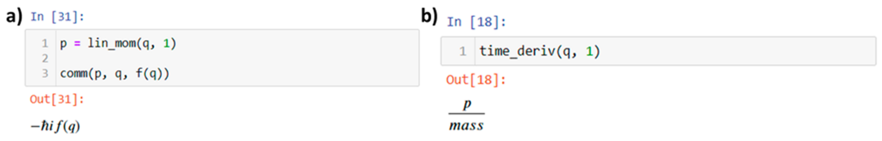
Among the flexibility and intuitive syntax, Python-based educational systems and tools can take advantage of multiple Python libraries and resources for efficient numeric calculations,[30,31] visualization,[32−34] data analysis and processing,[35] data storage,[36] and domain-centric objectives,[37] most being charge-free and often open-source. These are the criteria that may be important for smaller institutions and schools that are unable to afford licenses for commercial CASs.

Python is an optimal programming language to use due to extensive resources and initiatives on introducing Python to undergraduate students, such as the compute-to-learn project designed at the University of Michigan,[9] the Molecular Sciences Software Institute (MolSSI) that provides software experiences and workshops to quantum chemistry students,[38] and pythoninchemistry[39] that provides chemists with tutorials and applications of scientific computing in Python. Python is also an optimal choice due to its relevance in higher level computational research.[26,37,40−43]

Python can be run through three main interfaces: via a Python interpreter, the IPython interpreter, or self-contained Python scripts.[44] Self-contained Python scripts are .py files that allow users to write and run multiple lines, where the former two methods are run line by line. The Jupyter Notebook format is a combination of the self-contained script and the interactive terminal[44] and is a powerful environment for developing code in Python, making it a good platform for both developing code and creating interactive examples and tutorials

Received: October 4, 2022
Revised: August 29, 2023
Published: September 19, 2023





a)

```
In [31]:
p = lin_mom(q, 1)

comm(p, q, f(q))
Out[31]:
```

$-\hbar i f(q)$

b)

```
In [18]:
time_deriv(q, 1)
Out[18]:
```

$\frac{p}{mass}$

Figure 1. Example outputs from the PySyComp library. (a) A simple commutator calculation and (b) a QHD calculation for position to the first order.

for users who may not have a background in coding or programming languages.

The code presented was written using Python 3 through Jupyter Notebook 5.7.4. Python is a programming language that is popular due to its ease, simplicity of use, and its efficient runtime and feedback loop, making it ideal for undergraduate use.[45,46] Jupyter Notebook is a free and open source web app that is compatible with multiple programming languages.[47,48] Any code or notebooks discussed throughout this paper can be found in the Supporting Information Zip 1 or Zip 2 or on GitHub, which has the latest code and developments.

PySyComp provides a modernized way of computing in quantum mechanics and uses SymPy as a back end for symbolically solving quantum mechanical problems. SymPy,[49] or Symbolic Python, is a Python library aimed at evaluating mathematical expressions in a symbolic way rather than numerically. It has extensive functions that can compute derivatives and integrals and also has defined functions for several quantum topics,[50−52] some of which were used as a foundation for the PySyComp library, such as SymPy's derivations, integrations, and basic commutator function. We specifically chose to use SymPy because of its simplicity and ease of use for students who do not have prior experience with Python.

In this work, we report the development and application of the PySyComp library in two different examples. The first example is a simple particle in a box tutorial that discusses normalizing wave functions and expectation values and plotting various wave functions. The second example employs the quantized Hamiltonian dynamics (QHD) method to compute the equations of motion using natural variables as described by Akimov and Prezhdo.[53] QHD is a simple extension of classical Hamiltonian dynamics and captures essential quantum effects in a computationally efficient manner. The theory and calculations of the QHD equations of motion are described in the overview of the QHD theory section. The QHD method is applied to three different potentials: the Morse potential, Gaussian potential, and double well potential. The potentials are separated by modules, and each module is interactive for student use via Jupyter Notebooks. Only a beginner's knowledge of Python is necessary to operate the PySyComp code. When the learning modules are implemented in a classroom setting, one class period would suffice to teach and apply the code to various examples as outlined in the modules.

■ AN OVERVIEW OF QHD THEORY

The QHD approach is a semiclassical method based on the Heisenberg equations of motion:

$$i\hbar\frac{\mathrm{d}\langle\hat{A}\rangle}{\mathrm{d}t} = \langle[\hat{A}, \hat{H}]\rangle \tag{1}$$

where $\hat{A}$ represents an operator of some observable and $\hat{H}$ represents the Hamiltonian, given by

$$\hat{H} = \frac{\hat{p}^2}{2m} + V(\hat{q}) \tag{2}$$

Here, the first term represents the kinetic energy, and the second term represents the potential energy of a given system. Applying the Heisenberg equation of motion to position, $\hat{q}$, and momentum, $\hat{p}$, and operators along with their higher powers yields the QHD equations of motion. Approximations are made by closures of a hierarchy of QHD equations. The method was introduced by Pereverzev and Prezhdo[54,55] and extended by Akimov and Prezhdo.[53] Working with the Heisenberg equations of motion for quantum mechanical expectation values, one can obtain the QHD equation hierarchy, as described below to the second order:

$$\frac{\mathrm{d}\langle q\rangle}{\mathrm{d}t} = \frac{\langle p\rangle}{m} \tag{3}$$

$$\frac{\mathrm{d}\langle p\rangle}{\mathrm{d}t} = -\langle V'\rangle \tag{4}$$

$$\frac{\mathrm{d}\langle q^2\rangle}{\mathrm{d}t} = 2\frac{\langle(pq)_s\rangle}{m} \tag{5}$$

$$\frac{\mathrm{d}\langle(pq)_s\rangle}{\mathrm{d}t} = \left(\frac{p^2}{m} - \langle(qV')_s\rangle\right) \tag{6}$$

$$\frac{\mathrm{d}\langle p^2\rangle}{\mathrm{d}t} = -2\langle(pV')_s\rangle \tag{7}$$

The use of the symmetrized form $(AB)_s = \frac{AB + BA}{2}$ allows us to obtain the equations of motion with real variables only. Higher order variables are decomposed into products of the first and second order variables, as described in ref 53.

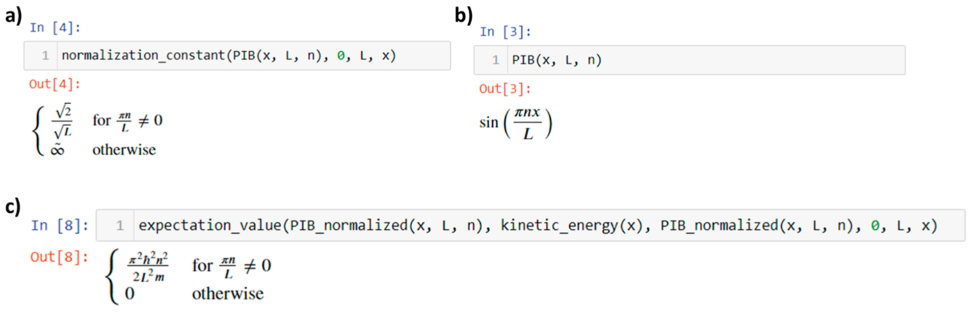
In this work, three different potentials are considered. The Morse, Gaussian, and double well potentials are all used to show the various benefits of using the QHD approach versus classical dynamics.

Description of the PySyComp Code

PySyComp serves as an introduction to various theories and methods that students may encounter in a quantum mechanics classroom and is sufficiently simple that even students with no prior coding experiences can benefit from its use. Symbolic notation is at the core of the PySyComp library in both the particle in a box and the QHD examples. Specifically, in the QHD example, a function "time_deriv" was created to derive the equations of motion symbolically, by applying eq 1. As the QHD method is derived from the Heisenberg equations of motion, which utilizes commutators, commutators were a focus of the PySyComp code. A commutator is defined as

$$[A, B] = A \times B - B \times A \tag{8}$$

Examples of commutators can be found in module 1 (Supporting Information Zip 2).



a)

In [4]:
```
normalization_constant(PIB(x, L, n), 0, L, x)
```
Out[4]:

$$\begin{cases} \frac{\sqrt{2}}{\sqrt{L}} & \text{for } \frac{\pi n}{L} \neq 0 \\ \tilde{\infty} & \text{otherwise} \end{cases}$$

b)

In [3]:
```
PIB(x, L, n)
```
Out[3]:

$$\sin\left(\frac{\pi n x}{L}\right)$$

c)

In [8]:
```
expectation_value(PIB_normalized(x, L, n), kinetic_energy(x), PIB_normalized(x, L, n), 0, L, x)
```
Out[8]:

$$\begin{cases} \frac{\pi^2 \hbar^2 n^2}{2L^2 m} & \text{for } \frac{\pi n}{L} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Figure 2. (a) The normalization constant for a one-dimensional particle in a box. (b) The normalized wave function for a one-dimensional particle in a box. (c) The kinetic energy expectation value for a one-dimensional particle in a box.

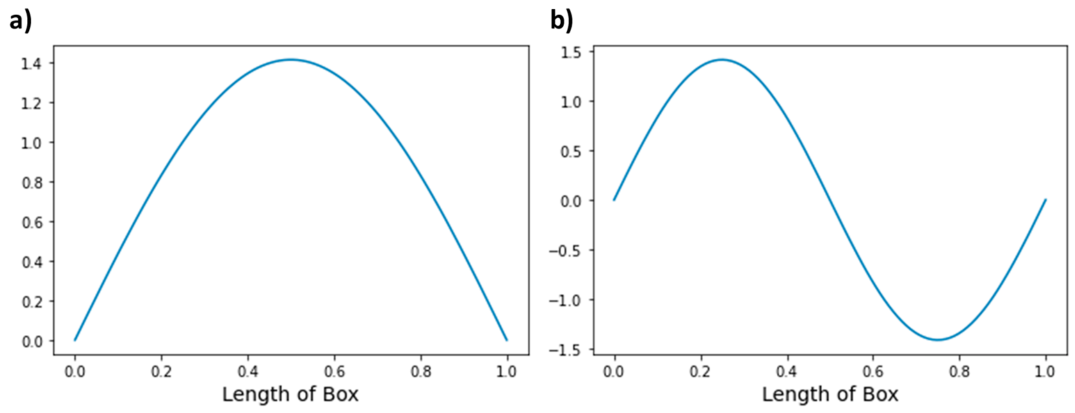


Figure 3. (a) Graph of a one-dimensional ground state particle in a box. (b) Graph of a one-dimensional excited state (n = 2) particle in a box.

The PySyComp library provides a new way to compute values without the need for pen-and-paper derivations. PySyComp can compute similar commutators as described in module 1 (Supporting Information Zip 2) with ease, including commutators with higher exponents, as shown in the following QHD example. PySyComp is also able to compute normalization constants and expectation values in a simple, easy to understand manner (Figure 1).

Within the Supporting Information, there are also further examples of solving similar computations. Without PySyComp, these calculations would require multiple steps and more knowledge of Python to yield the same or similar results. PySyComp offers various predefined wave functions, such as those used in the particle in a box example for both one and three dimensions. A complete list of functions found in PySyComp is shown in Table S1.

RESULTS

Particle in a Box

Due to the fundamental quantum mechanical principles encountered in a particle in a box problem, there is documentation as well as a tutorial for the particle in a box problem specifically. Within the particle in a box tutorial, there is information about both a one-dimensional and two-dimensional particle in a box.

In the particle in a box module, students are asked to calculate various properties related to a one-dimensional particle in a box, including the normalization constant and expectation values for both position and kinetic energy, as shown in Figure 2. Students can also plot both the 1D and 2D wave functions for a particle in a box using a combination of PySyComp and MatPlotLib codes, as shown in Figure 3.

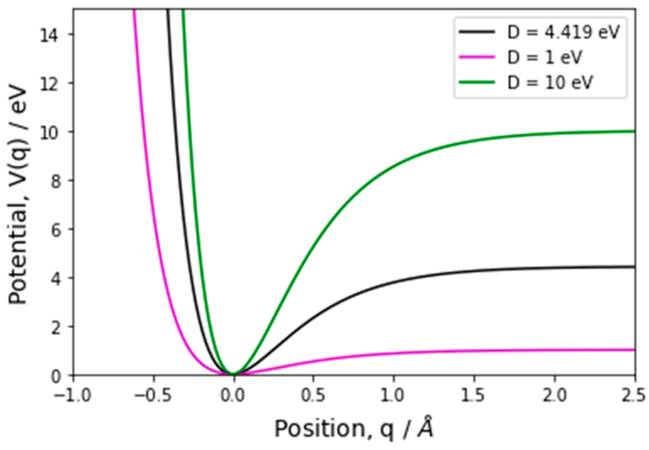


Figure 4. Three 1D Morse potentials with D = 4.419 eV (black), D = 1 eV (magenta), D = 10 eV (green), α = 2.56 Å^{-1}, and q_0 = 0.15 Å. The mass of the particle is 1836 au, which reflects the reduced mass of an O−H vibration in a water molecule. Lower D values correspond to shallower and more anharmonic potentials.

The plotted wave function peaks correspond to the most likely location to find the particle in the box. For example, in Figure 3a, the peak is at the center of the box, corresponding to the position expectation value of $L/2$.

Similar to the one-dimensional plots, the two-dimensional plots correspond to the position expectation values and indicate where one is most likely to find the particle in a box. Only values of n = 1 and 2 are included in this work, although any higher values can also be computed via PySyComp for both one-dimensional and two-dimensional

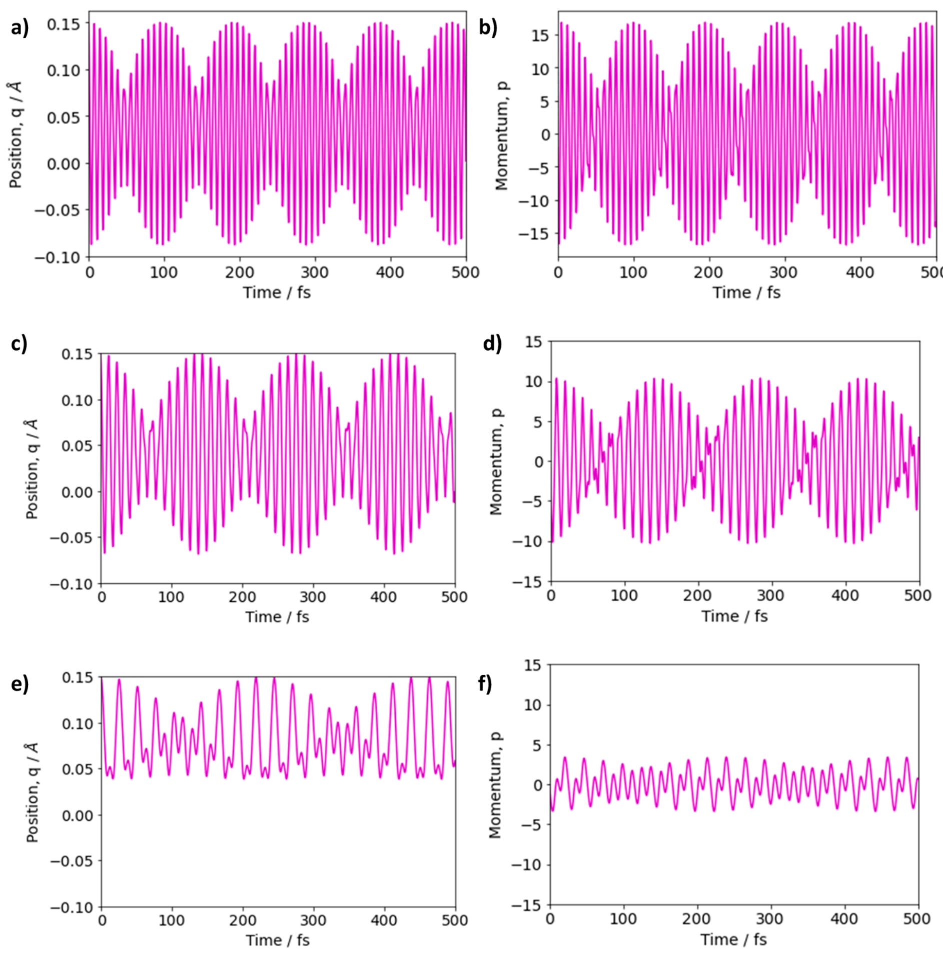


Figure 5. Position versus time for the Morse potential of an O−H vibration in a water molecule with the parameters (a) D = 10 eV, (c) D = 4.419 eV, and (e) D = 1 eV. Momentum versus time for the Morse potential with the parameters (b) D = 10 eV, (d) D = 4.419 eV, and (f) D = 1 eV. Anharmonicity increases with decreasing D, and the dynamics start to exhibit additional frequencies.

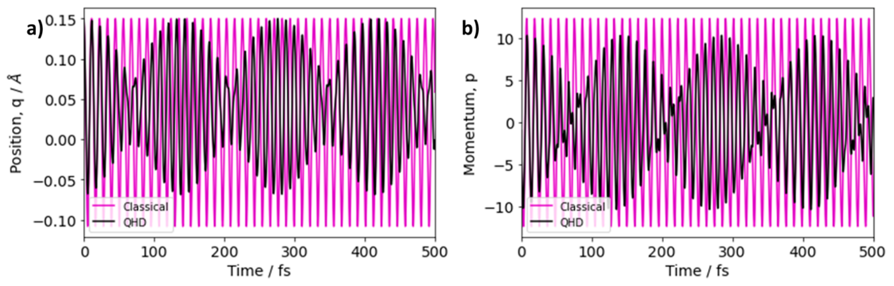


Figure 6. Comparison of the QHD method (black) and classical (magenta) equations of motion for (a) position vs time and (b) momentum vs time for the Morse potential with D = 4.417 eV, q_0 = 0.15. The QHD dynamics is more complex, showing both high and low frequencies due to anharmonicity, while the classical dynamics only shows high frequency.

boxes. The graphs for the two-dimensional particle in a box can be found in the Supporting Information.

Quantized Hamiltonian Dynamics

There are three modules dedicated to understanding the QHD method. Three different potentials are shown: the Morse potential, the Gaussian potential, and the double well potential. All three potentials are discussed in the following sections. The initial conditions for all examples are as follows: p_0 = 0.0 (mean momentum), s_0 = 0.05 (mean width), and ps_0 = 0.0 (mean width momentum), corresponding to a localized Gaussian wave function. A time step of 0.1 fs was used for a total of 1000 fs for each of the potentials, although smaller snapshots are shown for clarity.

Morse Potential. In module 3 (Supporting Information Zip 2), users can observe anharmonicity effects created from the QHD approach and compare them against the classical equations of motion. The potential energy of a Morse potential is given by the following equation:

$$V(q) = D \times (1 - e^{-\alpha(q-q_0)})^2 \qquad (9)$$

where D is a variable users can change, α is a constant provided for the O−H stretch in a water molecule, and q_0 is the initial

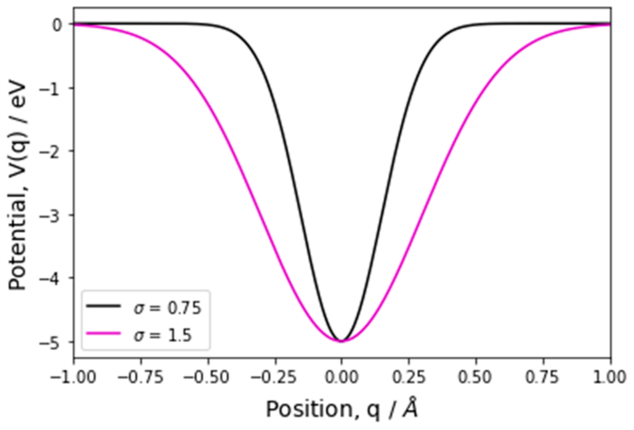


Figure 7. Two inverted Gaussian potentials with parameters V_0 = 5 eV and σ = 1.5 Å (magenta) or σ = 0.75 Å (black). σ describes the width of the Gaussian.

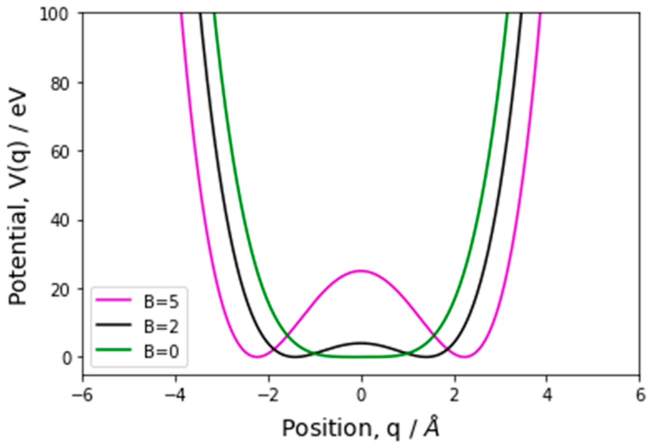


Figure 9. Double well potential with B = 5, 2, and 0. As the level of B decreases, the barrier in the middle of the well disappears.

position. For the calculation of the equations of motion, this value is equal to zero. Throughout the module, there are built-in questions and prompts for users to answer, with many of the answers being found in this work. By comparing three different D values, users can understand the deviation from classical trajectories and gain a better understanding of the anharmonicity that arises when an approximation method is used. It also shows that the QHD method is a more accurate representation of dynamics in the Morse potential because it accounts for this anharmonicity.

Users are first able to plot the potentials themselves, as shown in Figure 4, and then further plot the position versus time and momentum versus time graphs for a given time interval, as in Figure 5. For the Morse potential, three different values of D were used for calculations: D = 10 eV, D = 4.419 eV, and D = 1 eV, with D = 4.419 eV corresponding to an O—H vibration in a water molecule.

When one compares the three examples shown above, as the D parameter decreases, the period increases, producing a slower movement for systems of the same mass. This correlates with a broadening of the Morse potential and an increase in the “shallowness” of the potential. There is also an increase in the anharmonicity as the depth of the well decreases and a decrease in the frequency of the position over time. The dynamics becomes more complex as anharmonicity increases:

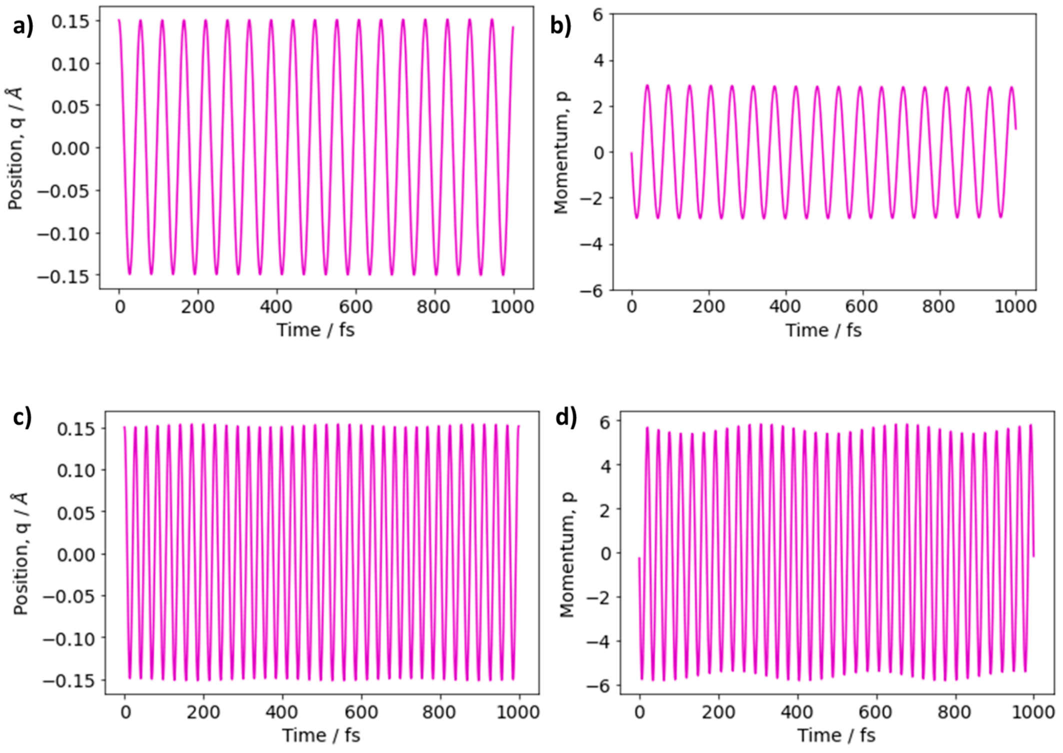


Figure 8. Position vs time for the Gaussian potential with (a) σ = 1.5 Å and (c) σ = 0.75 Å. Momentum versus time for the Gaussian potential with (b) σ = 1.5 Å and (d) σ = 0.75 Å. As the Gaussian width, σ, decreases, the oscillation frequency increases, and the anharmonicity becomes more pronounced, because the energy of the quantum particle, including zero-point energy, grows.

a new pronounced frequency appears for $D = 1$ eV, compared with the larger values of D. A similar trend is shown regarding the momentum values as was seen in the position values. Note the asymmetry in the functions, which was not present in the position versus time examples.

Users should be able to consider the three different D parameters and understand the changes in the position and momentum values over time, qualitatively. Users may also try their own D values to further understand the anharmonicity and frequency effects. Users are further encouraged to plot p^2 versus time and examine their relationships to varying D values.

In module 3 (Supporting Information Zip 2), users may also compare the position and momentum versus time for the classical equations of motion versus the QHD equations of motion, as shown in Figure 6. When plotted along with the QHD position and momentum (as shown in Figure 3), clear differences arise, showing anharmonicities and additional frequencies that are not taken into account in the classical equations of motion.

Gaussian Potential. In module 4 (Supporting Information Zip 2), users can solve the QHD equations of motion for the Gaussian potential given by

$$V(q) = V_0 \times e^{(-\alpha \times q^2)} \tag{10}$$

where

$$\alpha = \frac{1}{2 \times \sigma^2} \tag{11}$$

Two Gaussian potentials are included in this module, as shown in Figure 7. By an increase in the σ value, the Gaussian potential broadens. Users are able to change the σ value, thus changing the Gaussian potential.

As the σ value decreases, corresponding to a narrower Gaussian potential, the main oscillation frequency increases, as shown in Figure 7. The graphs show slight deviations from a harmonic system, as manifested in an additional slower oscillation frequency. The anharmonicity becomes more pronounced as the σ value is decreased from 1.5 to 0.75 because of the increased total energy of the quantum mechanical particle, which includes zero-point energy.

Users can also analyze why the momentum vs time graph shows more deviations from the harmonic motion than the position vs momentum graph by comparing the classical and QHD equations of motion as seen in Figure 8. In particular, the QHD equation for the position, eq 3, coincides with the corresponding classical equation, while the QHD equation for the momentum, eq 4, contains the expectation value of the potential energy. The quantum expectation value contains zero-point energy that is absent in classical mechanics.

Double Well Potential. In module 5 (Supporting Information Zip 2), users are introduced to the double well potential, as described by

$$V(q) = (q^2 - B)^2 \tag{12}$$

The B constant controls the height of the barrier in the double-well, and $B = 0$ indicates a single well potential. In this example, two different starting positions were used along with three different values of B.

Two double well potentials and a single well potential are shown in this module, as shown in Figure 9. In the double well potentials, two different starting points are used to show movement between the wells and movement in just one of the wells.

The figures for this example can be found in the Supporting Information. The particle stays in the right well when $q_0 = 0.15$ Å and moves freely between the two wells when $q_0 = -2.5$ Å. Users should be able to understand the importance of the starting position in module 5 (Supporting Information Zip 2), as it will affect both the position and the momentum over time. This is due to the quadratic nature of the potential. In this module, users are able to change the height of the center barrier in the well and can compare $B = 5$ (in which the height of the inner barrier is large), $B = 2$ (in which the inner barrier is low), and $B = 0$ (in which the double well becomes a single well) or choose any value of B. Users are also encouraged to change the starting position and compare the results to Figure S3.

■ CONCLUSIONS

By working through the modules, users will develop a hands-on understanding of basic time-independent and time-dependent quantum mechanics and will have a tool to experiment with simple models to explore the key quantum concepts, including zero-point energy, tunneling, and decoherence. The PySyComp library is an effective teaching tool with various capabilities, such as computing values associated with both one- and two-dimensional particles in a box. The PySyComp library is also equipped to handle solving for equations of motion, as seen in the QHD example and application. The QHD method is a simple semiclassical method to introduce users to. At a minimal complexity above classical mechanics, QHD captures zero-point energy, tunneling, decoherence, and other key phenomena in time-dependent quantum mechanics.

■ ASSOCIATED CONTENT

Data Availability Statement

All associated material can be found on GitHub at https://github.com/liz-stippell/pysycomp. The tutorials described in this work can be found in the "Tutorials" folder. The Python code and Jupyter notebooks used in this work can also be downloaded from the Supporting Information Zip 1or Zip 2.

Supporting Information

The Supporting Information is available at https://pubs.acs.org/doi/10.1021/acs.jchemed.2c00974.

Table of available functions in the PySyComp library and the QHD library as well as figures of the two-dimensional particle in a box wave function and position vs time and momentum vs time graphs for the double-well potential using the QHD method (PDF)

Source code in order to run PySyComp (ZIP)

Downloadable Jupyter Notebooks: Modules 1 to 5 (ZIP)

■ AUTHOR INFORMATION

Corresponding Authors

Alexey V. Akimov − *Department of Chemistry, University at Buffalo, The State University of New York, Buffalo, New York 14260, United States*; orcid.org/0000-0002-7815-3731; Email: alexeyak@buffalo.edu

Oleg V. Prezhdo − *Department of Chemistry, University of Southern California, Los Angeles, California 90089, United*

States; orcid.org/0000-0002-5140-7500;
Email: prezhdo@usc.edu

Author

Elizabeth Stippell − *Department of Chemistry, University of Southern California, Los Angeles, California 90089, United States*

Complete contact information is available at:
https://pubs.acs.org/10.1021/acs.jchemed.2c00974

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

O.V.P. acknowledges support from the US National Science Foundation, grant CHE-2154367. A.V.A. acknowledges financial support from the National Science Foundation, grant NSF-2045204.

■ REFERENCES

(1) Binous, H. Liquid-Liquid Equilibrium and Extraction Using Mathematica. *Comput. Educ. J.* **2006**, *16* (1), 78−81.
(2) Smith, W. R.; Missen, R. W. Using Mathematica and Maple To Obtain Chemical Equations. *J. Chem. Educ.* **1997**, *74* (11), 1369−1371.
(3) Hansen, J. C. Schröedinger.m: A Mathematica Package for Solving the Time-Independent Schrödinger Equation. *J. Chem. Educ.* **1996**, *73* (10), 924.
(4) Lang, P. L.; Towns, M. H. Visualization of Wavefunctions Using Mathematica. *J. Chem. Educ.* **1998**, *75* (4), 506−509.
(5) Sokolik, C. W. A Maple Program That Illustrates the Effect of PH on Peptide Charge. *J. Chem. Educ.* **1998**, *75* (11), 1500−1502.
(6) Zdravkovski, Z. Mathcad in Chemistry Calculations II: Arrays. *J. Chem. Educ.* **1992**, *69* (9), A242−A244.
(7) Zielinski, T. J. Mathcad in the Chemistry Curriculum. *J. Chem. Educ.* **1998**, *75* (9), 1189−1190.
(8) Abbas, A.; Al-Bastaki, N. The Use of Software Tools for ChE Education: Students' Evaluations. *Chem. Eng. Educ.* **2002**, *36* (3), 236−241.
(9) Jafari, M.; Welden, A. R.; Williams, K. L.; Winograd, B.; Mulvihill, E.; Hendrickson, H. P.; Lenard, M.; Gottfried, A.; Geva, E. Compute-to-Learn: Authentic Learning via Development of Interactive Computer Demonstrations within a Peer-Led Studio Environment. *J. Chem. Educ.* **2017**, *94* (12), 1896−1903.
(10) Potratz, J. P. Making Enzyme Kinetics Dynamic via Simulation Software. *J. Chem. Educ.* **2018**, *95* (3), 482−486.
(11) Fisher, A. A. E. An Introduction to Coding with Matlab: Simulation of X-Ray Photoelectron Spectroscopy by Employing Slater's Rules. *J. Chem. Educ.* **2019**, *96* (7), 1502−1505.
(12) *Wolfram Mathematica: Modern Technical Computing*; https://www.wolfram.com/mathematica/ (accessed 2020−12−27).
(13) Mathcad. *Mathcad: Math software for engineering calculations*; https://www.mathcad.com (accessed 2020−12−27).
(14) Maplesoft. *Software for Mathematics, Online Learning, Engineering*; https://www.maplesoft.com/ (accessed 2020−12−27).
(15) MathWorks. *MATLAB*; https://www.mathworks.com/products/matlab.html (accessed 2020−12−27).
(16) Gorgol, I. WOLFRAM DEMONSTRATIONS PROJECT PLATFORM AS A SUPPORT IN TEACHING. *Adv. Sci. Technol. Res. J.* **2015**, *9* (26), 143−147.
(17) Maclachlan, F.; Bolte, W. J.; Chandler, S. Interactive Economic Models from the Wolfram Demonstrations Project. *J. Econ. Educ.* **2009**, *40* (1), 108−108.
(18) Etcuban, J. O.; Campanilla, B. S.; Horteza, A. D. The Use of Mathcad in the Achievement of Education Students in Teaching College Algebra in a University. *Int. Electron. J. Math. Educ.* **2019**, *14* (2), 341−351.
(19) Parulekar, S. J. Numerical Problem Solving Using Mathcad in Undergraduate Reaction Engineering. *Chem. Eng. Educ.* **2006**, *40* (1), 14−23.
(20) Karagiannis, P.; Markelis, I.; Paparrizos, K.; Samaras, N.; Sifaleras, A. E-Learning Technologies: Employing Matlab Web Server to Facilitate the Education of Mathematical Programming. *Int. J. Math. Educ. Sci. Technol.* **2006**, *37* (7), 765−782.
(21) *Eitan Geva*; Wolfram Demonstrations Project; https://demonstrations.wolfram.com/author.html?author=Eitan%20Geva&start=1&limit=20&sortmethod=recent (accessed 2020−12−27).
(22) Menke, E. J. Series of Jupyter Notebooks Using Python for an Analytical Chemistry Course. *J. Chem. Educ.* **2020**, *97* (10), 3899−3903.
(23) Arrabal-Campos, F. M.; Cortés-Villena, A.; Fernández, I. Building "My First NMRviewer": A Project Incorporating Coding and Programming Tasks in the Undergraduate Chemistry Curricula. *J. Chem. Educ.* **2017**, *94* (9), 1372−1376.
(24) Tan, S. W. B.; Naraharisetti, P. K.; Chin, S. K.; Lee, L. Y. Simple Visual-Aided Automated Titration Using the Python Programming Language. *J. Chem. Educ.* **2020**, *97* (3), 850−854.
(25) Green, M.; Chen, X. Data Functionalization for Gas Chromatography in Python. *J. Chem. Educ.* **2020**, *97* (4), 1172−1175.
(26) 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* (6), 813−815.
(27) Kurniawan, O.; Koh, L. L. A.; Cheng, J. Z. M.; Pee, M. Helping Students Connect Interdisciplinary Concepts and Skills in Physical Chemistry and Introductory Computing: Solving Schrödinger's Equation for the Hydrogen Atom. *J. Chem. Educ.* **2019**, *96* (10), 2202−2207.
(28) Jameson, G.; Brüschweiler, R. Active Learning Approach for an Intuitive Understanding of the Boltzmann Distribution by Basic Computer Simulations. *J. Chem. Educ.* **2020**, *97* (10), 3910−3913.
(29) Srnec, M. N.; Upadhyay, S.; Madura, J. D. Teaching Reciprocal Space to Undergraduates via Theory and Code Components of an IPython Notebook. *J. Chem. Educ.* **2016**, *93* (12), 2106−2109.
(30) Harris, C. R.; Millman, K. J.; van der Walt, S. J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N. J.; Kern, R.; Picus, M.; Hoyer, S.; van Kerkwijk, M. H.; Brett, M.; Haldane, A.; del Río, J. F.; Wiebe, M.; Peterson, P.; Gérard-Marchant, P.; Sheppard, K.; Reddy, T.; Weckesser, W.; Abbasi, H.; Gohlke, C.; Oliphant, T. E. Array Programming with NumPy. *Nature* **2020**, *585* (7825), 357−362.
(31) Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; van der Walt, S. J.; Brett, M.; Wilson, J.; Millman, K. J.; Mayorov, N.; Nelson, A. R. J.; Jones, E.; Kern, R.; Larson, E.; Carey, C. J.; Polat, İ.; Feng, Y.; Moore, E. W.; VanderPlas, J.; Laxalde, D.; Perktold, J.; Cimrman, R.; Henriksen, I.; Quintero, E. A.; Harris, C. R.; Archibald, A. M.; Ribeiro, A. H.; Pedregosa, F.; van Mulbregt, P. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nat. Methods* **2020**, *17* (3), 261−272.
(32) Rego, N.; Koes, D. 3Dmol.Js: Molecular Visualization with WebGL. *Bioinformatics* **2015**, *31* (8), 1322−1324.
(33) Hunter, J. D. Matplotlib: A 2D Graphics Environment. *Comput. Sci. Eng.* **2007**, *9* (3), 90−95.
(34) *pyglet*; http://pyglet.org/ (accessed 2020−12−31).
(35) McKinney, W. Data Structures for Statistical Computing in Python. *Proc. 9th Python Sci. Conf.* **2010**, 56−61.
(36) *HDF5 for Python*; http://www.h5py.org/ (accessed 2021−01−05).
(37) Smith, D. G. A.; Burns, L. A.; Sirianni, D. A.; Nascimento, D. R.; Kumar, A.; James, A. M.; Schriber, J. B.; Zhang, T.; Zhang, B.; Abbott, A. S.; Berquist, E. J.; Lechner, M. H.; Cunha, L. A.; Heide, A. G.; Waldrop, J. M.; Takeshita, T. Y.; Alenaizan, A.; Neuhauser, D.; King, R. A.; Simmonett, A. C.; Turney, J. M.; Schaefer, H. F.;

Evangelista, F. A.; DePrince, A. E.; Crawford, T. D.; Patkowski, K.; Sherrill, C. D. Psi4NumPy: An Interactive Quantum Chemistry Programming Environment for Reference Implementations and Rapid Development. *J. Chem. Theory Comput.* **2018**, *14* (7), 3504−3511.

(38) *Molecular Science Software Institute*; GitHub; https://github.com/MolSSI (accessed 2022−09−29).

(39) *pythoninchemistry*; https://pythoninchemistry.org (accessed 2020−12−28).

(40) Kohnle, A.; Benfield, C.; Hähner, G.; Paetkau, M. Interactive Simulations To Support Quantum Mechanics Instruction for Chemistry Students. *J. Chem. Educ.* **2017**, *94* (3), 392−397.

(41) Polizzi, N. F.; Beratan, D. N. Open-Access, Interactive Explorations for Teaching and Learning Quantum Dynamics. *J. Chem. Educ.* **2015**, *92* (12), 2161−2164.

(42) Birkeland, T. PyProp - A Python Framework for Propogating the Time Dependent Schrödinger Equation. Ph.D. Dissertation, University of Bergen, September, 2009; https://core.ac.uk/download/pdf/30926372.pdf (accessed 2020−12−09).

(43) Weiss, C. J. Scientific Computing for Chemists: An Undergraduate Course in Simulations, Data Processing, and Visualization. *J. Chem. Educ.* **2017**, *94* (5), 592−597.

(44) VanderPlas, J. *A Whirlwind Tour of Python*; O'Reilly Media, Inc.

(45) *What is Python? Executive Summary*; https://www.python.org/doc/essays/blurb/ (accessed 2020−12−09).

(46) Pérez, F.; Granger, B. E.; Hunter, J. D. Python: An Ecosystem for Scientific Computing. *Comput. Sci. Eng.* **2011**, *13* (2), 13−21.

(47) *Project Jupyter*; https://www.jupyter.org (accessed 2020−12−09).

(48) Kluyver, T.; Ragan-Kelley, B.; Pérez, F.; Bussonnier, M.; Frederic, J.; Hamrick, J.; Grout, J.; Corlay, S.; Ivanov, P.; Abdalla, S.; Willing, C. Jupyter Notebooks—a Publishing Format for Reproducible Computational Workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*; 2016; DOI: 10.3233/978-1-61499-649-1-87.

(49) Meurer, A.; Smith, C. P.; Paprocki, M.; Čertík, O.; Kirpichev, S. B.; Rocklin, M.; Kumar, Am; Ivanov, S.; Moore, J. K.; Singh, S.; Rathnayake, T.; Vig, S.; Granger, B. E.; Muller, R. P.; Bonazzi, F.; Gupta, H.; Vats, S.; Johansson, F.; Pedregosa, F.; Curry, M. J.; Terrel, A. R.; Roučka, Š.; Saboo, A.; Fernando, I.; Kulal, S.; Cimrman, R.; Scopatz, A. SymPy: Symbolic Computing in Python. *PeerJ. Comput. Sci.* **2017**, *3*, No. e103.

(50) Cugini, A.; Curry, M.; Granger, B. Symbolic Quantum Computation Simulation in SymPy. In *2010 Annual Meeting of the California-Nevada Section of the APS*, October 29−30, 2010; abstract id C4.004.

(51) *Quantum Mechanics — SymPy 1.7 Documentation*; https://docs.sympy.org/latest/modules/physics/quantum/index.html (accessed 2020−12−09).

(52) Cugini, A. *Quantum Mechanics, Quantum Computation, and the Density Operator in SymPy*; 2011; http://digitalcommons.calpoly.edu/physsp/38 (accessed 2023−08−29).

(53) Akimov, A. V.; Prezhdo, O. V. Formulation of Quantized Hamiltonian Dynamics in Terms of Natural Variables. *J. Chem. Phys.* **2012**, *137* (22), 224115.

(54) Prezhdo, O. V.; Pereverzev, Yu. V. Quantized Hamilton Dynamics. *J. Chem. Phys.* **2000**, *113* (16), 6557−6565.

(55) Prezhdo, O. V. Quantized Hamilton Dynamics. *Theor. Chem. Acc.* **2006**, *116* (1−3), 206−218.