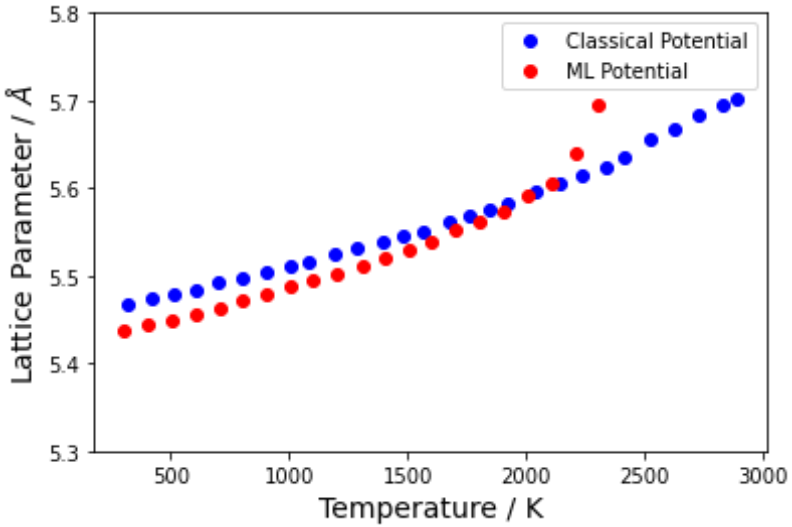
PySyComp: A Symbolic Python Library

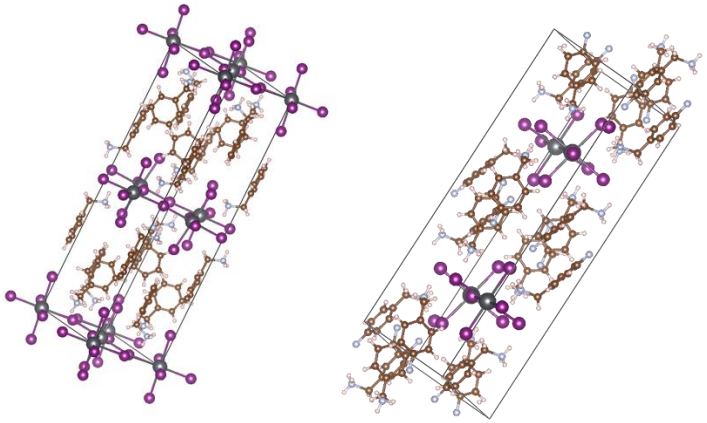
Elizabeth Stippell

Other Projects:

- Work for Los Alamos National Laboratory
  - Comparing properties computed with a Machine Learning Potential (ANI) with properties computed by a classical potential (EAM) for Uranium Oxide Systems



- Work in Collaboration with the Beljonne Group at Umons
  - Using projection operator diabatization on 2D lead halide perovskite with organic spacers to study the charge transfer between layers



Outline: PySyComp

Background & Motivation

→

Commutators

Computing the commutator

`steps` function

→

Quantized Hamiltonian Dynamics

Methods development

Applications

Background & Motivation

- Python is a good introductory programming language for students to gain experience in
- There exists ample resources for students for Python

Background & Motivation

- Symbolic libraries create a simple way for students to learn how to code in a way that is familiar to them

- Objectives:

1. Develop a method for symbolically computing equations of motion
2. Apply equations of motion to various systems

Commutators

$$[\hat{A}, \hat{B}]$$

- An introductory problem to quantum mechanics for undergraduate students
- Foundation for computing the equations of motion for Quantized Hamiltonian Dynamics

Theory: Commutators

$$\left[\hat{A}, \hat{B}\right] = \hat{A} * \hat{B} - \hat{B} * \hat{A}$$

Examples:

$$[\hat{x}, \widehat{p_x}]\Psi = i\hbar\Psi$$

$$\left[\hat{x}, \widehat{p_x^2}\right]\Psi = 2i\hbar p\Psi$$

$$\left[\widehat{p_x^2}, \widehat{x^2}\right]\Psi = 2\hbar^2(-2q\Psi' - \Psi)$$

Code: Commutators

```python
def comm_1(commutator_1, commutator_2, aux):
    """
    This function is not used directly, it is only used in the comm() function below.

    Args:
        commutator_1: The first operator in the commutator
        commutator_2: The second operator in the commutator
        aux: The auxiliary function. This is defined below, as F(x).

    Returns:
        The commutator of commutator_1 and commutator_2 with respect to the auxiliary function.

    """

    return expand(Operator(commutator_1)*Operator(commutator_2)*aux-
Operator(commutator_2)*Operator(commutator_1)*aux)
```

Code: Linear Momentum Operator

```python
def lin_mom(x, exp = 1):
    """

    Args:
        x: The variable in which the linear momentum operator is with respect to
        exp: The exponent of the linear momentum operator

    Returns:
        The linear momentum (p) operator for a selected variable (x, y, etc.)
    """

    if exp == 1:
        return (-i*hbar*Derivative("1", x))
    else:
        return (-i*hbar*Derivative( lin_mom(x, exp - 1), x))
```

```python
lin_mom(x)
```

$$-\hbar i \frac{d}{dx} \boxed{1}$$

```python
lin_mom(x, 2)
```

$$-\hbar i \frac{\partial}{\partial x}\left(-\hbar i \frac{d}{dx} \boxed{1}\right)$$

```python
lin_mom(x, 3)
```

$$-\hbar i \frac{\partial}{\partial x}\left(-\hbar i \frac{\partial}{\partial x}\left(-\hbar i \frac{d}{dx} \boxed{1}\right)\right)$$

$Derivative(\ ,x)$

Code: Computing Commutators

$$[\hat{x}, \widehat{p_x}]f(x) = i\hbar f(x)$$

```
p = lin_mom(x, 1)
f = Operator(Function("f")(x))

comm(x, p, f)
```

$$\hbar i f(x)$$

$$\left[\hat{x}, \widehat{p_x^2}\right] f(x) = 2i\hbar p f(x)$$

```
p = lin_mom(x, 1)
f = Operator(Function("f")(x))

comm(x, p**2, f)
```

$$2\hbar^2 \frac{d}{dx} f(x)$$

$$\left[\widehat{p_x^2}, \widehat{x^2}\right] f(x) = 2\hbar^2(-2xf'(x) - f(x))$$

```
p = lin_mom(x, 1)
f = Operator(Function("f")(x))

comm(p**2, x**2, f)
```

$$2\hbar^2 \left(-2x \frac{d}{dx} f(x) - f(x)\right)$$

Code: `steps` Function

$[\hat{x}, \widehat{p_x}]$

```
p = lin_mom(x, 1)

comm_steps(x, p, f(x))
```

$$\left[x, -\hbar i \frac{d}{dx} 1\right] f(x)$$

$$x - \hbar i \frac{d}{dx} 1 f(x) - -\hbar i \frac{d}{dx} 1 x f(x)$$

$$\hbar i f(x)$$

$\left[\widehat{p_x^2}, \widehat{x^2}\right]$

```
p = lin_mom(x, 1)

comm_steps(p**2, x**2, f(x))
```

$$-\left[x^2, \hbar^2 i^2 \left(\frac{d}{dx} 1\right)^2\right] f(x)$$

$$-x^2 \hbar^2 i^2 \left(\frac{d}{dx} 1\right)^2 f(x) + \hbar^2 i^2 \left(\frac{d}{dx} 1\right)^2 x^2 f(x)$$

$$2\hbar^2 \left(-2x \frac{d}{dx} f(x) - f(x)\right)$$

- Provides a way to help students **learn** how to compute commutators rather than just giving students an answer

Jupyter Notebook: Commutators

1. Introduction to Commutators

1.1 Momentum and Position

In this first example, the momentum operator (defined in pysces as `lin_mom(x, exp)` where `x` is the variable of interest, and the position `q` is shown. You can calculate the commutator by using the `comm(commutator_1, commutator_2, aux)` function. The first arguement is the first variable, the second argument is the second variable, and the `aux` argument describes the auxilliary function, usually of the form `f(var)` where `var` represents the variable of interest, which is usually what the first and second variables are with respect to.

```
p = lin_mom(q, 1)

comm(p, q, f(q))
```

$-\hbar i f(q)$

Quantized Hamiltonian Dynamics

Theory

Methods Development

Applications

Theory: Quantized Hamiltonian Dynamics

- Based on the paper:
  Akimov, A. V.; Prezhdo, O. V. Formulation of Quantized Hamiltonian Dynamics in Terms of Natural Variables. *J. Chem. Phys.* **2012**, *137* (22), 224115. https://doi.org/10.1063/1.4770224.

Heisenberg Equation of Motion: $$i\hbar \frac{d\langle \hat{A} \rangle}{dt} = \langle [\hat{A}, \hat{H}] \rangle$$

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

Theory: Quantized Hamiltonian Dynamics

- Decomposition Procedure:
  - To end the list of equations of motion, a closure approximation is used

$$\langle ABC \rangle \approx \langle AB \rangle \langle C \rangle + \langle AC \rangle \langle B \rangle + \langle BC \rangle \langle A \rangle - 2 \langle A \rangle \langle B \rangle \langle C \rangle$$

  - This creates a decomposition into a product of lower order variables, thus ending the hierarchy

Theory: Quantized Hamiltonian Dynamics

$$\frac{d\langle q\rangle}{dt} = \frac{\langle p\rangle}{m}$$

$$\frac{d\langle p\rangle}{dt} = -\langle V'\rangle$$

$$\frac{d\langle q^2\rangle}{dt} = 2\frac{\langle (pq)_s\rangle}{m}$$

$$\frac{d\langle (pq)_s\rangle}{dt} = \left(\frac{p^2}{m} - \langle (qV')_s\rangle\right)$$

$$\frac{d\langle p^2\rangle}{dt} = -2\langle (pV')_s\rangle$$

$$\boxed{(AB)_s = \frac{AB + BA}{2}}$$

Quantized Hamiltonian Dynamics

Methods Development

Objective: Have Python derive these formulas of interest

$$\frac{d\langle q\rangle}{dt} = \frac{\langle p\rangle}{m}$$

$$\frac{d\langle p\rangle}{dt} = -\langle V'\rangle$$

$$\frac{d\langle q^2\rangle}{dt} = 2\frac{\langle (pq)_s\rangle}{m}$$

$$\frac{d\langle (pq)_s\rangle}{dt} = \left(\frac{p^2}{m} - \langle (qV')_s\rangle\right)$$

$$\frac{d\langle p^2\rangle}{dt} = -2\langle (pV')_s\rangle$$

Code: Quantized Hamiltonian Dynamics

The Hamiltonian:

```python
def ham(p, q):
    """
    Args:
        p: The variable for the given function. This is the momentum operator.
        q: The variable for the given function. This is the position operator.

    Returns:
        The Hamiltonian operator, made up of the kinetic energy operator and the general potential energy
operator.

    """
    p, q, v, mass = symbols("p q v mass")
    v = Function("v")
    return Operator(lin_mom(q, 2)/(2*mass)) + Operator(v(q))
```

```
1  ham(p, q)
```

$$-\frac{\hbar i \frac{\partial}{\partial q}\left(-\hbar i \frac{d}{dq} 1\right)}{2mass} + v(q)$$

Code: Quantized Hamiltonian Dynamics

The Time Derivative Function:

- 150+ Lines of Code

```
def time_deriv(var, order = 1):
    """
    Args:
        var: The variable of interest to take the time derivative of. For example, the position or momentum
operator.
        order: The order of the exponent of the variable of interest. The default value is 1

    Returns:
        The time derivative of the variable of interest to the desired order.

    """
    ...
```

$$i\hbar \frac{d\langle \hat{A} \rangle}{dt} = \langle [\hat{A}, \hat{H}] \rangle$$

Code: Quantized Hamiltonian Dynamics

The Time Derivative Function:

$$i\hbar \frac{d\langle \hat{A} \rangle}{dt} = \langle [\hat{A}, \hat{H}] \rangle$$

1. Compute the Hamiltonian using the placeholder values

```
def time_deriv(var, order = 1):
    ...
    h1 = str(comm_1(Operator(var**order), ham(p, q),
f(q)))
    ...
```

Objective: Replace the "placeholder" values with what is supposed to be taken the derivative of

$time\_deriv(q, 1)$

--hbar*i*Derivative(-hbar*i*Derivative(1, q), q)/(2*mass)*q*f(q) - v(q)*q*f(q) +
q*-hbar*i*Derivative(-hbar*i*Derivative(1, q), q)/(2*mass)*f(q) + q*v(q)*f(q)

Code: Quantized Hamiltonian Dynamics

The Time Derivative Function:
2. Find the “placeholder” values and the “end” values

$$i\hbar \frac{d\langle \hat{A} \rangle}{dt} = \langle [\hat{A}, \hat{H}] \rangle$$

```
...
s = h1.find("Derivative(1, q)", start)
while s != -1:

...
    start = s + len(f"Derivative(1,
q)")
...
    e = h1.find("f(q)", start) + 4
...
```

$time\_deriv(q, 1)$

--hbar*i*Derivative(-hbar*i*Derivative(1, q), q)/(2*mass)*q*f(q) - v(q)*q*f(q) +
q*-hbar*i*Derivative(-hbar*i*Derivative(1, q), q)/(2*mass)*f(q) + q*v(q)*f(q)

Code: Quantized Hamiltonian Dynamics

The Time Derivative Function:
3. Find the function that will be replace the placeholder

$$i\hbar \frac{d\langle \hat{A} \rangle}{dt} = \langle [\hat{A}, \hat{H}] \rangle$$

```
...
for i in range(len(start_points)):
  func = start_points[i] + len(f"Derivative(1, q)")
  func1 = h1.find("*", func, end_points[i])
  ...
  repl = h1[start_points[i]:end_points[i]].find("1") +
start_points[i]
  ...
  new_func = h1[func1 - 3:end_points[i]]
  ...
...
```

$time\_deriv(q, 1)$

--hbar*i*Derivative(-hbar*i*Derivative(1, q), q)/(2*mass)*q*f(q) - v(q)*q*f(q) +
q*-hbar*i*Derivative(-hbar*i*Derivative(1, q), q)/(2*mass)*f(q) + q*v(q)*f(q)

Code: Quantized Hamiltonian Dynamics

The Time Derivative Function:
4. Replace the placeholder with the expression of interest

$$i\hbar \frac{d\langle \hat{A} \rangle}{dt} = \langle [\hat{A}, \hat{H}] \rangle$$

```
...
        for i in range(len(nested_list)):
            temp[nested_list[i][0]] = \
                temp[nested_list[i][0]].replace(temp[nested_list[i][0]][nested_list[i][1]], "1/" +
new_derivative_function[i][1:])
...
```

--hbar*i*Derivative(-hbar*i*Derivative(1, q), q)/(2*mass)*q*f(q) - v(q)*q*f(q) +
q*-hbar*i*Derivative(-hbar*i*Derivative(1, q), q)/(2*mass)*f(q) + q*v(q)*f(q)

Code: Quantized Hamiltonian Dynamics

The Time Derivative Function:

$$i\hbar \frac{d\langle \hat{A} \rangle}{dt} = \langle [\hat{A}, \hat{H}] \rangle$$

5. Solve

```
...
return sympify(s2)
```

SymPy

```
sympy.core.sympify.sympify(a, locals=None, convert_xor=True, strict=False,
    rational=False, evaluate=None)                                  [source]
```

Converts an arbitrary expression to a type that can be used inside SymPy.

Code: Quantized Hamiltonian Dynamics

The Time Derivative Function:

```
--hbar*i*Derivative(-hbar*i*Derivative(1, q), q)/(2*mass)*q*f(q) - v(q)*q*f(q) +
        q*-hbar*i*Derivative(-hbar*i*Derivative(1, q), q)/(2*mass)*f(q) + q*v(q)*f(q)
```

```
time_deriv(q, 1)
```

$$\frac{p}{mass}$$

Code: Computing Equations of Motion for Quantized Hamiltonian Dynamics

$$\frac{d\langle q\rangle}{dt} = \frac{\langle p\rangle}{m}$$

```
time_deriv(q, 1)
```

$$\frac{p}{mass}$$

$$\frac{d\langle p\rangle}{dt} = -\langle V'\rangle$$

```
time_deriv(p, 1)
```

$$-\frac{d}{dq}v(q)$$

$$\frac{d\langle q^2\rangle}{dt} = 2\frac{\langle (pq)_s\rangle}{m}$$

```
time_deriv(q, 2)
```

$$\frac{2pq}{mass}$$

$$\frac{d\langle (pq)_s\rangle}{dt} = \left(\frac{p^2}{m} - \langle (qV')_s\rangle\right)$$

```
time_deriv((p*q+q*p)/2, 1)
```

$$-q\frac{d}{dq}v(q) + \frac{p^2}{mass}$$

$$\frac{d\langle p^2\rangle}{dt} = -2\langle (pV')_s\rangle$$

```
time_deriv(p, 2)
```

$$-2p\frac{d}{dq}v(q)$$

Quantized Hamiltonian Dynamics

Applications

$$\frac{d\langle q\rangle}{dt} = \frac{\langle p\rangle}{m}$$

$$\frac{d\langle p\rangle}{dt} = -\langle V'\rangle$$

$$\frac{d\langle q^2\rangle}{dt} = 2\frac{\langle (pq)_s\rangle}{m}$$

$$\frac{d\langle (pq)_s\rangle}{dt} = \left(\frac{p^2}{m} - \langle (qV')_s\rangle\right)$$

$$\frac{d\langle p^2\rangle}{dt} = -2\langle (pV')_s\rangle$$

Results: Quantized Hamiltonian Dynamics

Morse Potential:

$$V(q) = D * \left(1 - e^{-\alpha(q-q_0)}\right)^2$$

Variables:

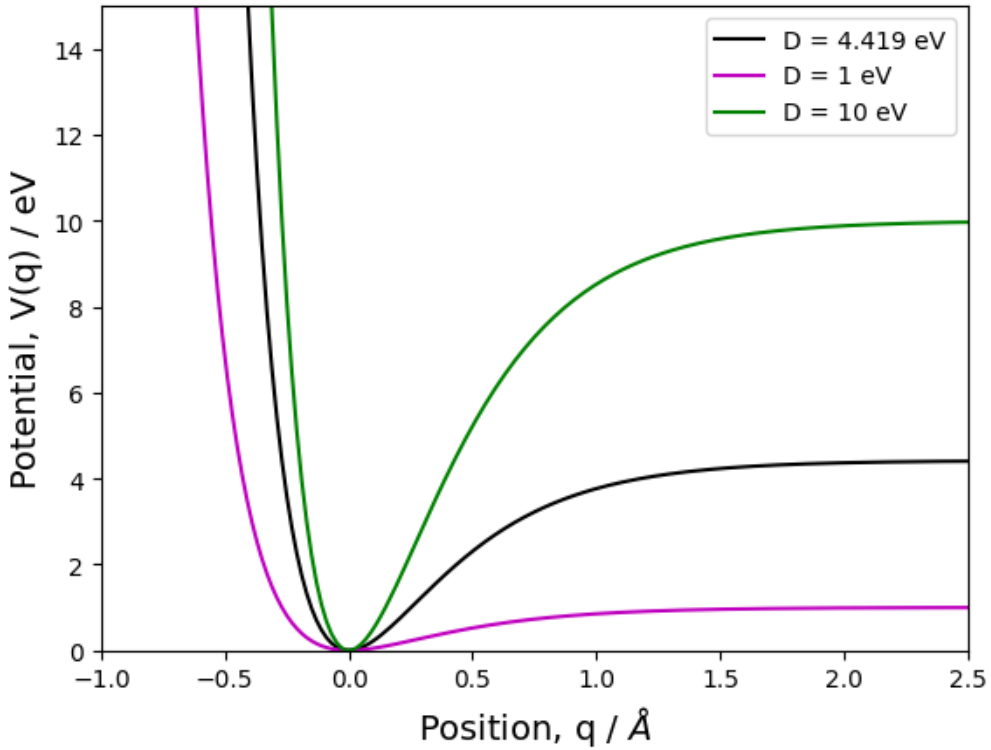
mass = mass of water

q0 = equilibrium position

alpha = “width” of potential

D = well depth

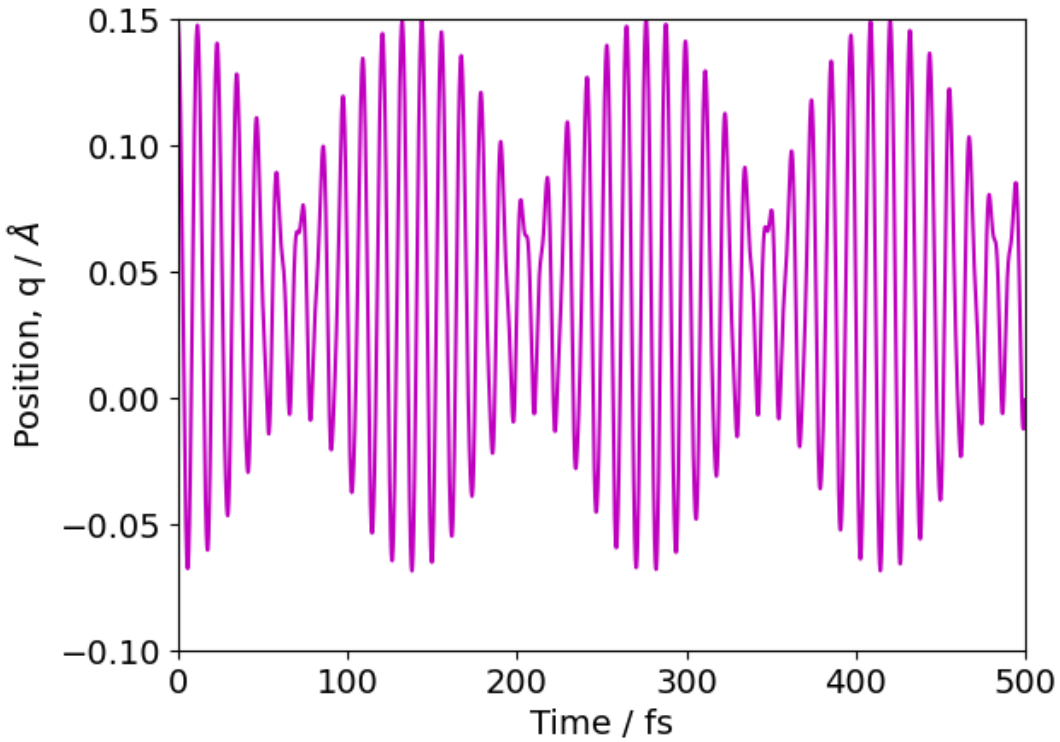
dt = timestep

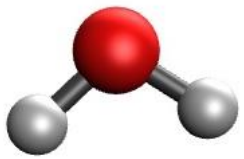


- Graphs created with MatPlotLib library

Results: Quantized Hamiltonian Dynamics

Morse Potential





Variables:

mass = 1836 a.u.

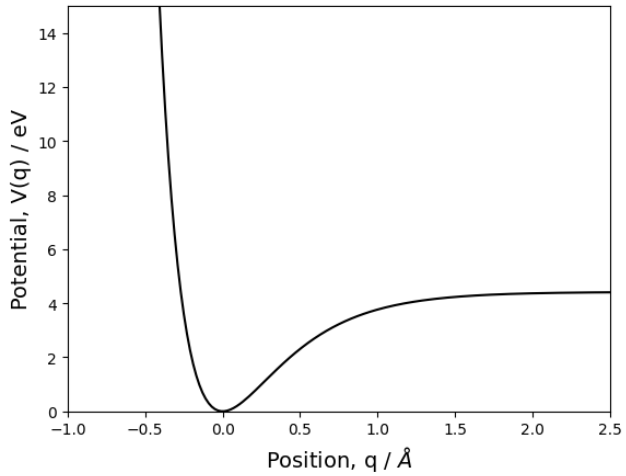
q0 = 0.0 Å

q = 0.15 Å

alpha = 2.567 $Å^{-1}$

D = 4.419 eV

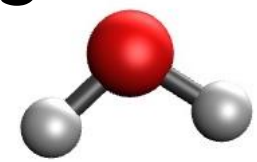
dt = 0.1 fs



- Graphs created with MatPlotLib library

Akimov, A. V.; Prezhdo, O. V. Formulation of Quantized Hamiltonian Dynamics in Terms of Natural Variables. *J. Chem. Phys.* **2012**, *137* (22), 224115. https://doi.org/10.1063/1.4770224.

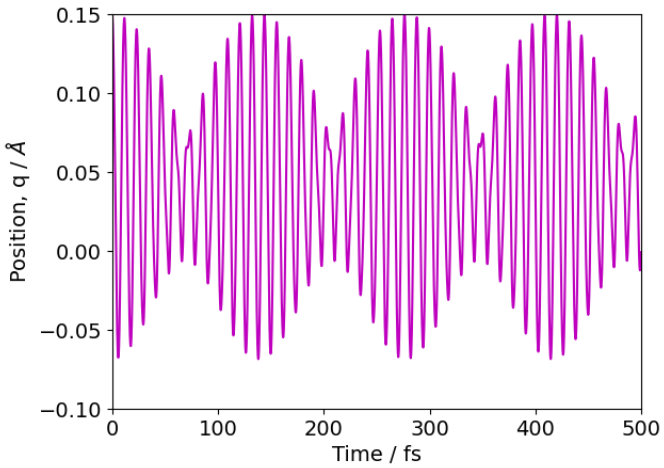
Results: Quantized Hamiltonian Dynamics



Morse Potential

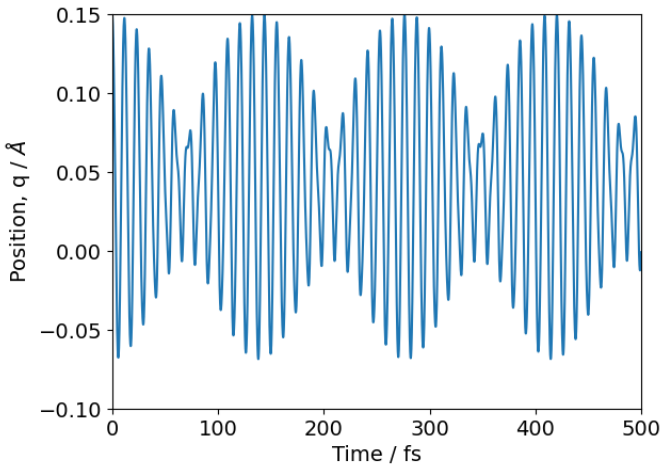
Symbolic Method

- Provides a simple yet versatile way of computing the equations of motion
- Computes the equations of motion using one function
- Can be applied to any potential



Original Method

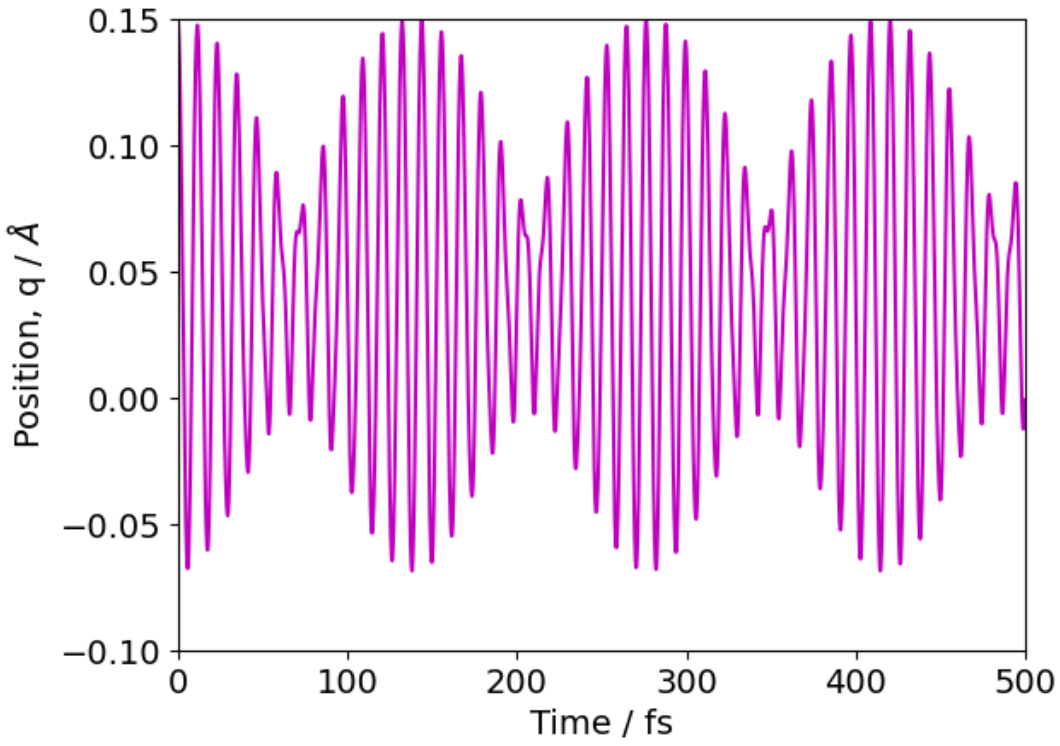
- Had to compute each equation of motion by hand
- Uses different functions for each equation of motion
- Specific to one potential



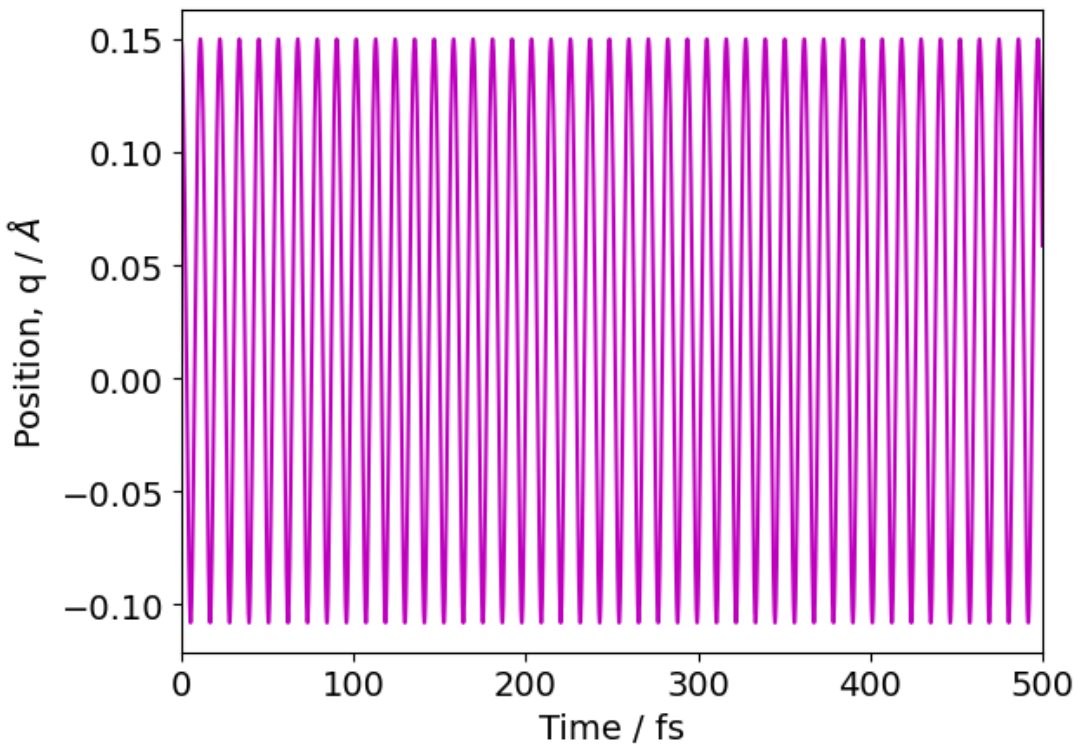
Results: Quantized Hamiltonian Dynamics

Morse Potential

QHD Method



Classical Method



- Graphs created with MatPlotLib library

Results: Quantized Hamiltonian Dynamics

Gaussian Potential:

$$V(q) = -V_0 * e^{(-\alpha * q^2)}$$

Where:

$$\alpha = \frac{1}{2 * \sigma^2}$$

Variables:

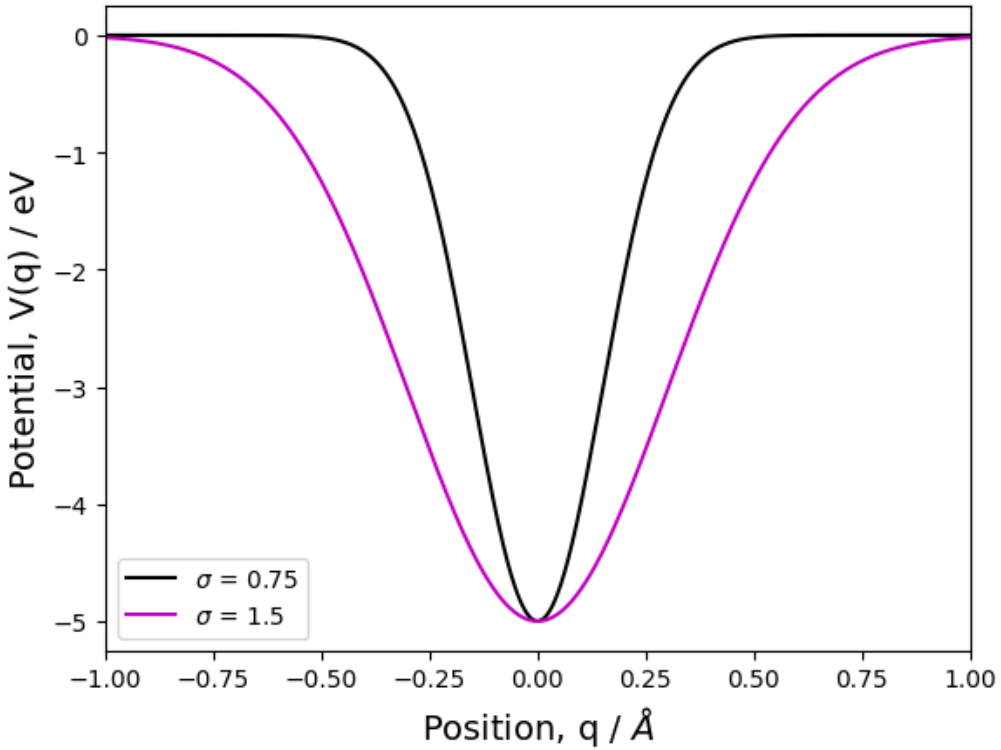
mass = mass of water

V0 = well depth

sigma = broadness of well

q = position

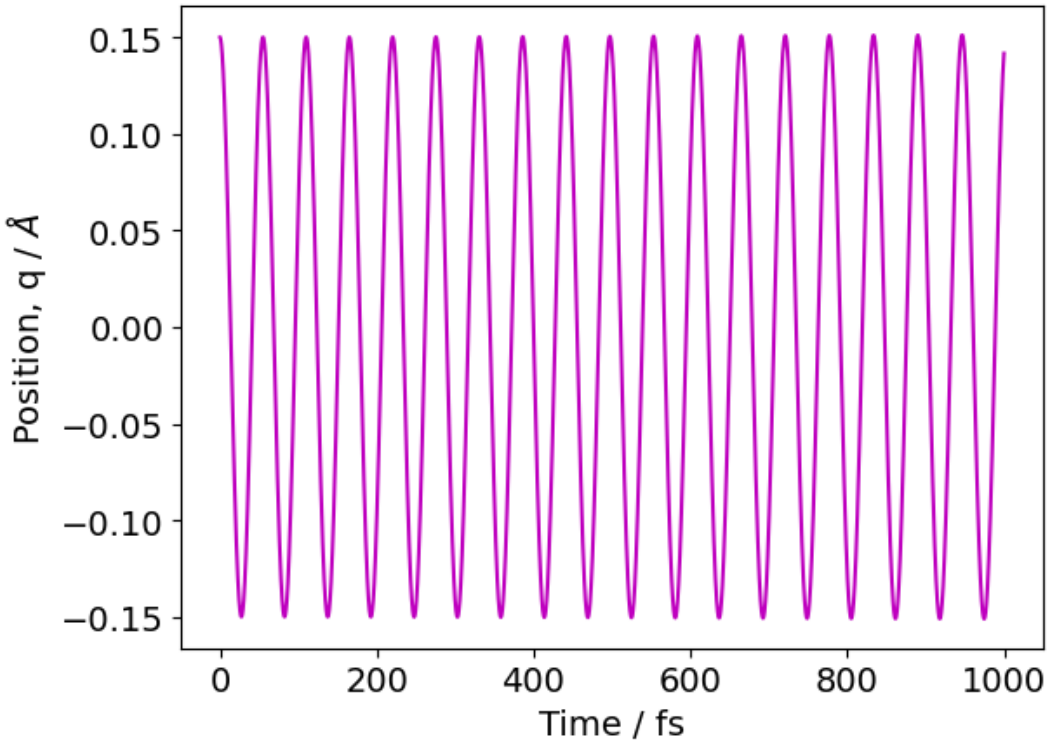
dt = timestep

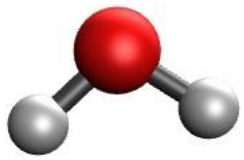


- Graphs created with MatPlotLib library

Results: Quantized Hamiltonian Dynamics

Gaussian Potential





Variables:

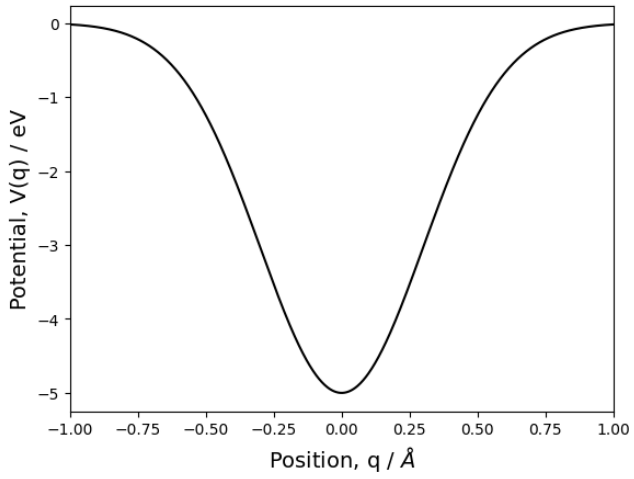
mass = 1836 a.u.

V0 = 5 eV

sigma = 1.5 Å

q = 0.15 Å

dt = 0.1 fs



- Graphs created with MatPlotLib library

Jupyter Notebook: Quantized Hamiltonian Dynamics

Below, empty lists are created to store the data generated. Feel free to create your own lists to examine how different variables change over time.

```
##### Empty Lists for Data Storage: #####

q_list = []
p_list = []
```

Q2. Create an empty list for `p2` values and observe how this variable changes over time.

```

```

Now that the parameters have been defined, we can compute the desired values of position and momentum over a given time interval.

Q3. Try changing the time step (`dt`) or the finish time (`t_f`).

Results: Quantized Hamiltonian Dynamics

- Tunneling

Potential:

$$V(q) = \frac{q^2}{2} + aq^3$$

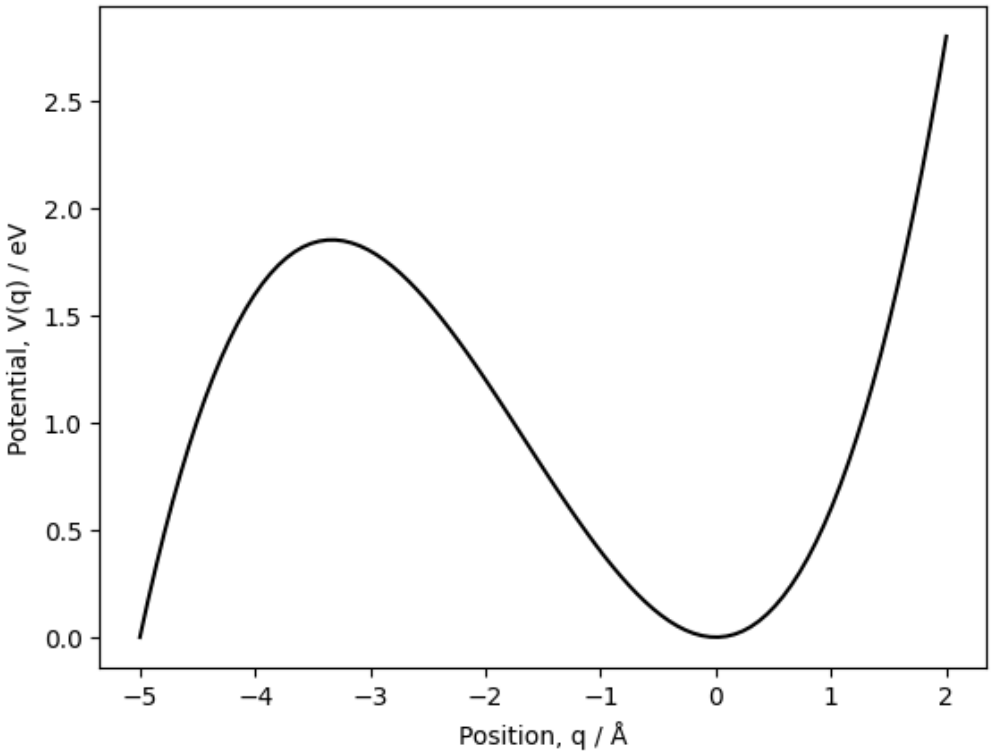
Variables:

mass = mass of water

q = position

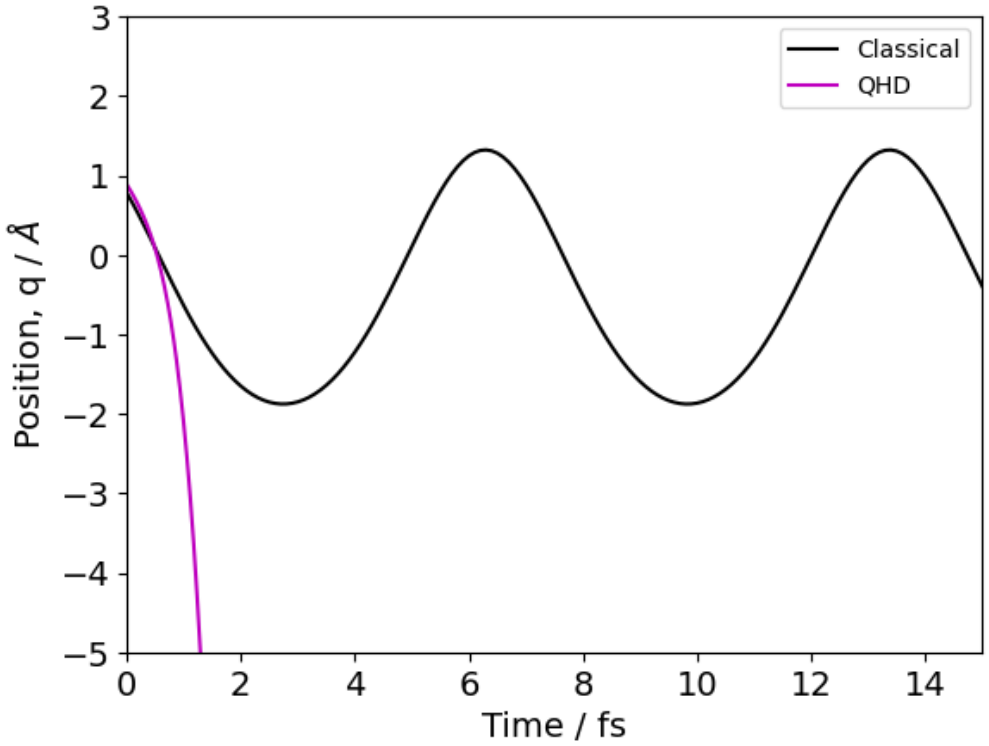
a = barrier size

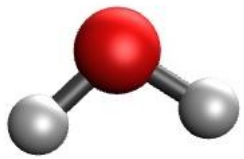
dt = timestep



Results: Quantized Hamiltonian Dynamics

Tunneling





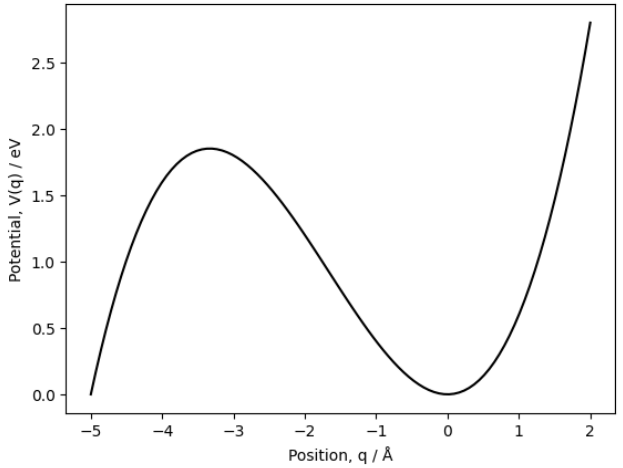
Variables:

mass = 1836 a.u.

q = 1 Å

a = 0.1

dt = 0.1 fs



Conclusions

- The PySyComp library is an effective introduction to using Python, Jupyter Notebooks, and the SymPy and Matplotlib libraries
- The `time_deriv` function is effective in producing the desired equations of motions

Working on…

- Running the QHD code from the command line
- Optimizing the code

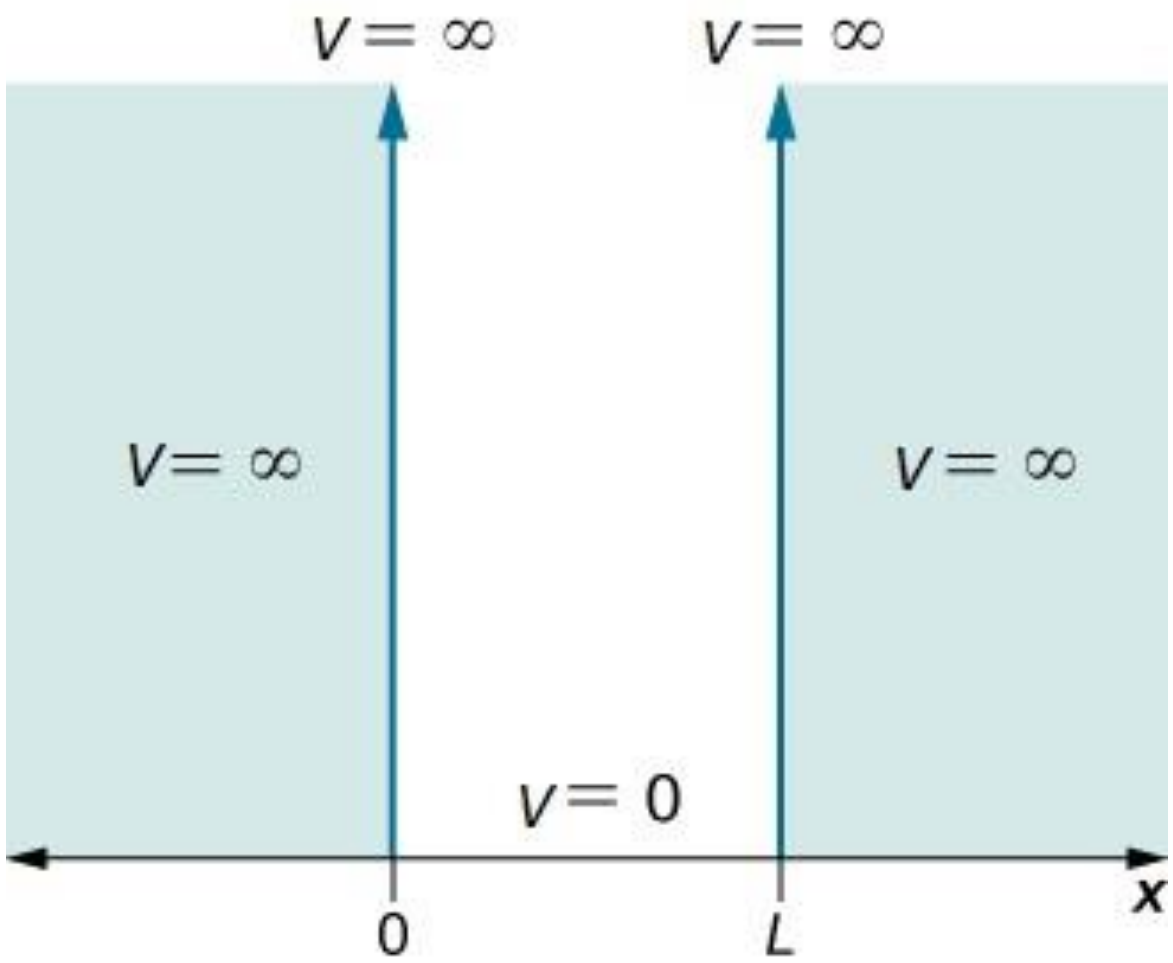
Extra Slides

Outline

Background & Motivation	Particle in a Box	Commutators	Quantized Hamiltonian Dynamics
	Computing the normalization constant, expectation values	Computing the commutator `steps` function	Methods development Applications

Particle in a Box

An Introductory Problem to Quantum Mechanics for Undergraduate Students

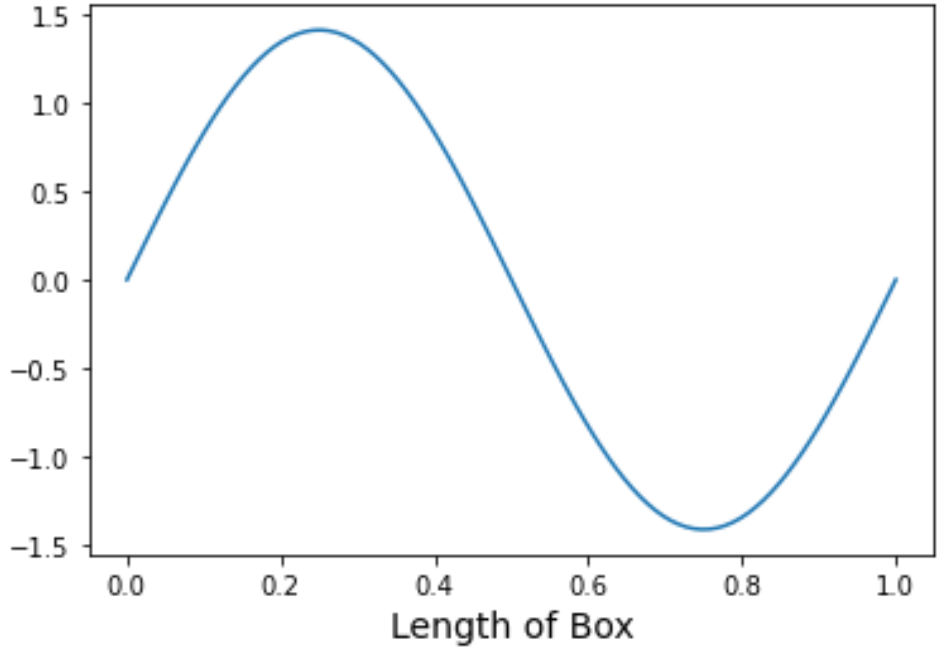


Theory: Particle in a Box

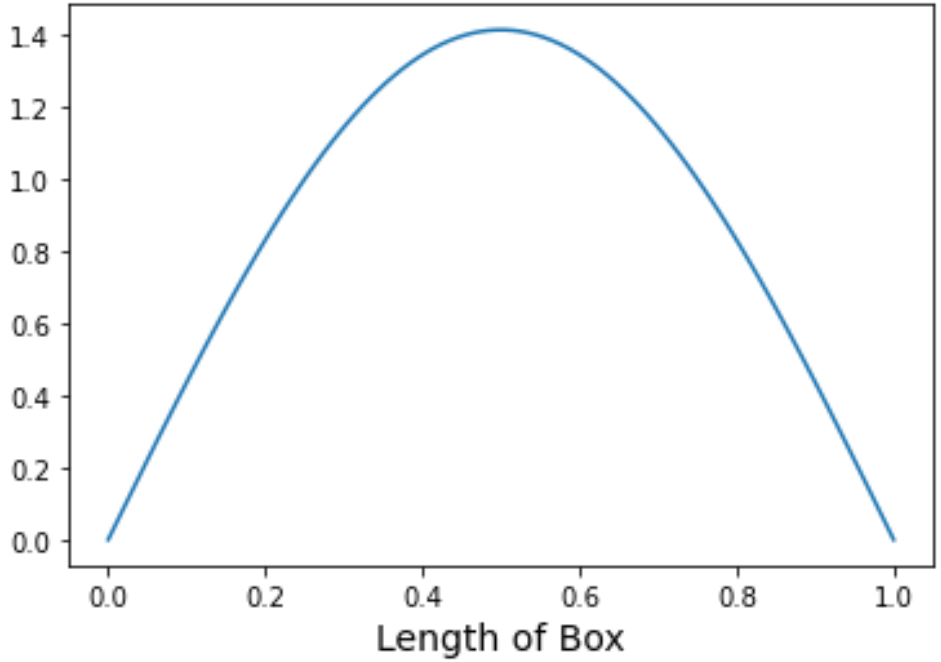
- If x < 0:
  - $V = \infty$
- If x > L:
  - $V = \infty$
- If 0 < x < L:
  - $V = 0$

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

n=2



n=1



- Graphs created with MatPlotLib library

Code: Particle in a Box

PIB Wave Function:

In [3]:

```
PIB(x, L, n)
```

Out[3]:

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

PIB Normalization Constant:

In [4]:

```
normalization_constant(PIB(x, L, n), 0, L, x)
```

Out[4]:

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

PIB Expectation Value:

In [8]:

```
expectation_value(PIB_normalized(x, L, n), kinetic_energy(x), PIB_normalized(x, L, n), 0, L, x)
```

Out[8]:

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

- Provides a straightforward way to compute different values associated with the Particle in a Box problem

Jupyter Notebook: Particle in a Box

1. Normalizing Particle in a Box

This can be done in one line, using the `PIB()` function, and the `normalization_constant()` function

1. Try running just the `PIB()` function with variables `x`, `L`, and `n`.

```
1
```

2. Now, try normalizing using the `normalization_constant()` function with the same parameters, from `0` to `L`.

```
1
```

Please note the normalization constant provides the *constant* and not the normalized wave function. To get the normalized wave function, multiply the normalization constant with the original wave function.

3. What is the normalized wave function?

```
1
```

Results: Quantized Hamiltonian Dynamics

- Morse Potential: Change of Variables

$$x = e^{-\alpha q} \qquad V(q) = D[\langle x^2 \rangle - 2\langle x \rangle]$$

Same $$\frac{d\langle q \rangle}{dt} = \frac{\langle p \rangle}{m}$$

$$\frac{d\langle p \rangle}{dt} = -\langle V' \rangle \longrightarrow \frac{d\langle p \rangle}{dt} = 2\alpha D[\langle x^2 \rangle - \langle x \rangle]$$

$$\frac{d\langle x \rangle}{dt} = -\alpha \frac{\langle (px)_s \rangle}{m}$$

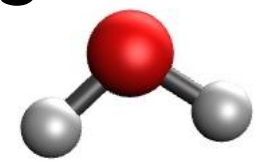
Same $$\frac{d\langle q^2 \rangle}{dt} = 2\frac{\langle (pq)_s \rangle}{m}$$

$$\frac{d\langle x^2 \rangle}{dt} = -2\alpha \frac{\langle (px^2)_s \rangle}{m}$$

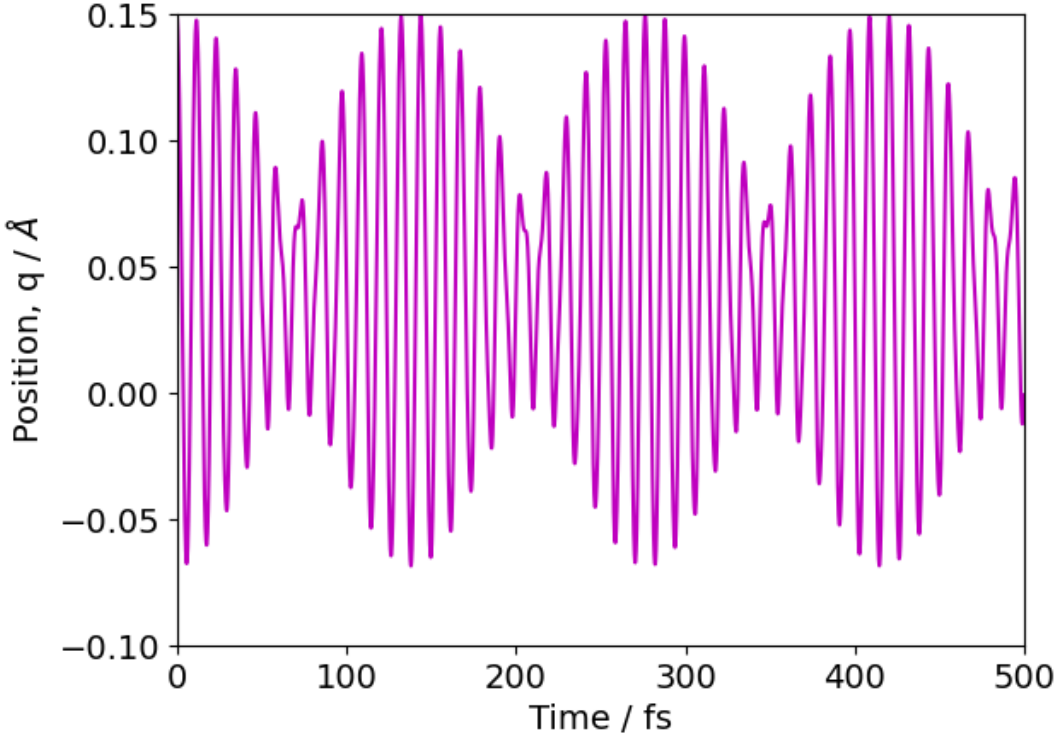
$$\frac{d\langle (pq)_s \rangle}{dt} = \left(\frac{p^2}{m} - \langle (qV')_s \rangle\right) \longrightarrow \frac{d\langle (pq)_s \rangle}{dt} = \left(\frac{p^2}{m} + 2\alpha D[\langle (qx^2)_s \rangle - \langle (qx)_s \rangle]\right)$$

$$\frac{d\langle p^2 \rangle}{dt} = -2\langle (pV')_s \rangle \longrightarrow \frac{d\langle p^2 \rangle}{dt} = 4\alpha D[\langle (px^2)_s \rangle - \langle (px)_s \rangle]$$

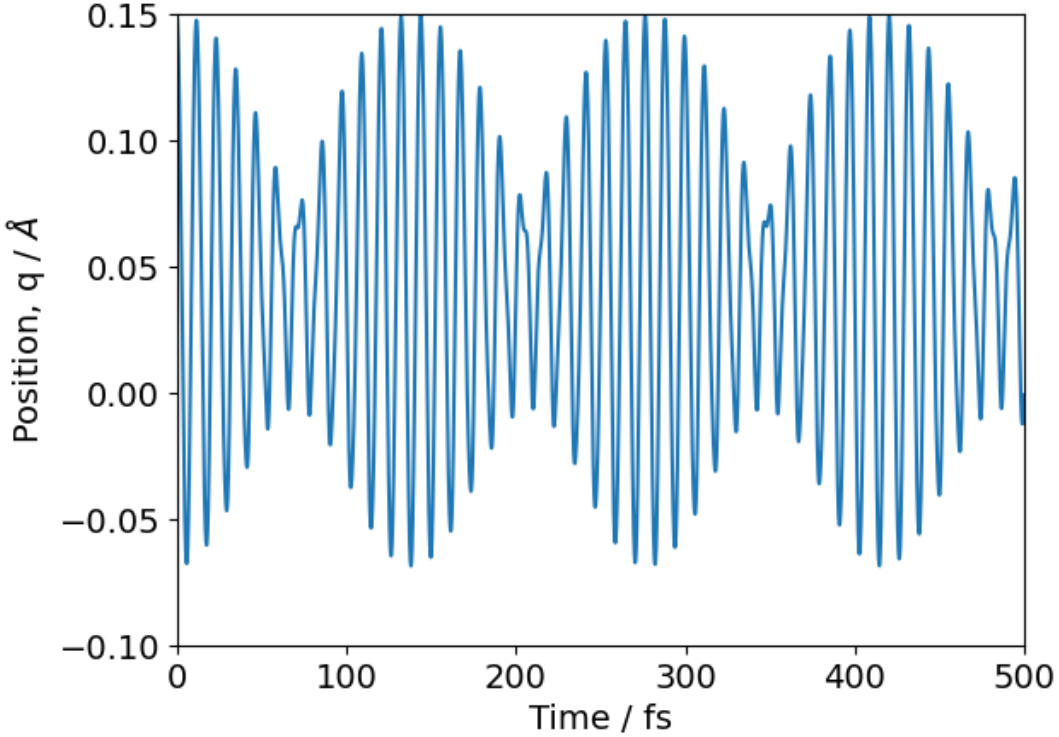
Results: Quantized Hamiltonian Dynamics



Morse Potential



Symbolic Method



Original Method

Results: Quantized Hamiltonian Dynamics

Variables:

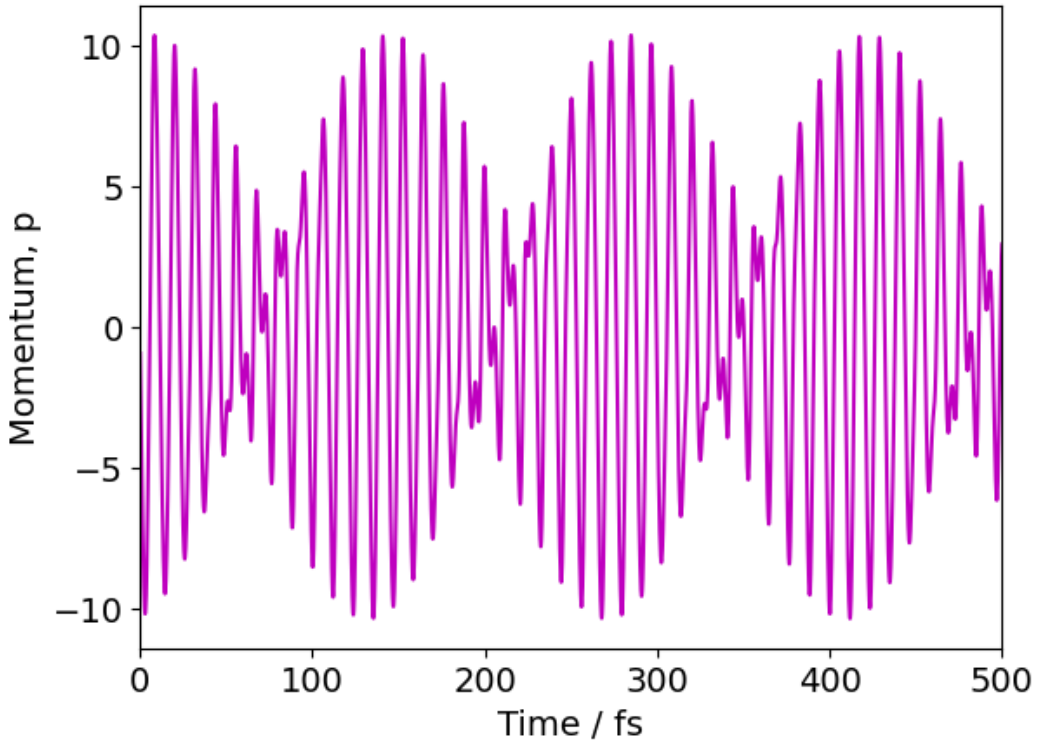
mass = 1836 a.u.

q0 = 0.0 Å

alpha = 2.567 $Å^{-1}$

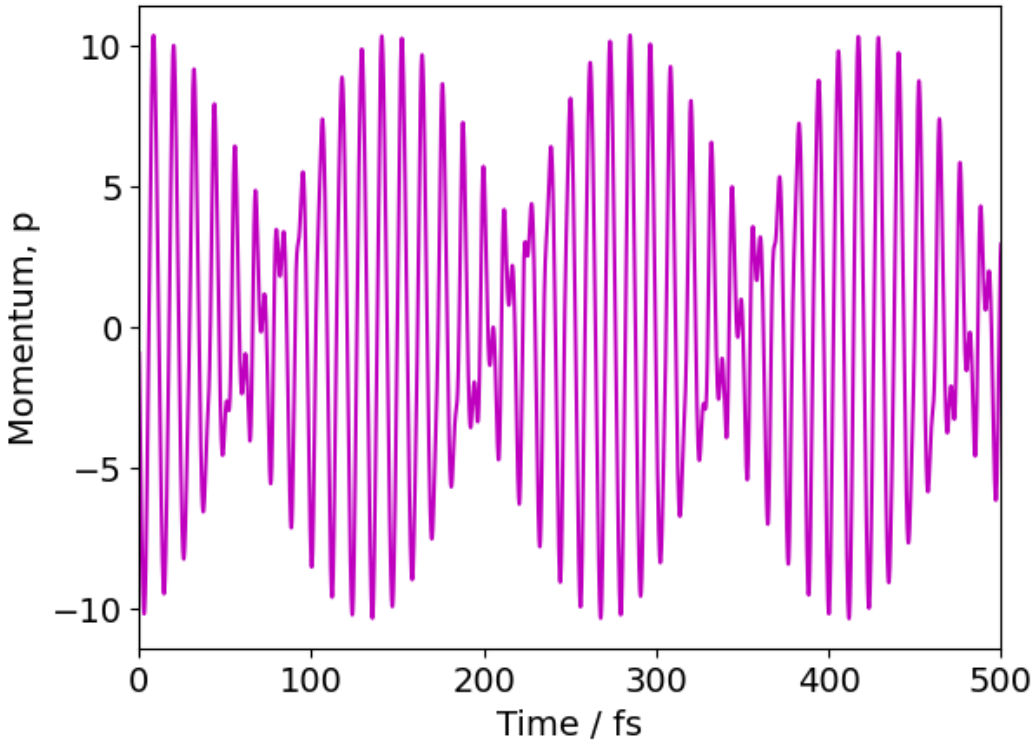
D = 4.419 eV

dt = 0.1 fs

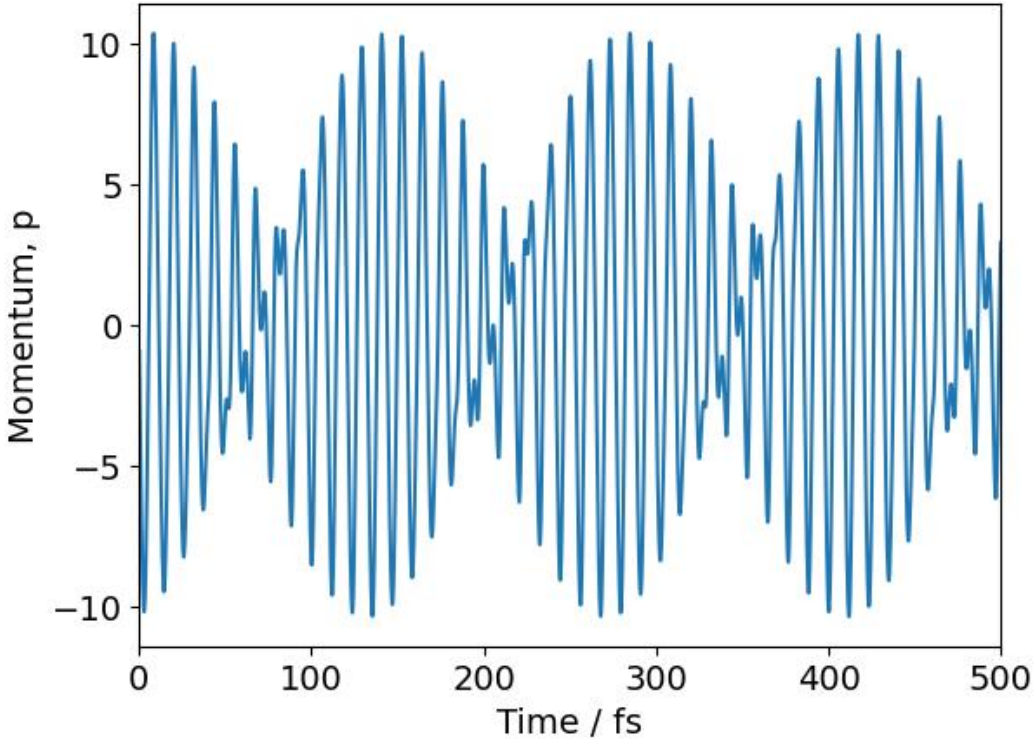


Akimov, A. V.; Prezhdo, O. V. Formulation of Quantized Hamiltonian Dynamics in Terms of Natural Variables. *J. Chem. Phys.* **2012**, *137* (22), 224115. https://doi.org/10.1063/1.4770224.

Results: Quantized Hamiltonian Dynamics

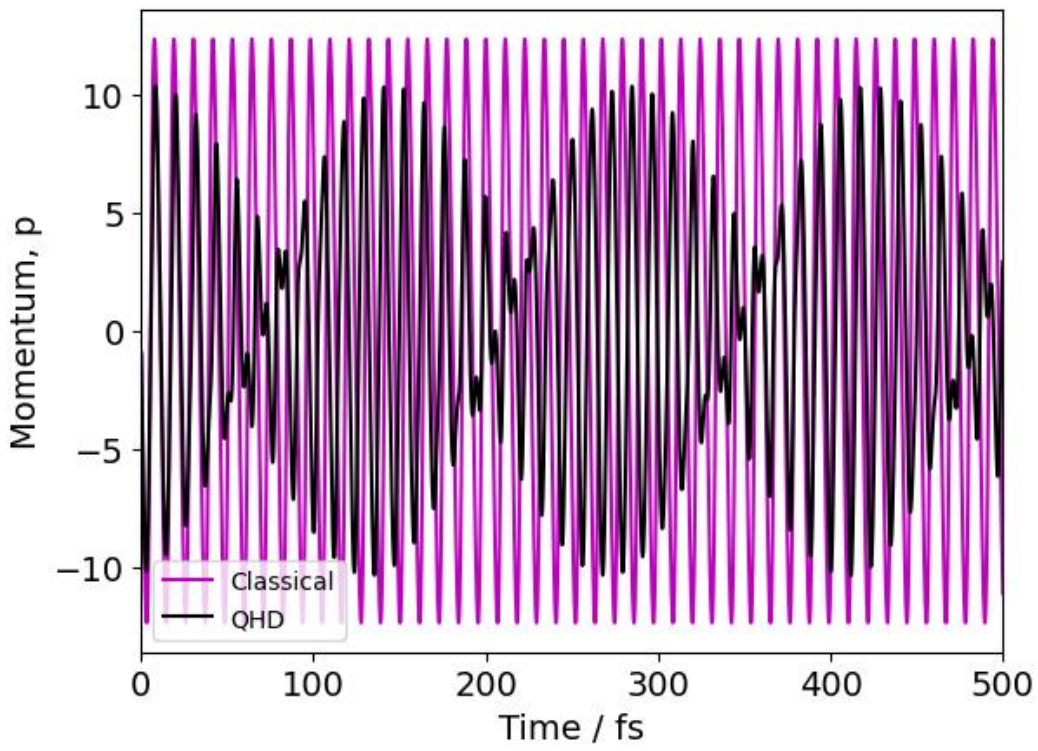


Symbolic Method



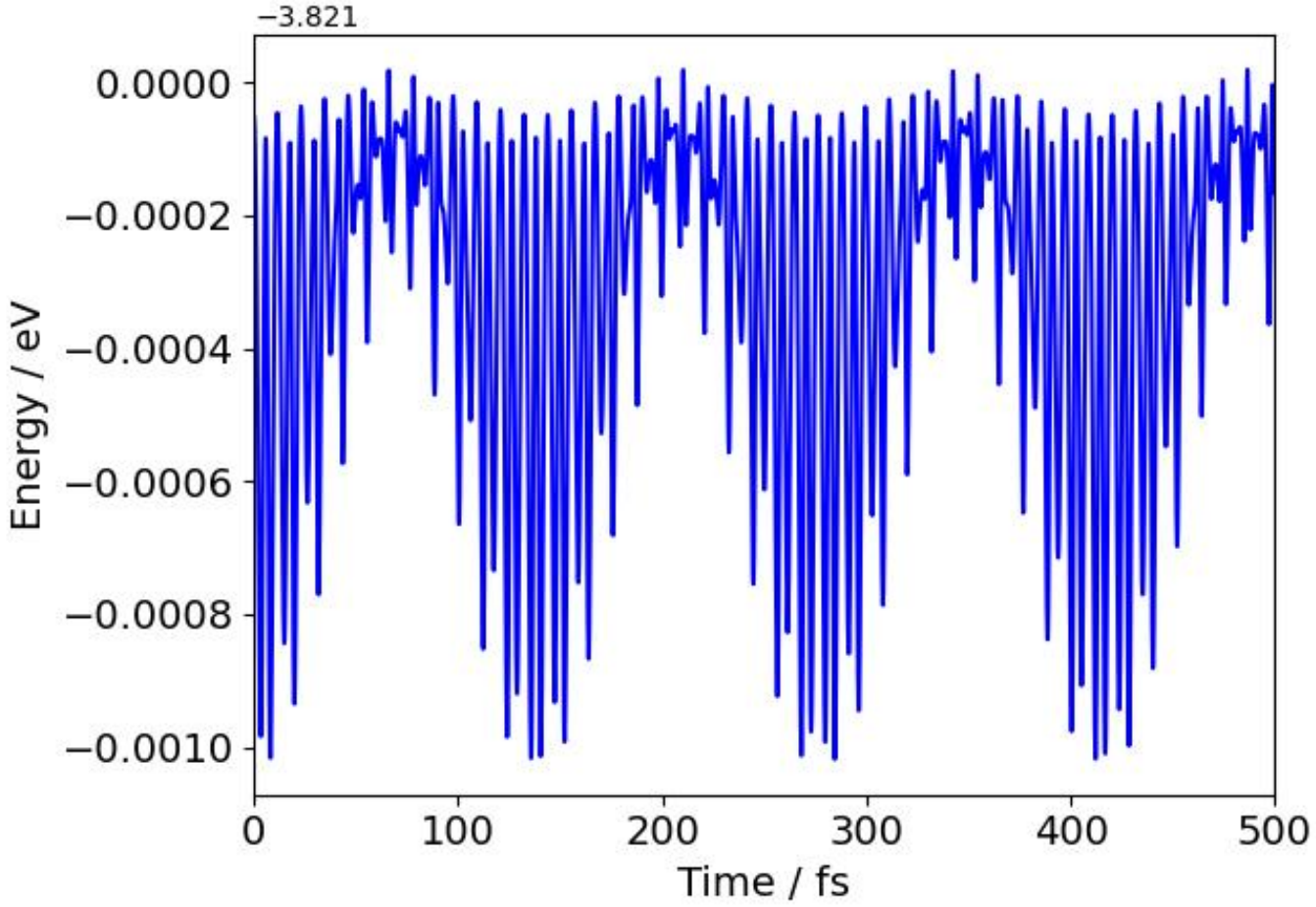
Original Method

Results: Quantized Hamiltonian Dynamics

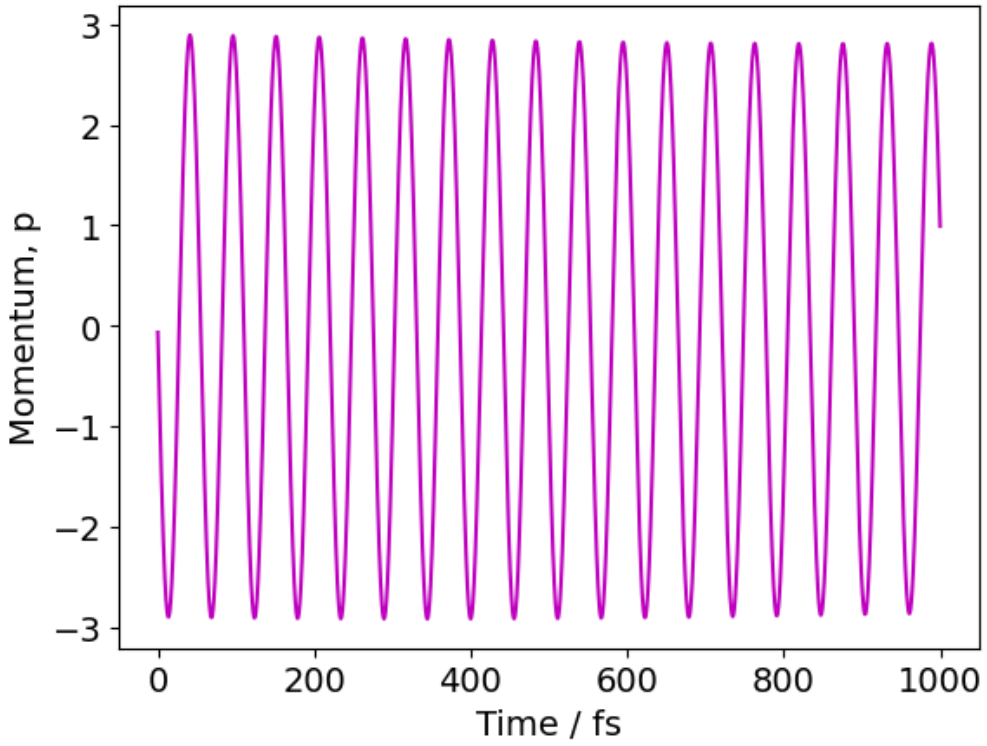


Results: Quantized Hamiltonian Dynamics

Energy Conservation



Results: Quantized Hamiltonian Dynamics



Results: Quantized Hamiltonian Dynamics

Energy Conservation