Problem Set 5

Showing your work

Your homework assignments should show incremental development with at least two different commits over two different days of work. Assignments that do not show incremental development will incur a 5% penalty.

AI Acceptable Use

"AI" refers to Artificial Intelligence and includes tools like ChatGPT, Claude, GitHub Copilot, and other language models or AI-powered assistants. AI tools may be used to research ideas or clarify concepts. However, they should not be used to generate code or complete assignments. All submitted work must reflect your own understanding and effort.

If you use AI in this assignment, include a statement in your README reflection on what AI you used and how you used it.

Python - Object-Oriented Programming, Decorators, and Context Managers

For this homework, you will design a set of Python classes to model the behavior of different types of gases: real gases and ideal gases. **Some of the design parameters for these classes are already defined in the provided testing file (`test_gases.py`).** However, please read the assignment explanation before moving to this file.

Besides pytest, you should only use the Python Standard Library for this assignment.

Ideal gases are commonly used to introduce thermodynamic concepts, with their behavior described by the equation:

$$PV = nRT$$

where:

- (P) represents pressure,
- (V) is the volume,
- (n) is the number of moles,
- (R) is the ideal gas constant, and
- (T) is the temperature.

The behavior of real gases is more complex. Real gases can be modeled different equations with one of them being the van der Waals equation. The van der Waals gas equation takes particle volume and interactions into account through parameters (a) and (b), as described by the equation:

$$\left(P + \frac{an^2}{V^2}\right)(V - nb) = nRT$$

This homework will have three main parts: **Class Design**, **Testing**, and **Documentation and Discussion**. You should first think about class structure (without implementing code), then use the provided tests in (2) to write your code.

Part 1 - Class Design

Design a class structure to model the behavior of gases. You should create two classes for this: `IdealGas` and `RealGas`. Decide whether to use inheritance or composition, and whether a base class is needed. You do not need to write code yet - more details such as what should be in the constructor for Part 2. Think about the equations for `RealGas` and `IdealGas`when making your decision.

You have a few options when it comes to designing the two classes. You might choose to write one as a base class then utilize inheritance, or you may decide to implement them separately. Consider properties such as temperature, volume, and moles, and determine where they belong in your design.

Part 2 - Developing with Test Driven Development

Test Driven Development to develop your code. Test Driven Development is a development technique where tests are created before the code is developed.

Tests for ideal gases are provided in `test_gases.py`. Use the provided tests to guide your implementation of the classes. You should write your `IdealGas` class so that these tests pass.

Recall that you can run these tests by making sure `pytest` is installed and running the following ommand in your terminal:

```
pytest -v
```

In addition to using the provided tests, you should also develop your `RealGas` class. You should also write additional tests for your `RealGas` class.

Part 3 PyTest Decorators and Context Managers

The provided tests are written as individual tests. You should refactor the tests to use the pytest parametrize. You will also identify a context manager used in the provided tests that is part of pytest which will allow you to test for expected exceptions.

Documentation and Discussion

As always, you should include overview information about how to install and run your code in the `README.md` file. In addition, you should include a discussion of your class design decisions.

1. **Class Design Decisions**: Discuss the relationship between `RealGas` and `IdealGas`. Did you choose to use inheritance, composition, or separate classes? What factors influenced your decision?

2. **Shared Properties**: Explain how you handled shared properties like temperature, volume, and moles. Why did you place these attributes in certain classes, and how did that contribute to code clarity and reusability?

3. **Mixing Behavior**: Describe how you approached the implementation of gas mixing parameters. How did you implement these methods?

4. **Decorator Usage**: Where did you use `@property` in your class? Why was it useful for your design? Did you use any other decorators, and how did they simplify or improve your code?

5. **Testing and Design Alignment**: How did the provided test cases influence your class design? Did you need to adjust your design decisions to ensure your classes passed the tests?

6. **Future Extensibility**: If you needed to model other real gas equations beyond the van der Waals equation, how would your design change, if at all? Was your current approach flexible enough to accommodate this?

Debugging a C++ application

The file `crashes.cpp` contains a program that sometimes crashes. Use your debugging knowledge to find the issue.

Write up the cause of the issue in the README file. Include any output from debuggers or sanitizers that helped you.

C++ - Eigen and Principal Moments of Inertia

In this repo, the `molecule.cpp` file contains some of the Molecule class from the last problem set. I have given you a version of the code that calculates the inertia tensor. You have the following two tasks - convert the code to use Eigen rather than nested `std::array`, and then to use Eigen to compute the principal moments of inertia, from which you will determine the molecular rotor type.

We talked about the Eigen library in lecture. I have included a version of the library in this repo, so you do not have to install it. Use `-I./include` when compiling the `molecule.cpp` file.

Use the functions you create in this section on the two molecules in the `main` function - one representing water, and the other methane.

Makefile: Create a makefile that compiles and runs your code

Converting the code to use Eigen

In the current code, the `inertia_tensor` function returns a 3x3 matrix stored in nested arrays. This is kind of annoying, and it should really be stored in a matrix.

Modify the code so that Eigen matrices are used rather than nested arrays.

Principal Moments of Inertia

The principal moments of inertia are an important quantity which can be used to determine some things about rotational behavior of the molecule, as well as for aligning molecules in a standard orientation.

The principal moments of inertia are calculated by diagonalizing the inertia tensor, forming its eigenvectors and eigenvalues.

For an example of how to do this in Eigen, see here.

Write a class method that returns the eigenvalues from the above diagonalization.

Note: The inertia tensor is represented by *symmetric* matrices. The eigenvalues and eigenvectors of a matrix are complex unless the matrix is hermitian (or self-adjoint), in which case they are real. A real, symmetric matrix is hermitian/self-adjoint, so we can use the eigensolver from Eigen that handles self-adjoint matrices. If you used the regular Eigensolver, you would have to allow for complex eigenvalues and eigenvectors.

Molecular Rotor Type

The eigenvalues obtained from the previous step can be used to determine the molecular rotor type.

- If all moments are equal (to within some tolerance), then the rotor type is a "spherical top"
- If one moment is small (close to zero) and the others are equal, the rotor type is "linear"
- If two moments are equal (but different from the third) then the rotor type is is a "symmetric top"
- If all three moments are different, the rotor type is "asymmetric top"

Write a class method that returns the molecular rotor type (as a string). Use this function on the two in the `main` function to determine the molecular rotor types.