Compute Software and Infrastructure Intern Assignment One

January 20, 2025

Instructions

As a compute software and infrastructure intern you will work across multiple teams to support modeling and simulation of reactor systems in the areas of research, design, safety, etc. In the nuclear space, we must develop and maintain code for analyses that have safety related implications making software quality critical to mission success. In this assignment, you are tasked with the development of a python module to tackle toy problems which will assess many of the skills you need to be successful in this intern role. The following problem scenarios constructed are fictitious and may vary widely from the applications in which you are tasked if hired. These scenarios should not require advanced/in-depth domain specific knowledge but such knowledge may be need to be learned in the actual role. You will be assessed on problem completion, performance, and implementation. Additionally, questions may offer multiple methodologies to solve a single problem. It will be up to your discretion to decide which is the best option. Programmatic testing is provided and reports the results of critical tests, performance, and edge case tests. It is recommended that you start at the beginning and progress through the tasks in order as they increase in difficulty. Read all instructions carefully and complete the tasks in order as they are interdependent and will become increasingly difficult. You may use any resources available to you but all work should be your own.

Environment

The environment is provided for you in the form of a docker image which can be found for both x86 and arm64 at `https://hub.docker.com/repository/docker/mskairospower/csione`. You should pull the **latest** image for your architecture. We suggest cloning the repo in the first problem on your local machine and bind mounting **-v $(pwd):$(pwd)** such that you can easily package your solution. The default entry environment will be the user **dev** in /**home**/**dev**. Do not bind to this location, choose a sub-folder such as /**home**/**dev**/**repo** as you will prevent environment activation.

After instantiating a container from the image you will notice there is a **conda** environment called official. This will be the official environment you are tested against. You may develop in any environment of your choice, however, your submission will be tested strictly against what is contained in the provided image. This consists of Python 3.10 and limited additional packages on Ubuntu 24.04. You are not restricted from installing additional packages to verify your solutions, however, these packages will not be available in the test environment and attempting to use them in submissions will result in failure.

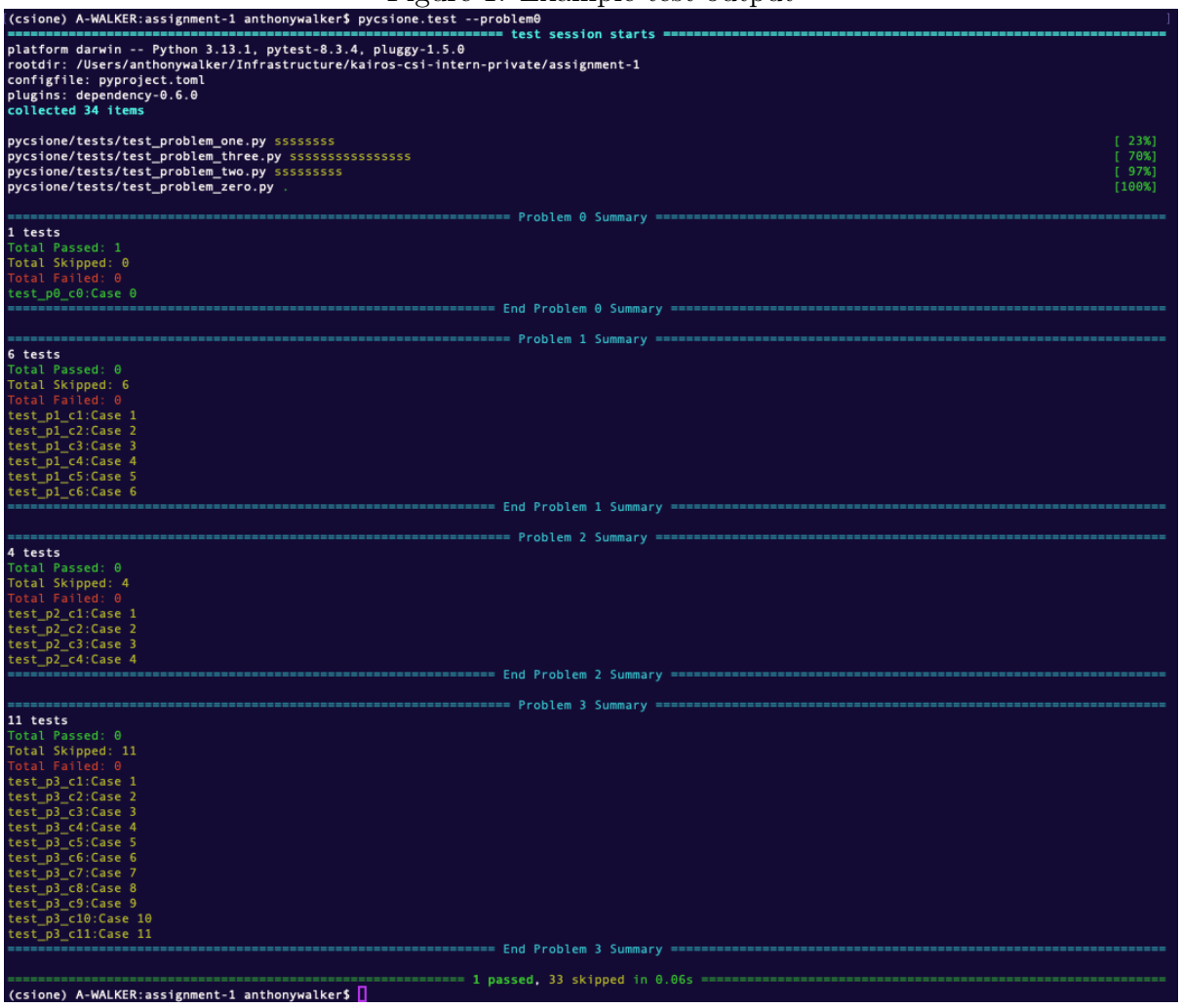
Additionally, there is a provided package called **pycsione** which provides some utilities which you may leverage in completing these tasks. Visible test cases are available to you by importing **from pycsione.cases import Cases** and investigating the Cases object. This package also contains testing for your provided module which you may execute at any time. You may execute the tests for a given problem with the command **pycsione.test** –**problem***. Omit the –**problem*** flag to execute all test cases for all problems. Be sure to visit the **Submission** section even if you have not completed all parts of the assignments.

Problem 0: Signing Your Name

To get you acquinted with solving these problems and running the tests, we are going to start with something simple. Follow the steps to get started. **These steps are very important as this will include your name with your work.**

- Clone the repo `https://github.com/anthony-walker/kairos-csi-intern-public.git`.
- In the file, **solution.name** set the **MY_NAME_IS** variable to your name.
- Complete the **pyproject.toml** in **pysolution** to install a package called **solution** from the given directory.
- Make it so you can import **MY_NAME_IS** variable directly from as **from solution import MY_NAME_IS**.
- Install your package or ensure it is on your **PYTHONPATH** and run **pycsione.test –problem0** to verify that you have done this correctly. You should see output resembling that in Figure 1.
- Note, some tests may reveal a percentage next to them. This is your performance score in comparison to our developed solution. For example, a score of 10% means that your code took 10 times longer to run than the internal solution. Focus on performance if you have the time, after you have completed all of the problems.

Figure 1: Example test output



Problem 1: Arrhenius Interpolation

Now the that the you are acquinted, imagine that you have just started as an engineer and been tasked with estimating the rates of reactions for flammable gases. In high temperature environments, accumulation of flammable gas is always a concern. Consequently, it is important that we are able to estimate the accumulation of potentially flammable substances and provide adequate ventilation or methods. The net production rates of chemical species in kinetics simulations are determined via the law of mass action:

$$\dot{\omega} = k_r \sum_{i}^{P} [n_i]^{\nu_i} - k_f \sum_{i}^{R} [n_i]^{\nu_i}, \tag{1}$$

where P is equal to the number of product species, R is equal to the number of reactant species, k_i is known as the rate constant, and $[n_i]$ is the concentration of species i.

The rate constant is typically determined by an arrhenius expression and equilibrium constant for elementary reactions,

$$k = A \exp\left(\frac{E_a}{RT}\right). \tag{2}$$

In the event that the Arrhenius expression does not capture the physical behavior well, we must use alternative expressions or non-continuous experimental data. Your first task is to develop code to make the necessary estimates. Part of the task has been done for you with the implementation of **pycsione.rate_constant.RateConstant**. The remainder should be completed with the following steps.

- Inside **solution/interpolation_rate.py** develop a class called **InterpolationRate** that inherits from **pycsione.rate_constant.RateConstant** and is instantiated with a list of points (T_i, k_i).
- Overload the **__call__** method to use these points and evaluate the rate constant at a given temperature.
- Ensure that the class can be imported with the statement **from solution import InterpolationRate**.
- Your software will be used one day by other contributors, consequently, be sure to capture as many edge cases as possible with **pycsione.exception.PYCSIException**.
- Test your solution by running **pycsione.test** and view the **Problem One Summary**.

Problem 2: ODE Solution

Great Work! Analysts were able to use your code up to this point to estimate rates of flammable species. Their initial estimates revealed that we need more in-depth analyses and should be solving zero-dimensional kinetics systems at the very least. Modeling physical phenomena typically requires solving systems of ordinary or partial differential equations (ODE/PDE)s. There are many different numerical methods to solve these problems based on the circumstance. Your next task is to develop a solver called **KineticSolver** which uses a simple numerical integration technique of your choice to solve a system of linear ODEs represented by Equation (1). The solver should be able to be imported as **from solution import KineticSolver**. **KineticSolver** should have a method **solve**, which takes the final time of the simulation and time step as floats. This function should return a list of the concentrations at the given final time.

The following assumptions and guidelines apply to the simulation.

- There are many more potential edge cases in this example, assume that all test cases will be valid inputs.
- Assume the simulation starts at time, $t = 0$.
- Assume that if the final time is not equally divisible by the time step that the number of steps taken should be rounded up.
- Assume that temperature for the system as a function of time is known (provided input).
- Assume the gas can be modeled as an **ideal gas** ($Pv = nRT$) with a **constant volume**.
- Assume that all reactions are first order, e.g., $A <=> B$ (eliminating the need for a non-linear solution step). The order of the reaction, if you are unfamiliar, is the exponent of concentration terms in Equation (1).
- Assume the simulation is zero dimensional with no spatial variation.
- Species will strictly start with capital letters and only contain alphabetical characters and numbers.
- Reactions will be in the format $"B <=> C"$ with consistent spacing.
- Reactions will be stoichiometrically balanced.
- A forward and reverse rate will be provided for all reactions. The order of these rates will match the order of the reactions.
- The order of the given species will match the order of given initial concentrations.
- Units are in SI: m^3, s, kmol, kJ, K.

This solver should be instantiated with **pycsione.solver_input.SolverInput** which provides the following:

- **SolverInput.T**, A function that returns temperature as a function of time.

- **SolverInput.initial_conditions**, a list of initial concentrations at time, $t = 0$.
- **SolverInput.fwd_rates**, a list of forward reaction rates which take an argument, temperature (T).
- **SolverInput.rev_rates**, a list of reverse reaction rates which take an argument, temperature (T).
- **SolverInput.reactions**, a list of reactions in the model.
- **SolverInput.species**, a list of species included in the model.

Problem 3: Markov Scrubbers

Congratulations on making it this far, there is only a little ways left to go! Your work has been invaluable up to this point, analysts have used your former code to assess concentrations of flammable species inside of active plants. The bad news is they discovered that there are some cases in which dangerous limits are achieved. The good news is they have a plan to restore safe operation and you can help them achieve it. Analysts used the results to develop probability distributions of how concentrations will shift amongst accumulation zones in active plants. They need you to write a code which determines the optimal positioning of a "scrubber" which will operate regularly to remove any accumulated flammable gases.

Your task, develop a class called **Delegator** with a method called **delegate** that internally assigns two properties **best** and **worst**. These properties are the best and worst locations (integer indices) to place a scrubber given the probability distribution. We need both as sometimes the best position may be taken by other equipement but making the worst choice could be catastrophic.

The **Delegator** should be instantiated with a list of lists, **P**, which indicates the probability of the gas shifting between zones. For example,

$$P = \begin{bmatrix} 0 & 0.25 & 0.75 & 0 \\ 0.25 & 0 & 0.75 & 0 \\ 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

indicates that there is a 75% likelihood of shifting from zones 1 and 2 to zone 3 and a 25% likelihood of shifting to the other opposite zone respectively. From zone 3, there is equal likelihood of shifting to zones 1 and 2. Therefore, a scrubber should be placed in zone 3 (index 2). Additionally, there is potential for stagnant accumulation zones (zone 4 in the former example), i.e., there is a 100% chance that gas that enters the zone will never leave. Permanent ventilation will be installed in these zones to ensure safety.

The **best** location is defined as the index that minimizes the total number of transitions taken. The **worst** location is defined as the index that maximizes the total number of transitions taken. In either case, assume that states are ordered in level of difficulty to reach, e.g., if indices 5 and 6 both take 10 transitions, then 5 is the better index.

You should follow the following guidelines in completing this problem:

- The probability distribution will always be square according to the number of accumulation zones, e.g., $n_z \times n_z$.
- When you place a scrubber in a state, no gas will leave that state.
- Transitions from any given state to a vented accumulation zone or scrubber should be rounded before forming the total number of transitions.
- Assume you will always have at least one zone which can be scrubbed.
- Assume all rows of the given probability matrix will sum to 1.
- Assume that states are in order of difficulty to reach.
- Assume that a lower number of transitions is more important than reachability, i.e., reachability does not matter if the number of transitions are greater.

Submission

You will notice that when you run **pycsione.test** that a file called **output** is produced. Your final submission should include your solution directory in a **zip** file along with this **output** file in the top directory. Ensure that your run **pycsione.test** without additional flags to capture all problems in the output.