Programming a Plasma Physics Calculator with the Use of Python

Krishna Patel, Dr. Jason Myatt, Sydney Easton

WISEST

August 11th, 2021

WISEST women in scholarship, engineering, science & technology
Canada
PSCL
UNIVERSITY OF ALBERTA

Abstract

Practitioners in plasma physics and engineering often make use of formula booklets when making calculations [e.g., The NRL Plasma Formulary...]. Computing these formulas by hand is often tedious and error prone, particularly if the quantities need to be converted from a different set of units. In our research, we decided to develop a Python program that solves this problem by taking user input, converting it into the correct units, performing the user's calculation and then outputting it with the units of the user's choice. This was done by using object oriented programming, python libraries that stored basic unit conversions, and tagged data which is essentially *tuples* that use a number as index zero and the respective units as index one. EX: (23.45, "cm"). Through implementing these three strategies we were able to successfully create a base program that is easy to use, manipulate, and further extend for future purposes. Therefore, by utilizing python to create a plasma physics calculator we were able to find an effective way to organize a plasma formulary. For future purposes, we would like to create a user interface for our program, and add additional formulas, and units to it.

Introduction and Motivation

Across the world many different measurement units are used, whether that be in engineering, science, technology, or just in daily life. Standardized units can also vary country to country for example, whereas Canada uses The Système International d'Unités (SI) the United States prefers The Imperial System of Measurement. Similarly, when it comes to plasma physics this same variation occurs. Depending on who is conducting the research, where it is taking place, etc units can change accordingly. However, this causes another problem because for certain formulas variables have to be in specific base units in order to ensure the calculation is done correctly. This starts to get difficult when you have various quantities that all have their own conversion factors because oftentimes one formula can have five to six different variables.

On top of variables these formulas also contain many different constant values which can create another colloquial to retrieve from data booklets because they can be hard to memorize. For example, the Debye length formula for electrons seen in Fig 1, is often described as a characteristic distance over which ions and electrons can be separated in a plasma (Chen, 1984). This formula takes in k which represents Boltzmann's constant, temperature T in electron volts (eV), density n in grams per cubic centimeter (1/cm³), and returns the Debye length in cm. However if instead of electron volts the temperature was in kelvin and the density was in kilogram per cubic centimeter (kg/cm³) this would change up the final units.

$$\lambda_D = (kT/4\pi ne^2)^{1/2} = 7.43 \times 10^2 T^{1/2} n^{-1/2}\,\mathrm{cm}$$

Figure 1: Debye Length Formula

This is just one example however for both temperature and density many other units exist. It would be much easier to plug numbers into a Python program that could easily convert the input into the correct units, perform the respective calculations, and then ask the user what units they would like their final answer in. This eliminates having to manually perform unit conversions by using various long formula sheets, and speeds up the process significantly.

Methods

The creation of this calculator was done using Python only, with imports like NumPy to do some extra mathematical calculations. The key goal remained creating two python classes. One called the *plasma* class which contained plasma formulary in *methods* that could be used to calculate the formula of the user's choice, example seen in Fig 2.

```
class Plasma_functions:
  def DebyeLength(temperature, density):
      """
          Taken from the NRL plasma formulary, J.D. Huber 2007.
          Temperature in eV
          Density in 1/cm^3
      """
      debye = 7.43e2*np.sqrt(temperature)/np.sqrt(density)
```

Figure 2: Example *Method* of the *Plasma* Class

The second class was named the *units* class, which was used to organize the unit conversions. We also implemented the *dir* built in Python function to print out all possible formulas the user could call on. We specifically used *dir* because this made it so in the future if any additional formulas were added, they would automatically print.

```
method_list = [method for method in dir
               (Plasma_functions) if method.startswith('__') is False]
print(method_list)
```

Figure 3: Printing *Methods* in the *Plasma* Class using the dir Function

In order to ensure that the unit conversions class had the ability to be easily extended in the future, we used the Python library to organize functions that contained basic unit conversions. For example, we had length and time conversions like cm, m, km, seconds, minutes, hours etc. Using these base conversions was key in order to ensure that the code remained concise. By doing so we eliminated the need for multiple functions for each conversion, for instance converting between radians/seconds and radians/minutes would typically require a function for both radians/seconds to radians/minutes and vice versa. However with our new implication we only require a seconds to minute method and minutes to second method which can both be reused for other units as well. The initial units and value input received from the user would be converted into tagged data, would go through a series of *if-else* statements in order to determine what conversions needed to take place and ensure the units were in the correct form before Python proceeded to calculate the answer.

```
input = tuple(input("Enter the temperature value and units separated by a space")
input.upper().split(" "))
in_value = float(tagged[0])
```

Figure 4: Converting User Input to Tagged Data

These *if-else* statements called on the unit conversion library which depending on what formula was being calculated also called on the *plasma* class. The final step was to prompt the user one to ask what units they would like their final answer in.

Then the data will once again go through a series of *if-else* statements to complete the final conversion. The final output is done using the format function in Python so the final answer can be presented in scientific notation. This whole user input, conversion, and calculation process is done through a *while loop* that can be used over and over again until the user specifies otherwise.

```
final_ans = "Here is your final answer {0:.2e}, {1}.".format(in_value, units_final)
print(final_ans)
```

Figure 5: Printing Final Output using the *Format* Function

Conclusion

Plasma formulary can be extensively used during research or coursework for students, typically calculating formulas would be a long and difficult process. However with the implication of a Python program that takes the formulas and also converts the answer into preferred units this process can be simplified making it much easier on researchers and students.

Although this is a base program containing only some formulas used in the field of plasma physics the calculator has been programmed in such a way that in the future extending it to include more formulas and unit conversions can be done without having to worry about changing the whole framework. Another future implication can be creating a user interface for the calculator making it even easier for users to input values, and units. It would also overall increase the user experience. Including a feature where the user can add in their own formulas using the interface would also be a great feature to have because as the research world is constantly evolving it may be deemed difficult to continuously add or update formulas.

Acknowledgments

I would like to thank my PI/Supervisor Dr. Jason Myatt for providing me with mentorship and valuable knowledge that allowed me to complete this research project. His encouragement and endless support throughout the program has been beyond helpful. I would also like to express my appreciation for my research partner, Sydney who has been a pleasure to work with and a true asset to this project. Another thank you to Process Solutions and Dr. Myatt for sponsoring me this summer and allowing me to participate in this incredible program. Last but not least thank you to the WISEST team for organizing the program and giving students like me the opportunity to participate in trail-blazing research,

Citations

Huba, J. D. (2006). 2006 Plasma Formulary - Handbook of Data and Tables for *Plasma Physics & Engineering*. Naval Research Laboratory. https://doi.org/10.21236/ada447173

Harris, C. R., Millman, K. J., & van der Walt, S. J. (2020). Array programming with NumPy. *Array Programming with NumPy*. Published.https://doi.org/10.1038/s41586-020-2649-2

Chen, F. F. (1984). *Introduction to Plasma Physics and Controlled Fusion*. SPRINGER. https://doi.org/10.1007/978-3-319-22309-4

Python 3.9.6 Documentation. (2020, October 5). Docs.Python. https://docs.python.org/3/