NMSC 2023: Final Project 1

Christian Cardin
christian.cardin@helsinki.fi

University of Helsinki — May 16, 2023

Abstract

This work analyzes the problem of ion stopping power in two-atom interactions, focusing on the systems of hydrogen (^1H) ions interacting with silicon (^{28}Si) targets and silicon ions interacting with gold (^{197}Au) targets. The method employed to tackle this problem involved implementing a simulation using numerical methods, specifically utilizing `scipy.optimize.root_scalar` for root finding and `scipy.integrate.quad` for integration. By varying the projectile energy, I analyzed the resulting stopping power and compared it with the result given by the universal stopping power formula [1]. The results confirm that the numerical approach fits well with the universal stopping power formula.

1 Introduction

Ion stopping power is a phenomenon of paramount importance in the study of ion-solid interactions. It refers to the energy loss experienced by ions as they traverse a solid target. Understanding this energy loss is crucial in various fields, including materials science, nuclear physics, and medical physics.
The problem of ion stopping power is challenging due to the complex interactions between the ion and the target atoms. As an ion passes through a material, it undergoes interactions with the atomic electrons and nuclei, resulting in energy transfer and dissipation. These interactions are influenced by factors such as the ion projectile's mass, charge, and energy, as well as the properties of the target material.
Studying ion stopping power allows us to gain insights into the fundamental physics behind ion-solid interactions. Moreover, it has practical applications, such as predicting the energy deposition in materials during ion implantation, designing radiation shielding, and understanding the behavior of particles in nuclear reactors and particle accelerators.
In this work, I aim to investigate the ion stopping power in the specific systems of hydrogen ions interacting with silicon targets and silicon ions interacting with gold targets. By employing a simulation based on numerical methods, we can analyze the energy loss of ions with varying projectile energies (E_{lab}). The simulation utilizes established numerical libraries for root finding and integration, included in `scipy`, ensuring accurate calculations of the relevant physical quantities associated with stopping power.

1.1 List of Files

Here is a list of files included in the project:

- `src/main.py`: Entry point of the program.
- `src/atom.py`: Contains the `Atom` class, which is used to store the properties of the atoms used in the simulation.
- `src/constants.py`: Contains the physical constants used in the simulation, some conversion factors and some shared global immutable variables.
- `src/formulas.py`: Contains the main formulas used in the simulation, corresponding to the equations in the project description.
- `src/studies.py`: Implements plotting of various functions used in the project, for better understsanding the nature of the problem and understanding the results.
- `src/benchmarks.py`: Implements benchmarking for some functions used in the computation.
- `output/*.png`: Plots generated by the simulation.
- `output/benchmark.csv`: benchmark data for selected functions, with time in ms.
- `output/out.txt`: A sample output of a full run of the simulation.

1.2 Running the Code

The code was written with python 3.10.4, and tested with python 3.8.10 installed in the Pangolin server. The following packages are required: numpy, scipy, matplotlib and pandas. The code can be run by executing the `main.py` file from the command line. During the execution it will generate plots and output files into a folder `output/`. Note that the program's output will be print to stdout.

Command Line

```
pip3 install numpy scipy matplotlib pandas

cd /path/to/project/src
python3 main.py
```

2 Methods

To determine the distance of minimum approach between the projectile and target atoms, it is necessary to find the solution equation (4) $g(r_{min}) = 0$, which is done by function `find_r_min()`. This function utilizes the `root_scalar()` method from the `scipy.optimize` module, which automatically selects an appropriate bracketing algorithm for root finding between several available. Bracketing algorithms are commonly used in root finding to locate the root of a function within a specific interval. These algorithms involve narrowing down the interval that contains the root by iteratively selecting sub-intervals until the interval length reaches a desired accuracy.
Functions `scattering_angle()` and `stopping_power()` use the `quad()` method from the `scipy.integrate` module for solving the integral in equations (11) and (8) respectively. This method utilizes numerical integration techniques provided by the QUADPACK Fortran library. QUADPACK implements efficient quadrature algorithms to compute definite integrals by dividing the integration interval into smaller sub-intervals, selecting a set of sample points within each sub-interval and assigning weights to these points. The function values at these sample points are then multiplied by their corresponding weights and summed to obtain an approximation of the integral.

3 Implementation

Python was chosen as the primary programming language for the implementation of the software for its ease of use, making it accessible to both beginner and experienced programmers. Its simple and intuitive syntax allows for quick development and easy maintenance of the codebase. Additionally, Python boasts a vast collection of mature scientific libraries, such as NumPy, SciPy, and Matplotlib, which provide powerful functionalities and accessible APIs specifically designed for scientific computations.
When developing the code, emphasis was placed on simplicity and readability. The code follows a modular structure, with each function designed to be self-contained. This approach ensures that functions do not rely on mutable state and do not cause any unintended side effects, leading to code that is more robust and easier to reason about. To further enhance the code's readability and maintainability, a class `Atom` was introduced as the only class in the project. The design of this class ensures that it does not contain mutable state and cannot be modified after its initialization. This immutability property simplifies the overall program flow and prevents unexpected modifications to critical data.
In order to maximize understandability and facilitate development, Python's type hints are employed throughout the codebase. By utilizing type hints for function parameters and return values, the intention of each component becomes clearer, allowing to quickly identify potential type mismatches and enabling the IDE to provide helpful suggestions and warnings. Consistent with good software engineering practices, each formula utilized in the project is implemented as a dedicated function. This approach ensures that every formula has its own well-documented function with clear input arguments and outputs. By encapsulating each formula in its own function, the code becomes more modular, maintainable, and easier to test and validate against known theoretical results. Furthermore, every function in the software is accompanied by a detailed docstring. These docstrings document the purpose of each function, as well as provide information about the expected input arguments and the output it produces.

3.1 main.py

The main.py script serves as the entry point for the software and orchestrates the execution of various studies related to the nuclear stopping power of colliding ions. The script begins by defining the interactions between

three instances of the Atom class, namely ^{1}H, ^{28}Si, and ^{197}Au, representing hydrogen, silicon, and gold isotopes, respectively. The interactions variable is a list of tuples that define the pairs of atoms involved in the interactions. Additional interactions can be added for further exploration and analysis by extending the list. Each study is performed by calling a corresponding function from the studies module, passing the interacting atoms as an argument. Additionally, the benchmarks are also invoked directly by the main script.

3.2 formulas.py

Contains the equations described in the assignment. Each formula has an accompanying docstring, which I suggest to check for detailed information. In this section I'll focus only on a selection of the most important functions.

screening_function(): It is a crucial function within the simulation as it is called most frequently, thus requiring optimization for efficiency. To achieve this, the function utilizes pre-calculated and cached values for α and β, reducing the need for repetitive calculations. Furthermore, to enhance performance, the function employs an explicit summation approach instead of list comprehension, as it has been observed to be faster in practice. Additionally, for added convenience and versatility, the function supports both scalar inputs and numpy arrays, allowing for efficient computation on individual values or bulk calculations.

coulomb_potential(): Some constant quantities have been cached globally for faster computation. Also, because the output of the formula described in the assignment is in Joules, the function converts the result to eV before returning it.

find_r_min(): Finds the distance of minimum approach r_{min} between two particles with a given impact parameter and center of mass energy, which is defined as the solution of the equation $g(r_{min}) = 0$. Because $g(r)$ contains a square root, it is inconvenient to use in conjunction with root-finding algorithms that are based on the bisection method, which requires that the lower and upper bounds of the search interval have opposite signs. Therefore, the function $g(r)^2$ is used instead, as it has the same zeros of $g(r)$ but does not result in a complex number for negative arguments. The chosen root finding function is `root_scalar()` from the SciPy library, which dynamically selects the most appropriate algorithm for the given function. `root_scalar()` requires either a valid interval that contains the zero, or the first derivative of the function. I've chosen to provide an interval, as it is easier to calculate and does not require any additional derivations. After a series of trial-and error experiments (see Section 4.2), it was determined that a valid lower bound for the search interval is $r_{left} = b/10$, and $r_{right} = 10/b$ when $b < 1$. When $b \geq 1$ a different value for the upper bound is used, namely $r_{right} = 10b$.

stopping_power(): Calculates the stopping power between two particles, considering the energy of the projectile in the lab frame and the maximum impact parameter b_{max}. The b_{max} parameter is an ad-hoc value that is determined by studying the behavior of the `stopping_power_integrand()` function. By analyzing the values of `stopping_power_integrand()` for different impact parameters, it is observed that the function's value drops exponentially becoming negligibly small for the energies and atomic interactions considered in the assignment. To choose b_{max}, I've found the value of b for which the function decreases below a threshold of $y = 10^{-12}$. Based on the studies shown in Section 4.4, $b_{\text{max}} = 10.6\,\text{Å}$ is chosen as a suitable threshold for the integration, ensuring accurate results while avoiding unnecessary computations for negligible contributions.

3.3 studies.py

The studies.py script provides functions for studying various properties related to atomic interactions. It uses the formulas.py module to visualize the behaviour of various functions and generate plots. Refer to the Results section for a complete description of the studies.

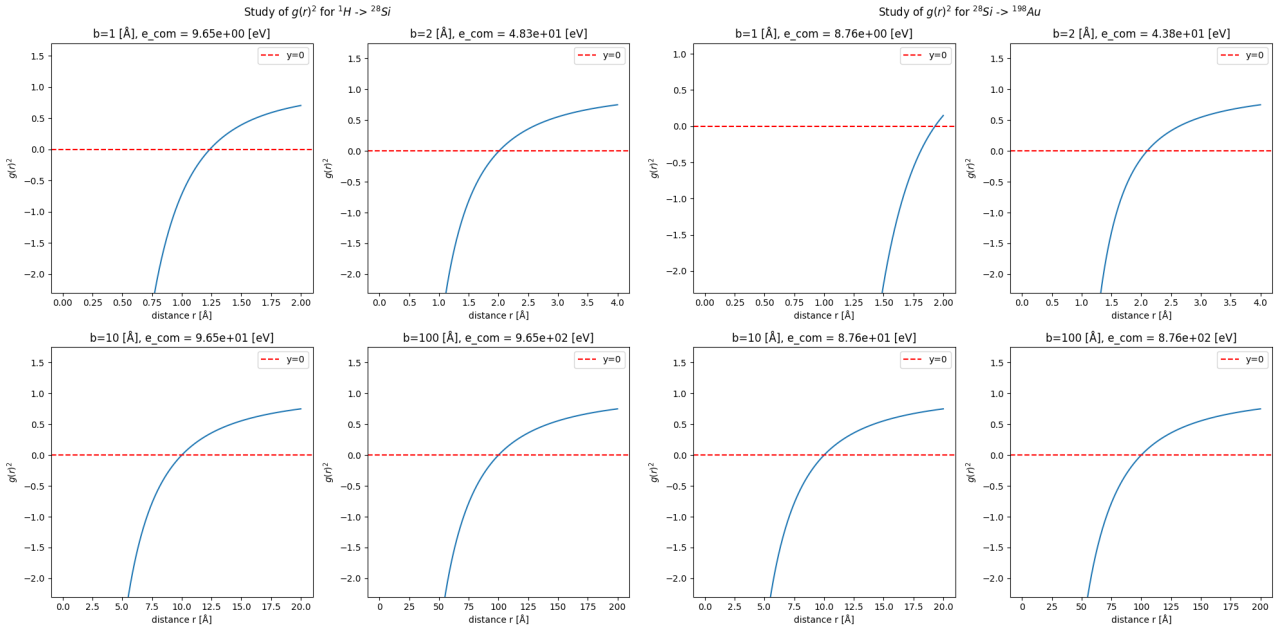
3.4 benchmarks.py

This script runs benchmarks for the most performance-critical functions in the project. It defines a dictionary of partial functions with pre-defined parameters, to simplify the benchmarking routine. Each function is executed multiple times, and the execution times are collected in an array. During the benchmarking process, some functions are tested with different parameters to assess if the parameter values have an impact on their performance. After running the benchmarks for all the functions, the results are printed, displaying the mean, standard deviation, minimum, and maximum execution times for each function. Additionally, the results are saved to a CSV file. See Figure 7 and Table 1 for the benchmark results.

4 Results

In this section I'll present the results of the studies performed with the software.

4.1 Study of g(r)



(a) $^1H \rightarrow^{28} Si$ (b) $^{28}Si \rightarrow^{197} Au$

Figure 1: Behaviour of $g(r)^2$ for different interactions

Plots the function $g(r)^2$ for chosen values of impact parameter b and center of mass energy E_{com}. The resulting plots helped with understanding the behavior of $g(r)^2$ and selecting the appropriate search interval for finding r_{min}, which requires solving $g(r_{min}) = 0$ numerically. We can see that the function is smooth and continuous, with a single zero for each value of b and E_{com}. Therefore, the root finding algorithm is guaranteed to converge to the correct solution given a good initial interval bound.

4.2 Study of r_min

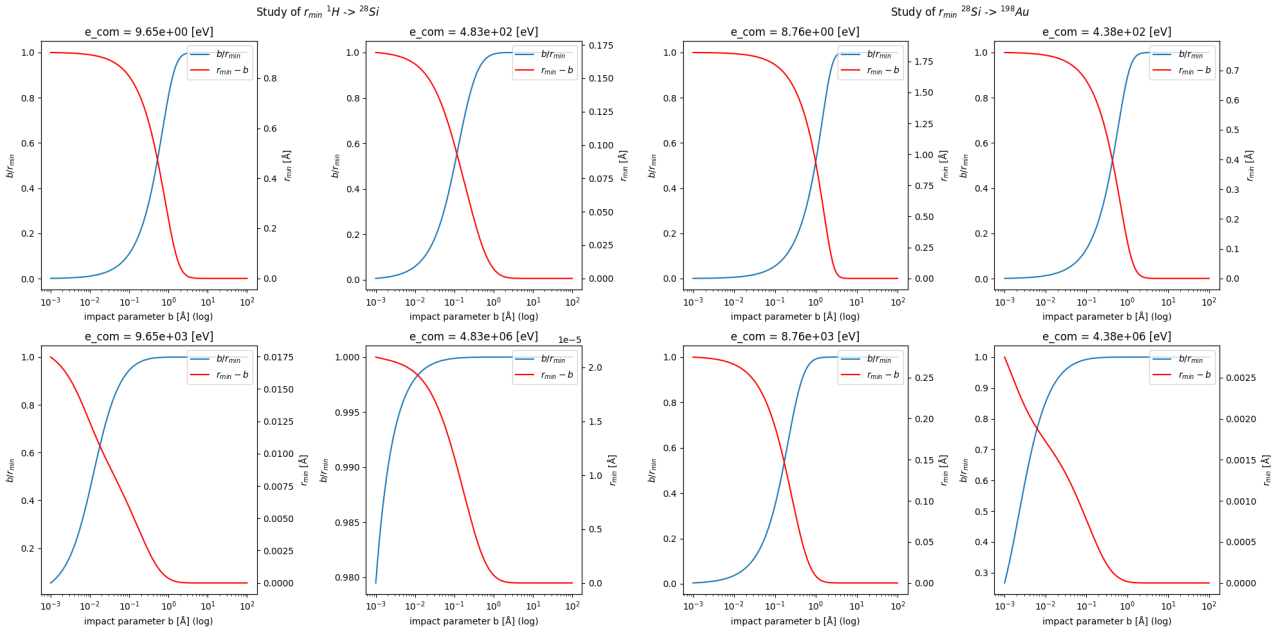
The distance of closest approach r_{min} is dependent on the impact parameter b, and so it is the range of values containing r_{min}; as we can see from Figure 2 the relationship is nonlinear. The purpose of this study was mainly to debug and troubleshoot the function `find_r_min()`, to have an immediate feedback for the correctness of the root finding algorithm and fine-tune the suitable range of possible search intervals. The tendency for r_{min} is to become increasingly closer to b as b and the system's energy increase. This means that the projectile particle is more likely to zoom past the target nucleus without much deflection. Instead, for small impact parameters and energies, the projectile particle will be more subject to the target's repulsion force, experiencing a larger deflection and a larger smallest distance.

4.3 Study of Scattering Angle

The scattering angle is defined by equation (11) and solved in function `scattering_angle()`. The observation made in Section 4.2 are reflected in the plotting of the scattering angle θ in Figure 3. As we can see, the scattering angle is close to 180° (a complete reverse of direction) for smaller impact parameters and energies, and it tends to zero as the impact parameter and energy increase.

4.4 Study of b_max

The integral in equation (8) needs to find a parameter b_{max} for which the potential contribution is negligible. This is done by finding the value of b_{max} for which the integrand is less than a given tolerance, arbitrarily chosen to be $y = 10^{-12}$. The plots in Figure 4 show the value of the integrand for different values of b and E_{lab}, with the



(a) $^{1}H \rightarrow^{28} Si$ (b) $^{28}Si \rightarrow^{197} Au$

Figure 2: Value of (b/r_{min}) in blue, and $(r_{min} - b)$ in red for different interactions

dotted line representing the value of b after which the condition is met. The greatest value found is $b_{max} = 10.6$, so it is set as default upper bound for the integration interval.

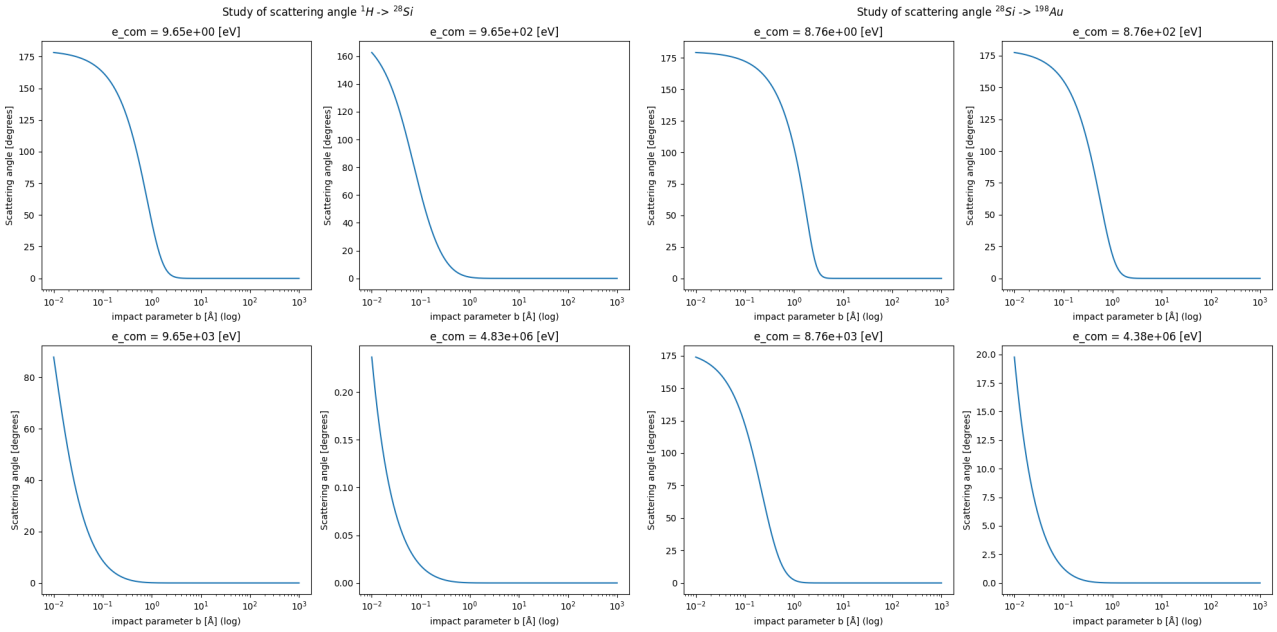
4.5 Nuclear Stopping Power

Finally, after all the necessary functions have been implemented, the stopping power $S_n()$ can be computed. The results are shown in Figure 5a, with the interactions $^{1}H \rightarrow^{28} Si$ and $^{28}Si \rightarrow^{197} Au$ visualized side by side. For Hydrogen - Silicon interaction, the stopping power peaks at around 0.4keV, with a stopping power of 5 eV/atoms/Å^2. For Silicon - Gold interaction, the stopping power peaks at around 54.8 keV, with a stopping power of 890 eV/atoms/Å^2. The stopping power is higher for the heavier projectile particle, as expected. Figure 5b shows the comparison with the universal formula for stopping power, which is given by equation (14). Both relative and absolute errors are computed and is shown in Figure 5c. Although visually the results look very similar, the relative error swings wildly reaching highs of 60% for $^{1}H \rightarrow^{28} Si$ at around 1100 keV, even though the absolute error is small. This is especially visible in the computed stopping power, there is a visible kink in Figure 5a that doesn't appear in Figure 5b. There seem to be a pattern in the error that is also visible in the bonus plots of Figure 6, as it looks very similar in shape and magnitude.

4.6 Benchmarks

Function	Mean (ms)	Std (ms)	Min/Max (ms)
screening_function_h	0.0040	0.0036	0.0026–0.0639
screening_function_l	0.0022	0.0015	0.0017–0.0219
coulomb_potential_h	0.0038	0.0018	0.0026–0.0331
coulomb_potential_l	0.0027	0.0005	0.0017–0.0060
g_squared_hh	0.0039	0.0010	0.0026–0.0217
g_squared_hl	0.0047	0.0033	0.0026–0.0362
g_squared_lh	0.0033	0.0027	0.0017–0.0269
g_squared_ll	0.0039	0.0039	0.0017–0.0601
universal_stopping_power	0.0034	0.0031	0.0017–0.0548
find_r_min	0.0803	0.0305	0.0639–0.3760
scattering_angle	1.1289	1.9344	0.9253–44.2441
stopping_power	48.0180	3.0685	43.5612–72.8471

Table 1: Benchmark Results



(a) $^{1}H \rightarrow^{28} Si$ (b) $^{28}Si \rightarrow^{197} Au$

Figure 3: Scattering angle θ in degrees for different interactions

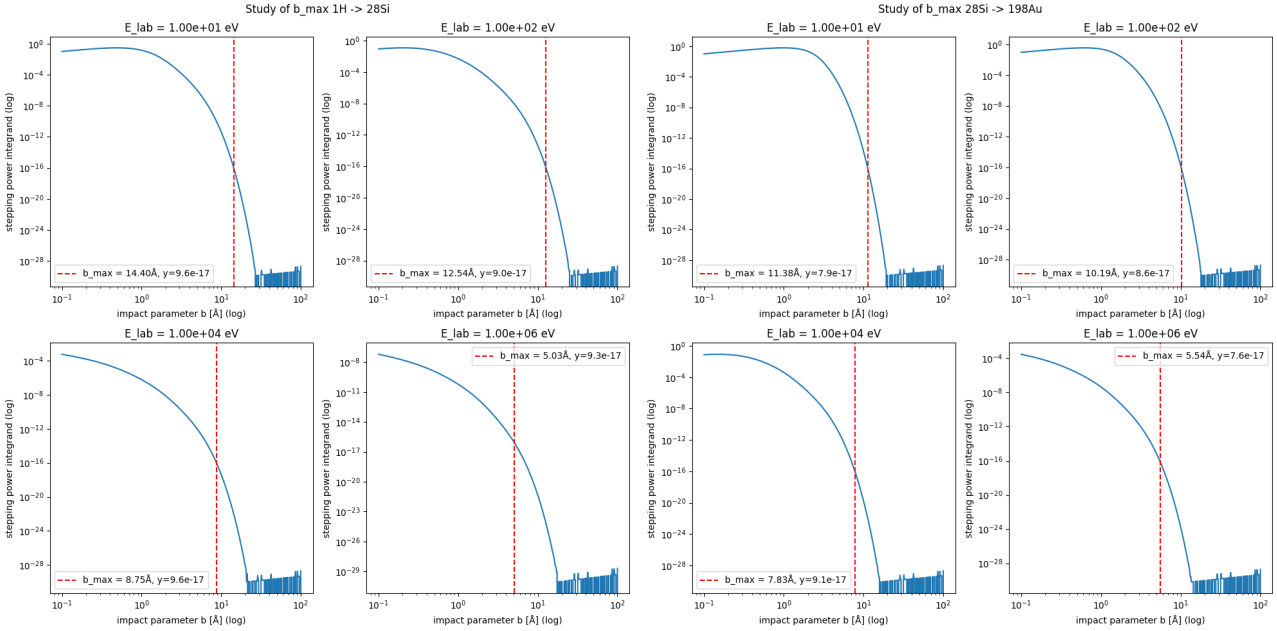
The benchmarks were run on a 2016 MacBook Pro with a 2.7 GHz Intel Core i7 processor and 16 GB of RAM. The results are shown in Figure 7 and Table 1. The functions that take the longest to run are `stopping_power()` and `scattering_angle()`, which are the functions that perform the numerical integration, and call all the other functions as well. Also the function `find_r_min()`, which runs the root finding algorithm, is computationally expensive, even not as much as the other two. All other functions only execute simple mathematical operations, and they're very performant (notice that the y axis is in logarithmic scale). It's interesting to notice that there seem a small, yet noticeable, difference in performance when `screening_function()` is called with parameters having high value (with suffix "_h") or low value (prefix "_l").

5 Conclusions

The simulation results of the nuclear stopping power demonstrate a reasonable alignment with the expected values computed using the universal stopping power formula. However, some discrepancies between the computed results and expected values were encountered, which require further investigation and troubleshooting. These discrepancies are clearly illustrated in Figure 5c, highlighting the need for additional analysis and refinement of the simulation. Furthermore, it is worth noting that the current implementation of the root finding algorithm in the `find_r_min()` function utilizes a fixed ad-hoc interval. Ideally, the selection of the interval should be optimized based on the specific parameters b and E_{com}. This optimization will surely enhance its performance, speeding up others methods that depend on it. Similarly, the b_{max} parameter, which represents the upper limit for numerical integration, could benefit from an optimal selection based on the input parameters. By dynamically adjusting b_{max} according to the specific input parameters, the numerical integration process would not only be more precise, but also be further expedited, leading to improved efficiency.
In conclusion, there are areas that warrant further exploration and optimization. By addressing the discrepancies observed, refining the root finding algorithm's interval selection, and optimizing the b_{max} parameter, future iterations of the simulation can achieve more accurate and efficient results.

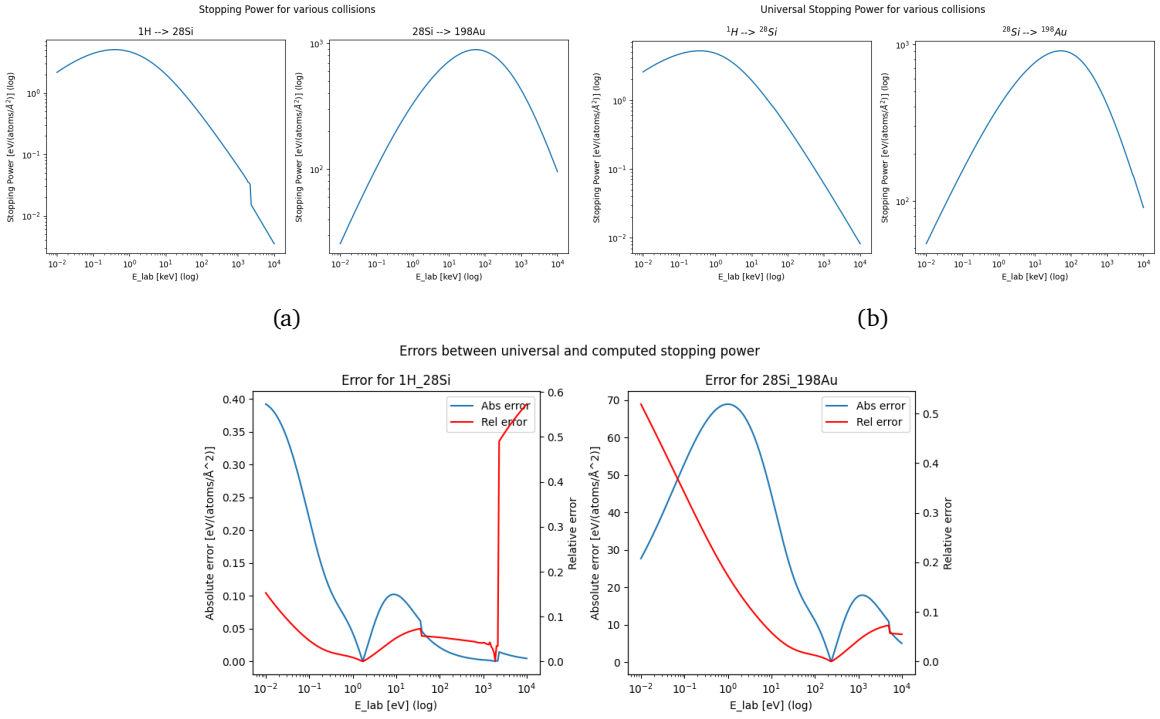
References

[1] Wikipedia article on stopping power: http://en.wikipedia.org/wiki/Stopping_power_(particle_radiation)



(a) ${}^{1}H \rightarrow {}^{28}Si$ (b) ${}^{28}Si \rightarrow {}^{197}Au$

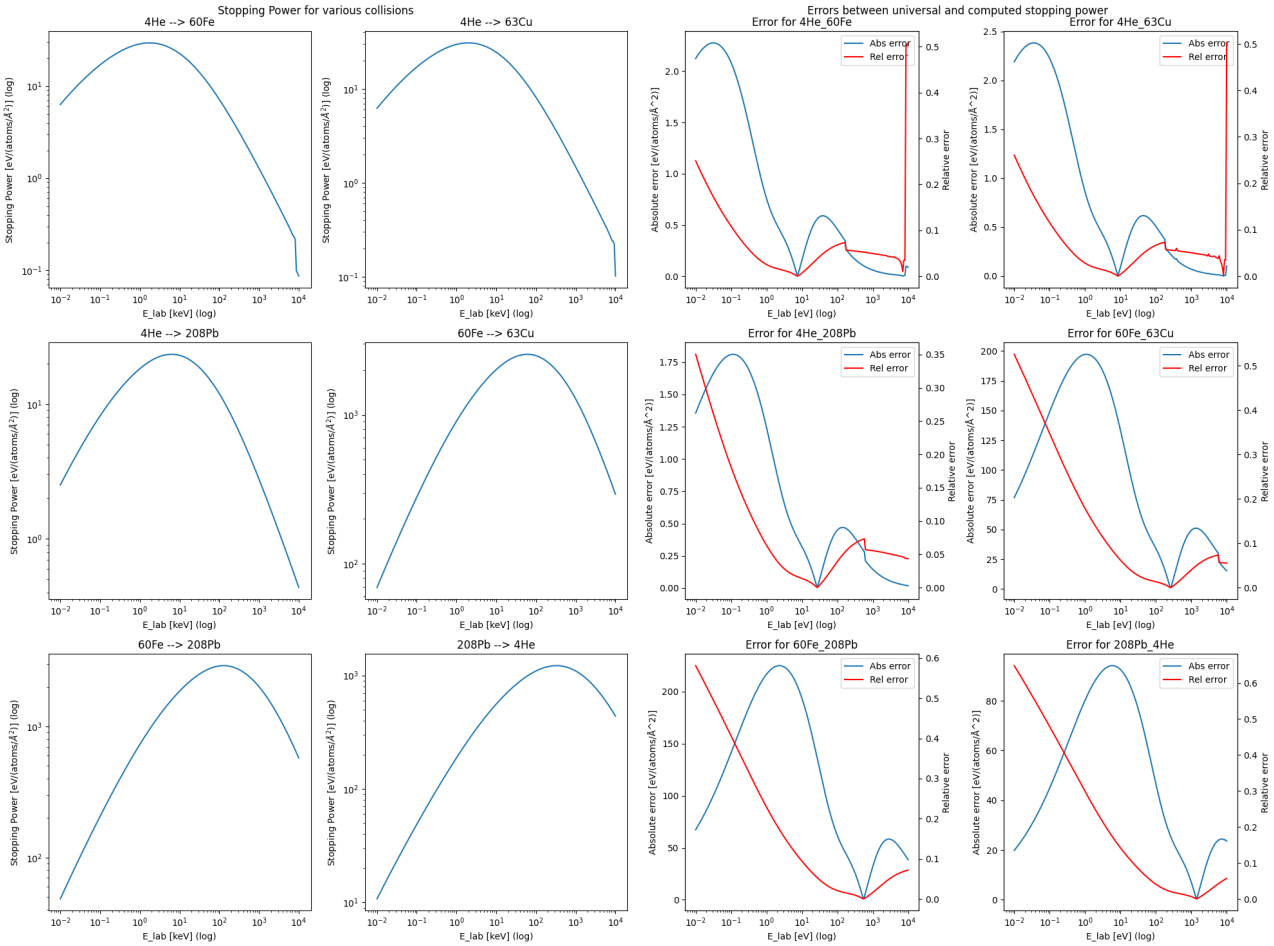
Figure 4: Integrand of (8) and corresponding b_{max} value for different interactions



(a) (b)

(c)

Figure 5: Comparison of computed stopping power with universal formula



(a) Bonus stopping powers

(b) Errors compared to universal formula

Figure 6: Bonus plots: other atomic Interactions

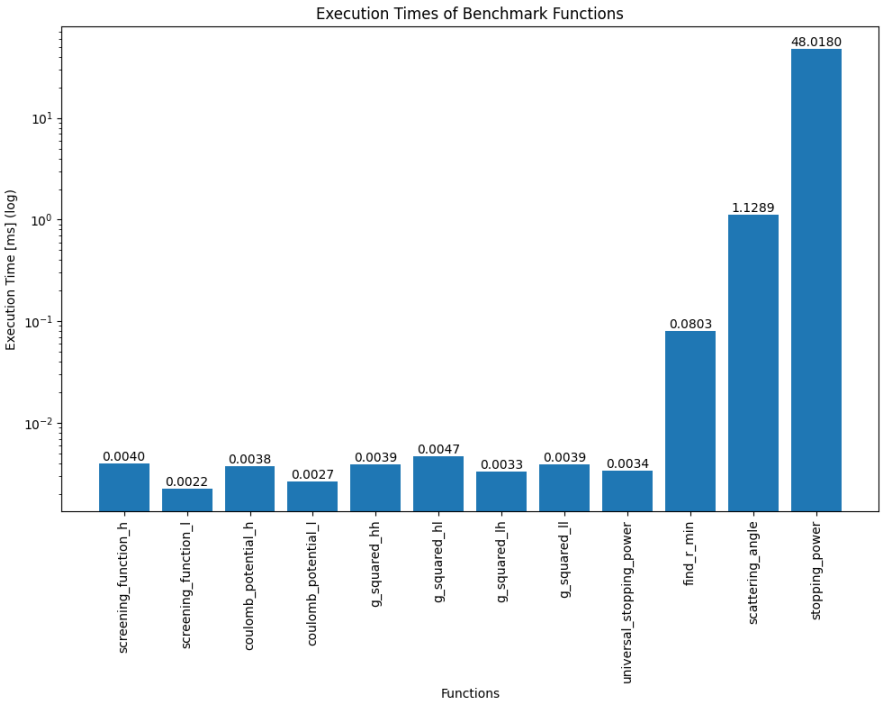


Figure 7: Benchmark Execution Times