Lepton-Catalyzed nuclear fusion: Calibration and Integration of Facilities for Fusion Product Measurements

Isaac Willden

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

John Ellsworth, Advisor

Department of Physics and Astronomy

Brigham Young University

ABSTRACT

Lepton-Catalyzed nuclear fusion: Calibration and Integration
of Facilities for Fusion Product Measurements

Isaac Willden
Department of Physics and Astronomy, BYU
Bachelor of Science

Muon-catalyzed fusion occurs when a muon replaces an electron in a hydrogen molecule, which reduces the Bohr radius of one of the atoms, increasing the probability of fusion. Past work shows that muons were ejected after catalyzing p-d fusion, carrying ~5.5 MeV—near the Q-value of the reaction. After observing this phenomenon, Jones hypothesized in 1986 that electrons could catalyze such reactions as well. Other work demonstrates this with a proton beam on deuterated graphite, finding ~5.5 MeV electrons, similar to the muons. A facility has been constructed to investigate the effects of a 4 keV deuteron beam onto a titanium hydride target. This thesis discusses the equipment built to detect fusion products from these reactions and the calibration methods used. After preliminary testing, it was determined that a higher deuteron flux would be necessary to increase the likelihood of experimentally observing fusion events. To address this, a higher-current ion source has been designed, is under development, and is nearly ready for testing.

Keywords: nuclear, fusion, instruments, spectrometer, neutron, gamma, ΔE-E, electron, spectroscopy, hydrogen, deuterium, lepton-catalyzed, muon-catalyzed, electron-catalyzed

ACKNOWLEDGMENTS

I am immensely grateful for my amazing wife, Janet. She has been so patient and loving to me during this phase in my studies.

I am grateful for the Brigham Young University College of Physical Mathematical Sciences and the Department of Physics and Astronomy for giving me the position as a Research Assistant and funding for research materials in support of this work.

I express gratitude for my advisor, John Ellsworth, who has dealt with my ignorance and taught me powerful lessons, both for life as well as for research. I'm grateful for my previous advisor, Dr. Scott Bergeson, who helped me get started in physics research and gain confidence in myself, despite my inexperience.

I'm grateful for my collaborators, Rhett Lundell, Tyler Hamm, and Trent Angell, who have supported me in this research.

Most importantly, I express gratitude and praise to my God, who has created a universe of endless wonder that I endeavor to understand.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Before physicists proposed nuclear fusion as the Sun's method of generating energy, a common idea was that the Sun was a burning ball of coal [1]. The concern with this was, with the Sun's size and luminosity, it should only last for a few thousand years. Geological evidence clearly pointed to a sun that was at least millions of years old. In 1920, Arthur Eddington postulated that the Sun's energy is produced by nuclear fusion. [2] This explanation was incomplete, since the energy required to classically overcome the Coulomb barrier for p-p fusion is hundreds of keV. This corresponds to billions of degrees, which is orders of magnitude greater than the Sun's actual temperature of 15.8 million Kelvin (on the order of 1 keV). [3] [4].

To help solve this discrepancy, Edwin Salpeter later proposed that the Coulomb barrier may be reduced by electron screening in stellar plasmas. [6] This is demonstrated in figure 1.1. This partially accounts for the energy discrepancy by lowering the effective Coulomb barrier, allowing fusion to occur at lower energies. Decades earlier, George Gamow also shed light on stellar fusion by applying quantum physics, producing the Gamow factor (see figure 1.2). [7] This factor is the probability that a nucleus will quantum tunnel through the Coulomb barrier and fuse with another nucleus, releasing energy. This bypasses the classical barrier, further explaining how the Sun is able to generate its energy.

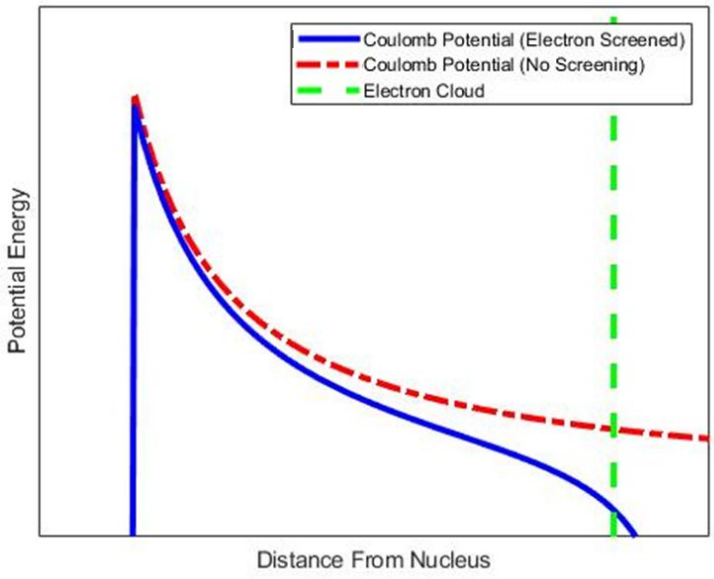

Figure 1.1 Qualitative demonstration of Coulomb screening. Note the difference in height on the right side of the potential when electrons are present, screening the nuclei. (Based on Rolf's figure 4.8 [5])

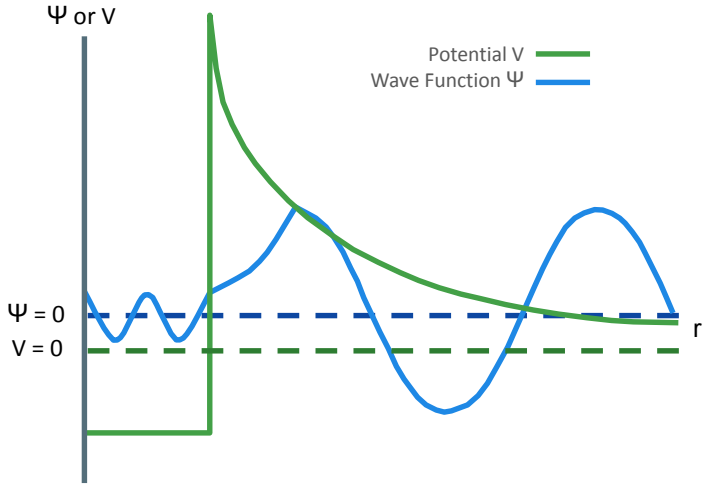

Figure 1.2 Demonstration of quantum tunneling through the Coulomb barrier. Note the exponential decay within the barrier and the attenuated wave function after the barrier, indicating a small but nonzero probability of tunneling. (Based on image found on University of Sheffield's teaching webpage [8])

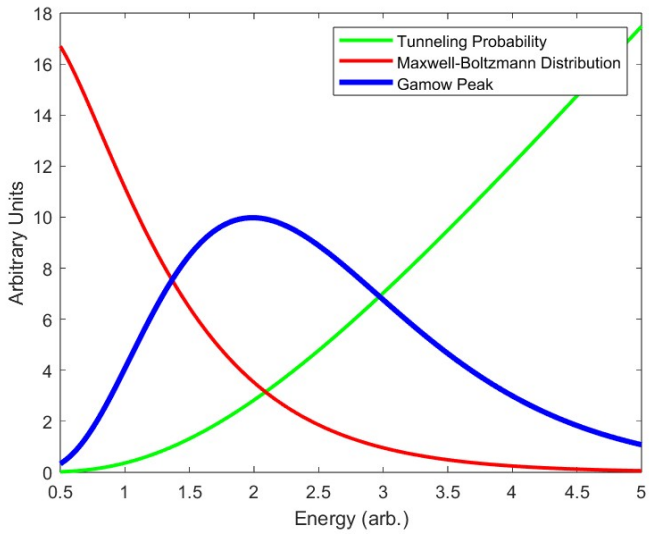

Figure 1.3 A MATLAB demonstration of the Gamow peak with arbitrary units. It is the combination (multiplication) of the Maxwell-Boltzmann distribution and the tunneling probability (Gamow factor) as functions of energy. Note that the maximum of the Gamow peak is the most probable energy for fusion to occur in the system (also see Rolf's figure 4.6 [5]).

Hans Bethe expanded on this idea in 1939. Bethe took the Maxwell-Boltzman energy distribution for the Sun, combined it with the Gamow tunneling factor, and created a model for the reaction probability as a function of particle energy (see figure 1.3) [9]. This gives a distinct peak in probability that is significantly above the average particle energy. The majority of fusion events in the Sun occur at this energy, known as the Gamow peak. Below this peak energy, particle density is high, but tunneling probability is too low for frequent reactions. At higher energies, the tunneling probability is higher, but there are too few particles for a significant reaction rate.

1.1 Lepton-Catalyzed Fusion

In a separate vein, Frank was the first to propose muon-catalyzed fusion (μCF) in 1947 after observing unexplained reactions in radiographic emulsion [10]. Muons have the same charge as

electrons but about 207 times the mass, so muons form smaller orbitals around nuclei, allowing the nuclei to come closer together. This boosts the tunneling probability, even at low energies. Alvarez was the first to pursue this experimentally in 1957. [11] After catalyzing p-d fusion reactions, the muon is ejected instead of the expected gamma ray at 5.5 MeV—the Q-value of the reaction (see the first reaction in table 1.1). The ejection of high-energy leptons is the evidence this work is looking for.

With this foundation crafted by his predecessors, Steven Jones explored muon-catalyzed fusion [12]. After observing μCF, he conjectured that since muons and electrons both belong to the lepton family, electrons might be able to catalyze fusion also. A few experiments involving nuclear fusion have measured high-energy particles that disagree with theoretical predictions, but may be explained by lepton-catalyzed fusion.

M. Lipoglavšek's team investigated this catalyzation in 2017 when they studied proton-deuteron fusion using a 260 keV proton beam [13]. They implanted graphite with deuterium and observed not only 5.6 MeV gamma rays, but 5.6 MeV electrons. These electrons may occur when an excited nucleus interacts with a nearby electron, transferring its energy through the electron instead of through gamma emission. Lipoglavšek presented this as evidence for electron-catalyzed fusion in these p-d reactions.

Later, in 2024, Czerski et. al. obtained adjacent results when investigating the deuteron-deuteron fusion reaction [14]. Using a 6–16 keV deuteron beam at 40 μA on a ZrD2 target, they measured an energy spectrum in a silicon detector and found the results consistent with a 0–23 MeV electron/positron source spectrum. This maximum of 23 MeV is approximately the Q-value of the $D(d,\gamma)^4He$ fusion reaction (see the fourth reaction in table 1.1).

In another experiment, Forbes' team loaded titanium foils with deuterium, then exposed the foils to a 950 eV argon ion beam [15]. The team detected some charged particles from 32–40 MeV

and several counts below 11 MeV. The origin and energies of these particles are still uncertain and not demonstrated by current models.

Holmlid and Olafsson later reported high-energy charged particles after irradiating hydrogen-loaded targets with a nanosecond laser [16]. Based on time-of-flight and decay measurements, they interpreted the signals as muons produced in laser-driven nuclear processes. The reported particle energies were much higher than the energy of the incident laser pulse. These results and interpretations have been disputed, with other authors claiming the signals are instrumental artifacts and not evidence of muons [17]. In any case, if the results are legitimate, this experiment may still be evidence for anomalous high-energy nuclear events.

In summary, various researchers over the last century have found that fusion rates at low energies are significantly higher than predicted estimates and that some fusion experiments yield unexpected high-energy products. The hypothesis of this work is that electron-catalyzed fusion is one of the reasons for this anomaly. This thesis seeks to examine this hypothesis, which may lead to important insights into the nature of fusion reactions, particularly the reactions within the Sun.

1.2 Experimental Goals

Past work demonstrates anomalous fusion yields at low energies, especially in conductive environments where electrons are present [18]. The goal of this work is to build on these discoveries to gain further understanding of these fusion processes.

There are 31 models for the atomic nucleus and their reactions [19], but few work well for nuclei of atomic mass less than 20 amu. Accurately predicting fusion rates of light nuclei at low energies remains challenging. These reactions are often studied at very high temperatures, such as in tokamak reactors, but this work seeks to investigate these reactions at low energies.

These experiments will explore the reactions of proton-deuterium fusion using a 4 keV deuteron beam onto a titanium hydride (TiH_x). This is the $H(d,\gamma)^3He$ reaction. The proton and deuteron

should combine, producing Helium-3 and emitting a ~5.5 MeV gamma ray that can be measured. These experiments will search for ~5.5 MeV electrons similar to what Lipoglavšek observed. [13] If observed, such electrons could be further evidence for electron-catalyzed fusion.

Experiments will also explore deuterium-deuterium fusion using a similar setup with TiD_x instead of TiH_x. These experiments are expected to observe high-energy neutrons, protons, and gamma rays (table 1.1 lists the possible fusion branches and energies for these reactions). Additionally, if high-energy electrons are observed, further work will investigate their energy spectrum, rates, and potential origins. If such electrons are consistently found at ~24 MeV, it may be additional evidence of catalysis.

Reaction	**Shorthand**	**Ratio**	Q **(MeV)**	**Product Energies (MeV)**
$^2\mathrm{H} + ^1\mathrm{H} \rightarrow \gamma + ^3\mathrm{He}$	$\mathrm{H(d,\gamma)^3He}$	~ 1	5.494	γ: ~ 5.5
$^2\mathrm{H} + ^2\mathrm{H} \rightarrow p + ^3\mathrm{H}$	D(d,p)T	~ 0.5	4.033	T: 1.01, p: 3.02
$^2\mathrm{H} + ^2\mathrm{H} \rightarrow n + ^3\mathrm{He}$	$\mathrm{D(d,n)^3He}$	~ 0.5	3.269	^{3}He : 0.82, n: 2.45
$^2\mathrm{H} + ^2\mathrm{H} \rightarrow \gamma + ^4\mathrm{He}$	$\mathrm{D(d,\gamma)^4He}$	$\sim 10^{-5}$–10^{-6}	23.846	γ: ~ 23.9

Table 1.1 Fusion reactions and their approximate branching ratios, total Q-values, and product energies for p–d and d–d fusion. For the first and last rows, due to conservation of momentum, nearly all the energy is transferred to the gamma rays.

In order to measure these fusion products, the facilities include several spectrometers that can run simultaneously. During an experiment, each detector sends signals to an eight-channel CAEN digitizer operating at 250 MS/s with 12-bit resolution [20]. When the CAEN receives a signal from a detector above a programmed voltage threshold, it is triggered and digitizes the event. These events are stored and processed on the computer.

These facilities can be repurposed for a wide variety of future experiments. The diverse selection of detectors enables measurement of various nuclear products, with energy ranges (MeV

scale) sufficient for many nuclear reactions. Additionally, the flexible chamber design allows for relatively rapid adjustments to accommodate experimental needs.

This thesis discusses the setup and calibration of these detectors for measuring particles and determining their energies. Each detector is designed to measure a different type of particle. The main detectors are the high-energy Electron Spectrometer, gamma ray spectrometer, ΔE-E charged-particle spectrometer, and Neutron Pseudo-Spectrometer. A large plastic scintillator (the Veto) sits above the chamber for detecting background events.

The Electron Spectrometer is especially important, since it is designed to search for electrons ejected from p-d or d-d fusion. If high-energy electrons are measured, their energies will be analyzed to determine if they are consistent with the electron-catalyzed fusion mechanism. The gamma-ray spectrometer's precision will help determine how much fusion is occurring in the p-d and d-d experiments. The ΔE-E charged-particle spectrometer is for heavier charged particles and can distinguish between them, such as alphas, protons, and tritons. This is important for verifying fusion, since d-d fusion should produce protons and tritons. The Neutron Pseudo-Spectrometer, though not calibrated to measure energies, is also useful for verifying and measuring fusion rates for d-d fusion since about 50% of reactions produce a neutron.

Outlined in Chapter 2, this thesis discusses the calibration of the ΔE-E, electron, and gamma spectrometers using various radioactive samples that emit particles of known energies. Chapter 2 also discusses the Graphical User Interface (GUI) in Octave designed for analyzing and displaying experimental data for all detectors during and after experiments. Chapter 3 summarizes and concludes these results.

Chapter 2

Instrument Setup

To investigate electron-catalyzed fusion, there is a wide array of spectrometers. Gamma, neutron, charged-particle, and high-energy electron detectors are arranged as shown in figures 2.1, 2.2, and 2.3. A basic CAD rendering of the setup is shown in figure 2.4.

An 8-channel CAEN waveform digitizer collects data from each detector. It has an input voltage from -1 to 1 V, which is digitized as a 12-bit number from -2047 to 2047. Like an oscilloscope, this device constantly scans the voltage. When it detects a voltage above a set threshold, the device is triggered and saves 8192 data points, each 4 nanoseconds apart.

The detectors often measure a Gaussian-like pulse or a jump with an exponentially decaying tail. The height of this pulse is often proportional to the energy of the event. Calibration involves converting this pulse height to a usable energy in keV or MeV. The most common method is to take a radioactive source that emits particles of a known energy, then use a linear relationship to convert from voltage to energy.

All detectors point to the center of the metal vacuum chamber (fig. 2.1) to measure the radiation sources for calibration or fusion products from the ion beam experiments. The chamber is electrically grounded, but to measure the ion current on the target, the target holder (figures 2.2 and 2.3) is isolated from ground and connected to an electrometer. The target is attached to a Conflat

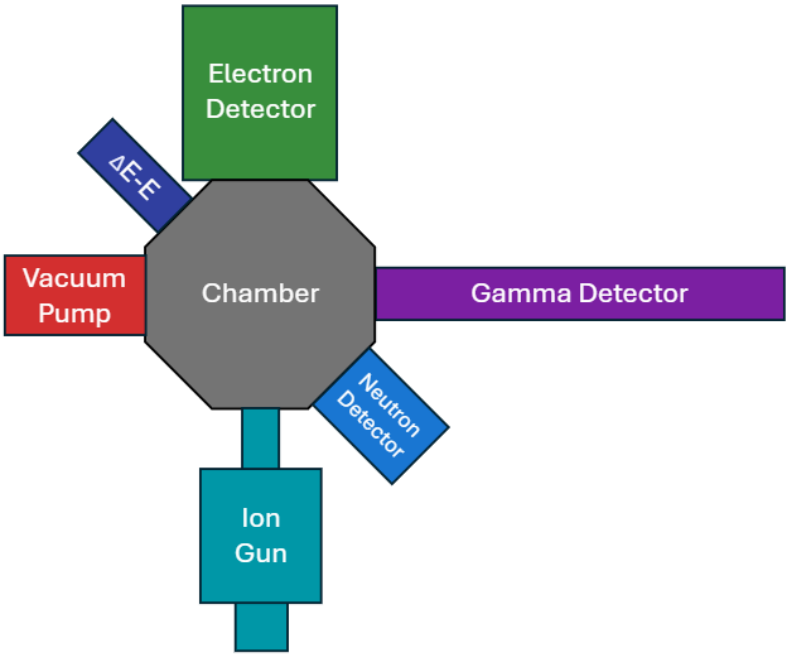

Figure 2.1 General birds-eye diagram of the chamber and detector setup. The target is placed in the center of the chamber under microtorr pressures. Physically above this setup is the veto panel (not shown), for measuring cosmic background noise.

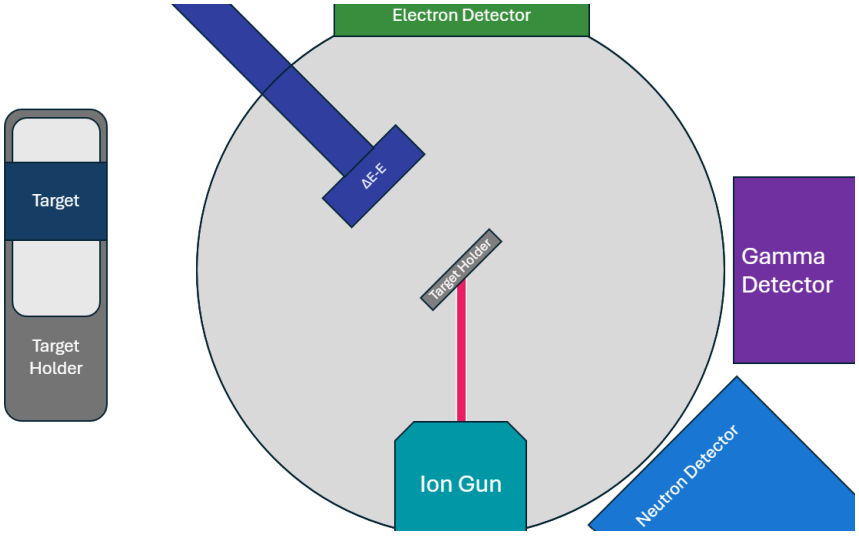

Figure 2.2 Closer look at the target holder and spectrometer setup in the chamber. The target can be rotated to any angle needed. Note the electron detector and $\Delta E - E$ are inside the chamber and under vacuum, while the gamma, neutron, and veto detectors are outside the chamber.

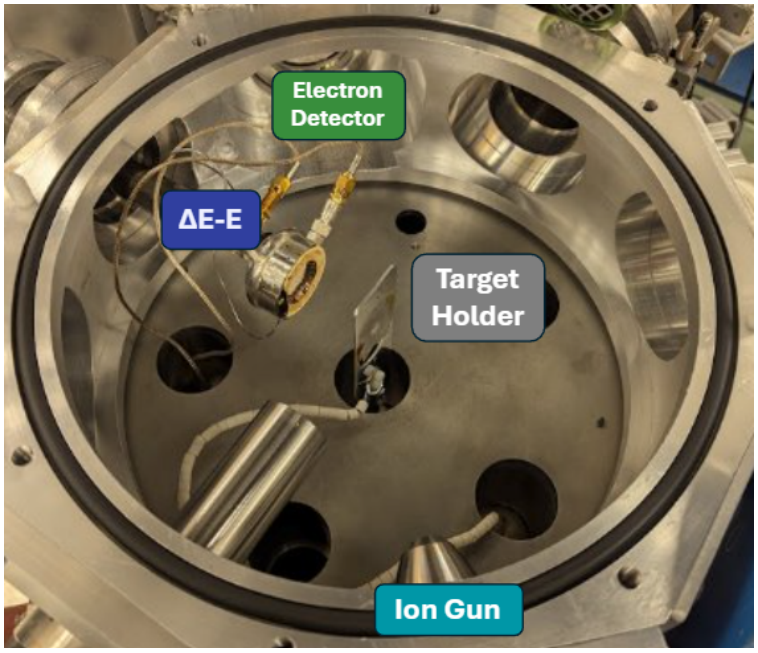

Figure 2.3 Picture of the inside of the chamber with a few components labeled. Note the beaded wire from the target holder. This goes to the electrometer to measure the beam current on the target.

(CF) rotary feed-through, allowing one to change the angle of the target even if the chamber is under vacuum.

To reach vacuum pressure, the chamber utilizes a Varian 300DS scroll roughing pump and a Pfeiffer High Pace 300 turbo pump. The chamber has an MKS 925 Pirani gauge and a Varian UHV24 ion gauge to measure the pressure in the chamber. The chamber consistently pumps down to the μTorr range during testing.

Table 2.1 includes a list of the main components of the experimental setup, including their dimensions and materials.

This chapter discusses each calibrated detector, including the methods used to calibrate them, the results, and some discussion. Afterwards, this chapter discusses the other equipment used in the facilities and the unified GUI developed for data processing.

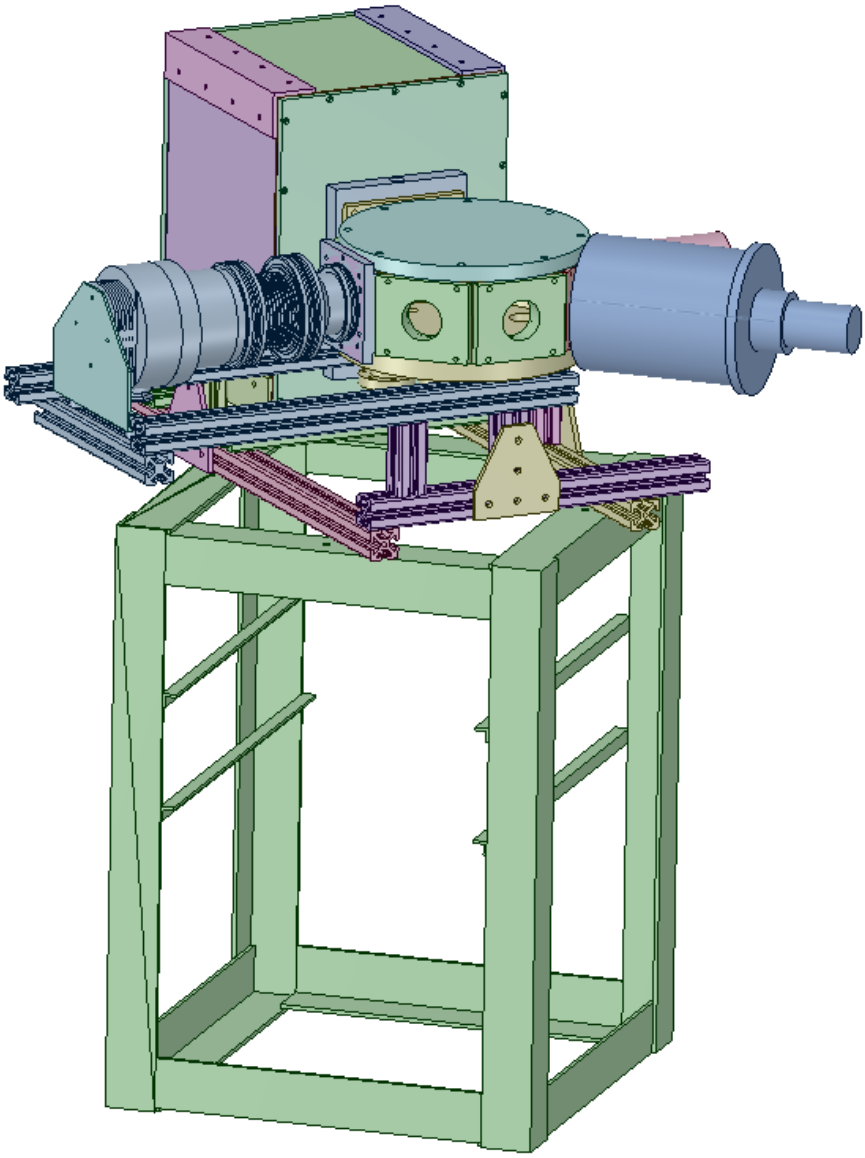

Figure 2.4 CAD rendering of the main chamber setup. Note the turbo pump on the left, the electron detector in the back, the neutron detector on the right in light blue, and the gamma detector partly hidden behind that. The ion gun and veto panel are not shown here. The STEP files for this had to be recovered from older unusable files. See appendix I for details on this file recovery.

Component	Dimensions / Geometry	Material / Type
Vacuum Chamber	30.5 cm diameter, 10.5 cm depth	Aluminum
Veto Spectrometer	2000 cm^2 area, 37.5 cm from target	Plastic scintillator
Neutron Pseudo-Spectrometer	16.5 cm diameter, 23 cm length	Polyvinyl toluene (PVT) with Li-Gd-Borate (LGB)
Gamma Spectrometer	8 cm diameter, 66 cm length	high-purity germanium (HPGe)
ΔE-E Spectrometer	300 mm^2 active area, 8 mm apart; ΔE: 26 μm thick; E: 130 μm thick	Two silicon SBDs, 13 V reverse bias
Electron Spectrometer	2000 mm^2 active area, 300 μm thick; and 10 liter PVT scintillator [21]	Silicon detector and PVT scintillator
Target	1 × 1 in, 45° from beam	Titanium

Table 2.1 The primary components of the experimental setup with their dimensions and materials.

2.1 High-Energy Electron Spectrometer

Measuring high-energy electrons is essential for identifying electrons potentially ejected by electron-catalyzed fusion. This would demonstrate electron catalysis if the detected electrons have similar energies to the gamma rays from p-d or d-d fusion.

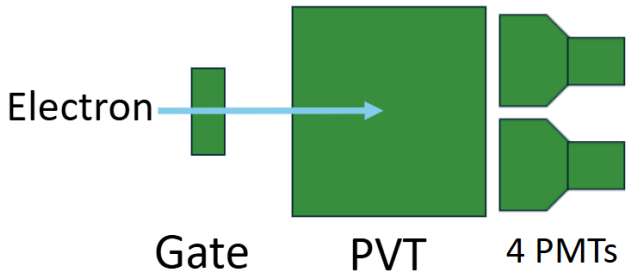

Figure 2.5 Simple diagram of the Electron Spectrometer. The electron enters the SBD gate before the PVT scintillator. The PMTs then record the pulse of light emitted in the PVT

The Electron Spectrometer can measure such high-energy electrons. This device has two main components: a silicon surface barrier detector (SBD) gate and a 10-liter polyvinyl toluene (PVT) scintillator (Eljen model EJ200) [21]. The SBD gate serves as a trigger device to filter out gamma rays and heavy charged particles. Since this gate is inefficient for detecting gamma rays and it absorbs heavy charged particles, high-energy electrons are expected to be the only particles that are both detected by and pass through the gate. After the electron passes through the gate, it enters the PVT scintillator, which absorbs the electrons, emitting light that is measured by four photomultiplier tubes (PMTs). The PVT scintillator will also emit light when other particles enter, such as gamma rays, but the gate allows the system to ignore these events. This is portrayed in figure 2.5.

The two signals from the gate and the PMTs go through delay amplifiers and a coincidence gate to ensure that only PMT pulses from electrons make it through to the digitizer (see appendix B for the block diagram and code).

2.1.1 Methods

This spectrometer was calibrated with Cs-137, which has a conversion electron peak at 624 keV [22]. To increase the instrument range, it was also calibrated with a Y-90 source that emits

electrons at energies from 0 to 2.28 MeV [23]. Using the spectra from these sources, one can find the voltage channel corresponding to 2.28 MeV. The calibration factor is therefore 2.28 divided by the corresponding voltage channel.

The uncertainty in the Y-90 calibration factor was found using a Kurie plot. This takes the spectrum N(E) and normalizes it according to equation 2.1, where p is the momentum of a beta particle, $F(Z,E)$ is the Fermi function which accounts for the coulomb force between the beta particle and the nucleus, and K(E) is the normalized beta spectrum. In this case, $F(Z,E) \approx 1$ works as a reasonable approximation for calibration purposes. The result linearizes the end of the spectrum, allowing one to plot a line of best fit (see figure 2.6). The x-intercept of this line then becomes the endpoint of the spectrum. Using the uncertainty in the line slope and intercept, one can estimate the uncertainty of the position (see appendix B for the code used). The result is a voltage channel that can be used for the calibration factor and the corresponding uncertainty values.

$$K(E) = \sqrt{\frac{N(E)}{p^2 F(Z,E)}} \tag{2.1}$$

2.1.2 Results

Figures 2.7 and 2.8 show the calibration results with energies in MeV. In the Y-90 data, the spectrum fell off at around channel 165 ± 5, which corresponds to 2.28 MeV. Therefore, the calibration factor is $2.28/165 \approx (1.38 \pm 0.1) \times 10^{-2}$ MeV/channel.

2.1.3 Discussion

Data from both Cs-137 and Y-90 electron sources establishes the calibration factor for the high-energy Electron Spectrometer. The high energy of the spectrum from Y-90 leads to a relatively

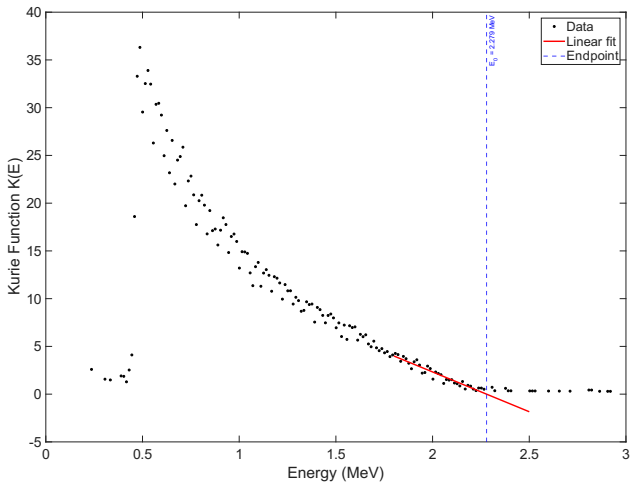

Figure 2.6 Kurie plot for the Electron Spectrometer calibration data. Note that the plot is fairly linear towards the tail end of the spectrum. The point where the line of best fit crosses the x-axis is the point determined to be the end of the spectrum. This was plotted using the optimized calibration channel of 165, with the end of the spectrum found to be 2.279 MeV.

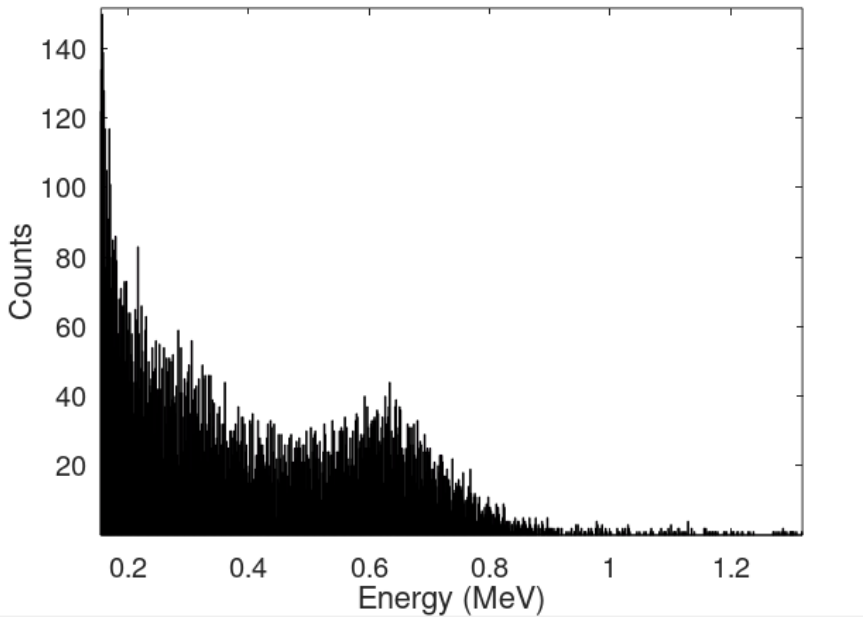

Figure 2.7 The Cs-137 electron spectrum measured by the high-energy Electron Spectrometer. The peak on the right was used for calibration.

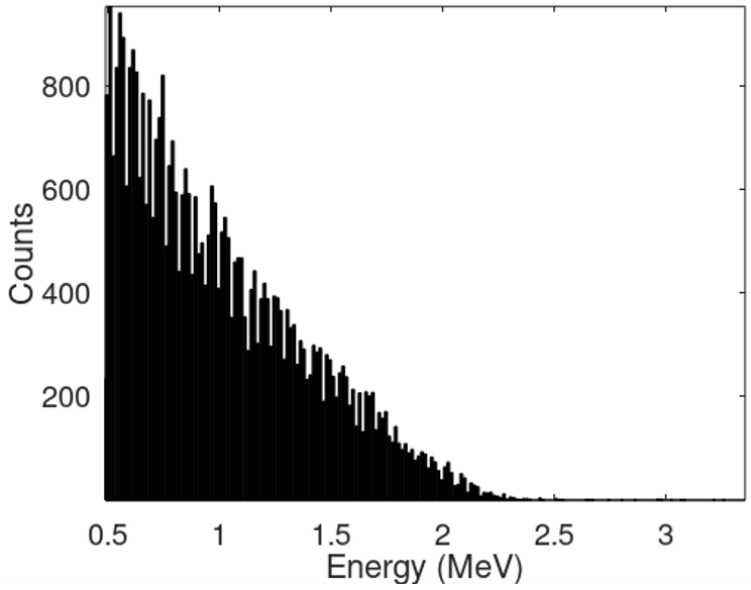

Figure 2.8 Electron spectrometer calibration data from Y-90. The spectrum falls off to 0 around 2.28 MeV.

wide range for the spectrometer. With the current setup, it can detect energies up to 30 MeV, well above the gamma energies from both p-d and d-d fusion. This means it should be able to detect any electrons ejected after catalyzing fusion reactions.

There is some underlying uncertainty in this method, both in the binning and in determining which channel corresponds to the end of the spectrum at 2.28 MeV. Also, since there was no calibration source of electrons higher than 2.28 MeV, it is uncertain if the calibration factor holds at higher energies. Despite the uncertainties, this device can reasonably detect electrons, distinguish them from other particles, and determine their approximate energy with an uncertainty of ˜0.16 MeV. This detection capability is critical for experiments to find the ejected electrons, much like it was for Lipoglavšek [13].

2.2 Gamma Spectrometer

A high-purity germanium spectrometer sits outside the chamber for gamma spectroscopy. To minimize noise and prevent damage, the Gamma Spectrometer needs to be cooled via conduction through a copper rod in liquid nitrogen for 24 hours before operation. This spectrometer is important for measuring the gamma rays from p-d or d-d fusion to verify that fusion is occurring in the chamber and to estimate the fusion rates.

2.2.1 Methods

Cobalt-60 is a suitable calibration sample since it has two peaks in its gamma energy spectrum: 1.17 MeV and 1.33 MeV [24]. These two points make it possible to collect data and find the m (slope) and b (y-intercept) calibration parameters for the spectrometer. The equations in 2.2 give these values, where P_1 and P_2 are the found peaks from the channel data. Using the last equation from 2.2, where E_{ch} is the energy of an event in channels and E_{MeV} is the energy in MeV, one can convert from channels to MeV.

$$m = \frac{1.33 - 1.17}{P_2 - P_1}, \qquad b = 1.17 - m \cdot P_1, \qquad E_{\mathrm{MeV}} = m \cdot E_{\mathrm{ch}} + b \tag{2.2}$$

2.2.2 Results

Figure 2.9 shows the energy histogram of a few thousand events from the Co-60 sample on the Gamma Spectrometer. Note the two very distinct peaks that can be used for calibration.

After this data was taken, the gain on the amplifiers was halved (see appendix F) in order to obtain a range of over 30 MeV. Additional data resulted in final peaks at channel 107.5 and 122, which correspond to $m = 0.011$ MeV/channel and $b = -0.0125$ MeV.

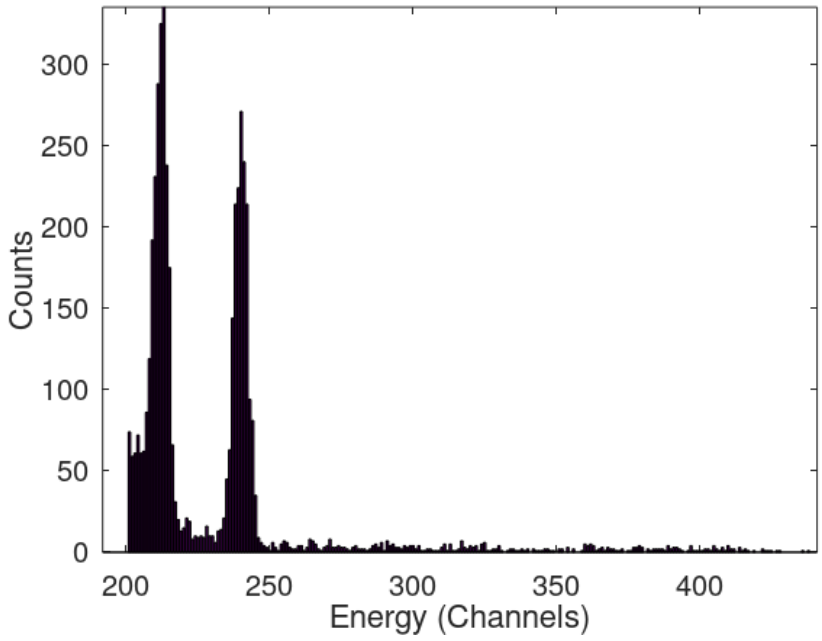

Figure 2.9 Co-60 calibration data from the Gamma Spectrometer. Note the two distinct peaks, which correspond to 1.17 MeV and 1.33 MeV respectively.

Octave's plotting tool was used to determine the FWHM of these pulses manually. The left peak has a FWHM of 40 keV and the right peak has a FWHM of 30 keV.

2.2.3 Discussion

These peaks provide two calibration points to accurately convert signals to energies in MeV. The limitations to this measurement come from the digitization and binning of the data. However, the low widths indicate a relatively high precision in the Gamma Spectrometer, allowing one to distinguish between close peaks.

The conservative FWHM of 40 keV is taken as the energy uncertainty for this device. With this resolution, the Gamma Spectrometer is able to distinguish between the energies of gamma rays on the MeV scale. Much like with the other detectors, these measurements are important to verify p-d and d-d fusion in future experiments.

2.3 dE-E Charged-Particle Spectrometer

The ΔE-E is made of two surface barrier detectors (SBDs) to form a telescope. This spectrometer is set inside the vacuum chamber close to the target. These SBDs are reversed biased, which makes them sensitive to charged particles. This is used to measure charged particles heavier than electrons, such as protons and tritons, which are expected products from d-d fusion.

An incoming charged particle deposits some kinetic energy into the first window SBD (26-μm-thick Ortec SBD), generating a voltage pulse with a height proportional to this energy. It deposits its remaining kinetic energy into the thicker second diode (130-μm-thick Ortec SBD). These energies are amplified and filtered, then recorded by the CAEN. See appendix A for more on this.

The sum of these two energies is the total energy of the incoming particle. The relationship between these two energies indicates the kind of particle it is, as demonstrated in figure 2.10. Heavier particles tend to deposit more energy in the first detector, for example, while lighter particles deposit less.

2.3.1 Methods

The ΔE-E can measure charged particles such as protons at ~3 MeV and tritons at ~1 MeV, which are d-d fusion products of interest. An Americium-241 source was used to calibrate this spectrometer since it emits alpha particles at ~5.5 MeV [26]. This gives an appropriate range to measure these fusion products.

A Monte Carlo program (Geant4) simulated the setup to find the expected energies deposited from Am-241. Python code mapped the spectrometer calibration data to the simulation results via equation 2.3, where the m and b values are the slopes and y-intercepts for the calibration and simulation data (see appendix A for the code and appendix H for related work).

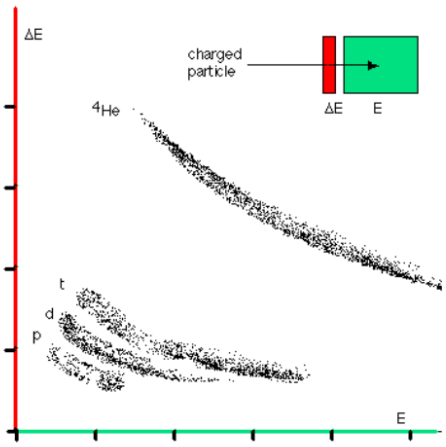

Figure 2.10 Plot demonstrating the ΔE-E system [25]. On the top, a charged particle deposits energy on the two SBDs. The plot of these energies is shown below. Note that different types of particles bunch up in different regions of the plot. Heavier particles generally are further away from the origin. This allows the classification of measured particle types as well as energies.

$$\Delta E_{new} = (\Delta E - b_{data}) \cdot \frac{m_{sim}}{m_{data}} + b_{sim} \tag{2.3}$$

From this, the code obtained calibration parameters m and b to alter the digitized ΔE values. ΔE was modified with equation 2.4. The final parameters $m = m_{sim}/m_{data}$ and $b = -b_{data} \cdot m_{sim}/m_{data} + b_{sim}$ were derived from equation 2.3.

$$\Delta E_{new} = m \cdot \Delta E + b \tag{2.4}$$

2.3.2 Results

Figure 2.11 shows the simulation data and the remapped calibration data together. This was the result of about 10,000 alpha samples from both simulation and real-world data.

The E SBD was already reasonably calibrated to fit the simulation data, meaning it has calibration parameters of $m \approx 1$ and $b \approx 0$.

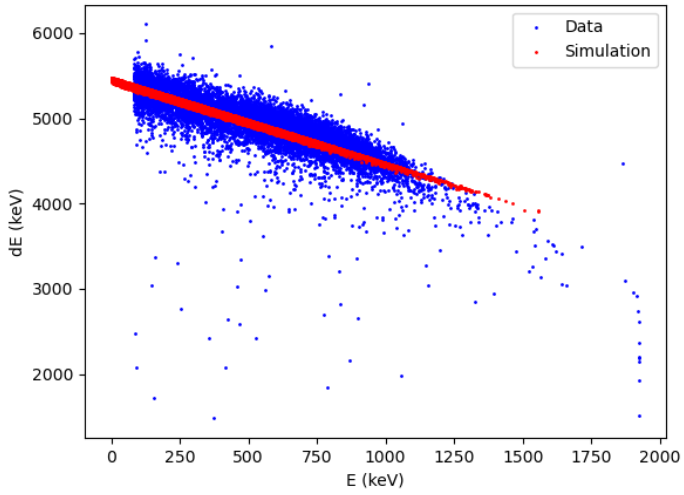

Figure 2.11 Am-241 calibration data measured by the ΔE-E. This is mapped to have the same slope as the Geant4 simulation data. Both are shown for comparison. Note that the dE is the ΔE, which is the thinner front SBD, while the E is the back SBD.

The ΔE had calibration parameters of $m = 3.29$ keV/channel and $b = -569.6$ keV.

Using these parameters, one can collect a signal from the spectrometer, multiply it by m and add b to obtain the actual energy of a given event.

To ensure proper calibration, the total deposited energy was plotted on a histogram to find the resolution (figure 2.12). The position of the peak was around 5500 keV. The most common alpha energy from ^{241}Am is 5.49 MeV, so this matches expectation. The FWHM of this peak is 313 keV, which can be attributed primarily to electronic noise and secondarily to a natural spread in alpha energy as it exits the source. The FWHM divided by the peak's position is 0.057, which is a normalized resolution factor.

2.3.3 Discussion

ΔE-E now has calibration factors, allowing the computer to convert the signals from this instrument into energies. The resolution of about 300 keV is sufficient for measuring particles in

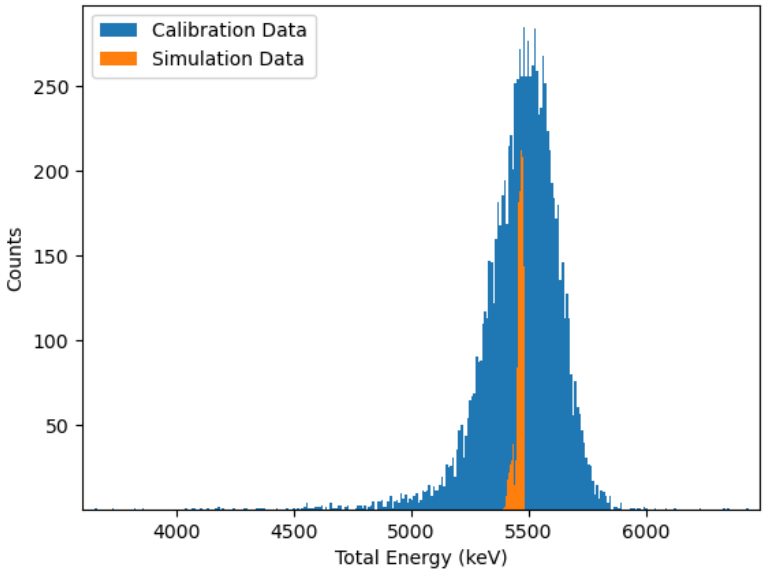

Figure 2.12 Histograms showing the total energy (ΔE+E) from the simulation and the calibration data. Note the peak around 5500 keV, as expected.

the MeV range, such as the $\sim$ 3 MeV proton from d-d fusion. This system is now able to distinguish between particles of different masses and characterize their energies. This will allow one to identify particles from reactions and verify fusion.

2.4 Neutron Pseudo-Spectrometer

The Neutron Pseudo-Spectrometer utilizes a 125 mm diameter x 125 mm thick PVT cylinder interspersed with lithium gadolinium borate (LGB) crystals (see figure 2.13). A neutron entering the detector is thermalized by a proton in the PVT, releasing a pulse of light, often called the "thermal pulse." The neutron is then slowed down by the LGB crystals and captured, releasing a large pulse of light that dies off exponentially—the "capture pulse." These scintillations are received by a PMT, which sends a voltage signal to the Nuclear Instrumentation Modules (NIM

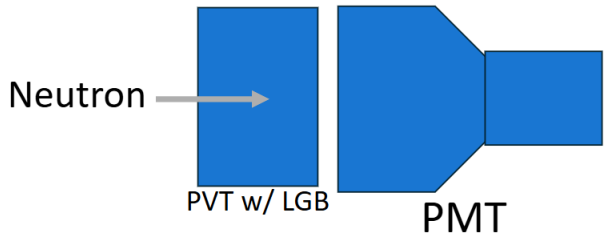

Figure 2.13 A diagram of the neutron detector. The LGB crystals are interspersed within the PVT disk. The neutron enters this, gets thermalized by a proton, and is relatively slowly captured by the LGB crystals. The PMT sees the resulting light pulses from this process.

modules). These process the analog signal before sending it to the CAEN to be digitized and processed by the computer (see appendix D). An example signal is shown in figure 2.14.

The author and his collaborators made multiple attempts to calibrate the neutron detector. The first attempt was with Californium-252, which was also used to test the code used to process the neutron pulses. The data revealed a neutron energy spectrum; however, the shape was difficult to distinguish and did not give conclusive calibration data. The second attempt was to use a student-made Farnsworth Fusor as a 2.54 MeV neutron source. The difficulty here came from the strong electric fields from the wires delivering high voltage to the device. This interfered with the signal through the BNC cable, so this data was unusable.

The neutron detector is not yet calibrated, but reactions like d-d and p+7Li will provide mono-energetic neutrons to do that.

2.5 Veto

The Veto detector is a large plastic scintillator paddle detector set above the vacuum system to detect background phenomena. If the energy detected by this device was significantly above background voltage, the system did not save any triggers from the other detectors but attributed it to external particles, such as from local nuclear processes or cosmic rays.

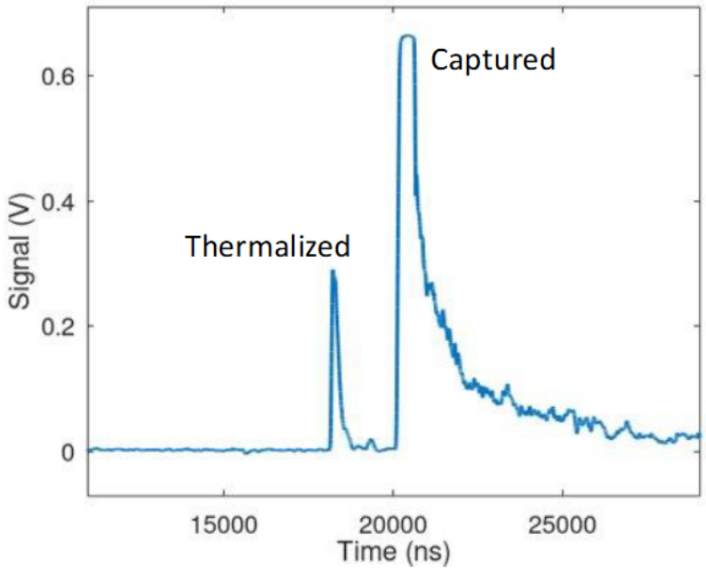

Figure 2.14 A sample neutron signal. An energetic neutron deposits energy in the hydrogen of the PVT, releasing light that is measured by a PMT. The thermalized neutron is then captured by the LGB crystals in a distinct slower process.

There was no need to distinguish the actual energy of these particles, so this detector did not need calibration (see appendix C for more).

2.6 Beam Current

To measure the current of the ion beam, the target holder was isolated from ground and connected via BNC to a Keithley Electrometer 610B. This electrometer was also connected to an HP 3457A Multimeter to digitally display the current as a voltage reading. The voltage reads from 0 to 1 volt, which corresponds directly to the reading on the electrometer. The digital multimeter was linked via a Prologix GPIB-Ethernet Controller to the lab network, allowing the data to be collected and plotted in Octave. This allowed remote recording of the beam current (see appendix E).

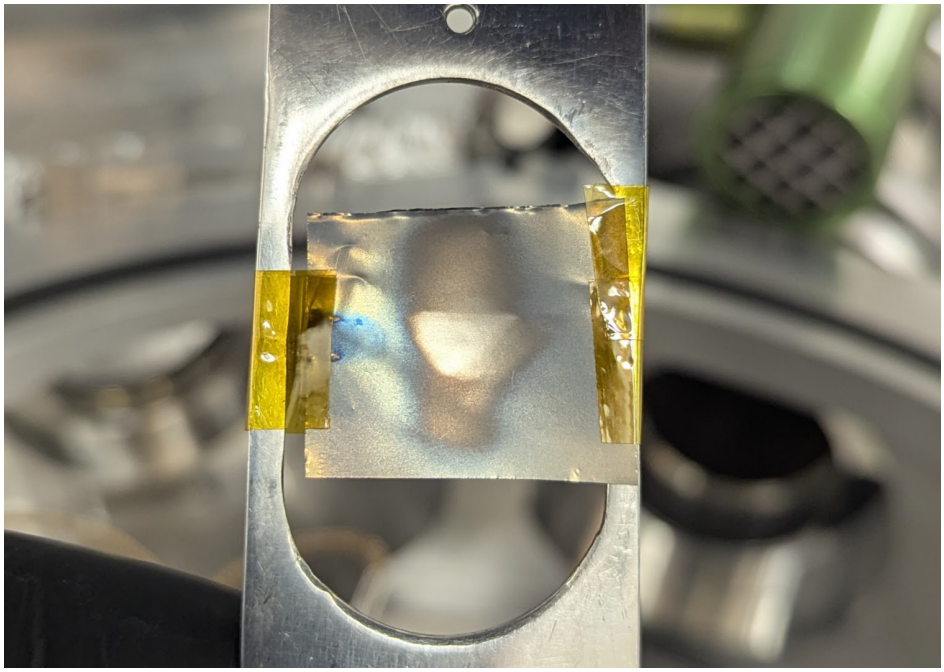

Figure 2.15 The target holder with one of the targets attached via kapton tape. Note the discoloration where the ion beam hit the target.

2.7 Ion Gun

The ion gun is the PerkinElmer 04-303 controlled by the 11-065 ion gun Control. This gun was capable of generating a beam of ions from 0 to 5 keV. The current from this gun on the beam target maximized around 25 μA, but 18–20 μA was far more stable for operation.

The device had both physical and electronic deflection methods. To ensure nominal alignment, a glass lid was placed on the chamber to observe the beam on the titanium target. Afterwards, to verify alignment, the current was maximized as a function of beam angle, which indicated that the beam was incident on the target or target holder. Alignment was further verified by observing the discoloration on the titanium, clearly caused by the ion beam itself (see figure 2.15). All of this discoloration was located around the center of the target.

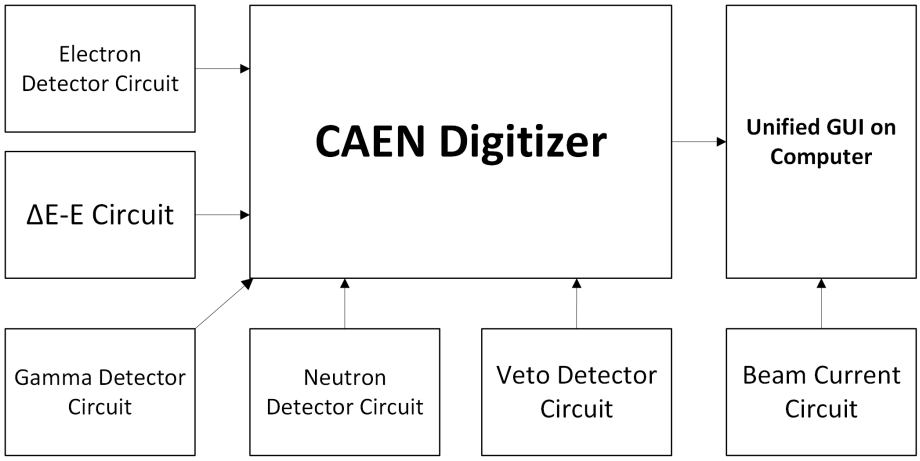

Figure 2.16 Block diagram of the overall lab setup showing how the data from the detectors is digitized and sent to the computer.

2.8 Integration into a Graphical User Interface (GUI)

The OneProgram Octave software reads the data from files created by the CAEN and plots the relevant detector data onto various histograms in a GUI (see figure 3.2). It also displays the live beam current from the electrometer and saves it to a CSV file. The code analyzes each channel from the CAEN data to process all the detector waveforms.

This program works on live data, with some processing delay, as well as previously obtained data to view later (see appendix G for the code).

Figure 2.16 demonstrates how all of the data from the detectors is digitized by the CAEN and processed on the computer. All of this processed data is then displayed on the GUI.

Chapter 3

Conclusion

In summary, this thesis discussed the setup and calibration of multiple spectrometers and the Graphical User Interface (GUI) to process and display instrument data. The ΔE-E charged-particle spectrometer, electron spectrometer, and gamma spectrometer are arranged and calibrated to measure fusion products for future experiments. Various radioactive sources with known energies provided anchor points to convert from signal voltage to energy. Other equipment, such as the beam current measurement system, neutron pseudo-spectrometer, and veto panel, also aids in monitoring the experiments.

Though there is uncertainty in the detectors, as shown in table 3.1, their resolution is high enough to verify the presence and energies of products from p-d or d-d fusion. If the ΔE-E and gamma spectrometers measure particles close to the expected energies (see table 1.1), this is evidence of fusion occurring.

The facilities are now ready for experiments using the 4 keV ion beam on TiD_x or TiH_x to search for these electrons. It will be important to collect data from all detectors both with and without the ion beam for comparison. After several hours of data with the ion beam, the facility is expected to measure fusion products as well as ejected electrons with the Q-value of the reactions. Due to the chamber's broad measurement capabilities and modular design, it will be possible to use

Detector	Uncertainty
Electron	160 keV
Gamma	40 keV
ΔE-E	313 keV

Table 3.1 Detector energy uncertainties. Note the neutron detector is not here since it was not successfully calibrated.

for other experiments beyond only p-d and d-d fusion. For example, proton-boron and deuterium-tritium fusion could be performed and measured with these facilities.

As shown in figure 3.2, data has already been collected from the deuterium ion beam onto a hydrated titanium target. Several runs are now complete, both for hydrated and deuterated targets, each typically a few hours at a time. To this point, no significant fusion reactions have been measured at the expected energies. In order to increase the probability of fusion events, the beam time and/or beam current must be increased. To address this, the author has designed and is building a new deuterium ion gun intended to have beam currents in the milliampere range instead of the microampere range (see appendices J and K for development details). Since this is based on the DC-25 from Oxford Applied Research used by Forbes' team, this is expected to have 100–1000 times the current of the original gun, which would aid significantly in the experiments [15]. This development process has included 3D CAD design, ion-optics simulations, and physical construction of the system. It is nearly ready to be installed on the chamber and tested (see figure 3.1.

If the detectors find evidence of fusion and measure 5.5 MeV or 23.9 MeV electrons, this could be evidence for electron-catalyzed fusion occurring in metal. This would provide a possible explanation for the various anomalous high-energy events other researchers have observed in low-energy nuclear research. This may also reveal more about how electrons interact with nuclei in fusion events. The energies used in these experiments are on the same order as the particles that undergo fusion in the Sun. Although the physical environments are different, the underlying

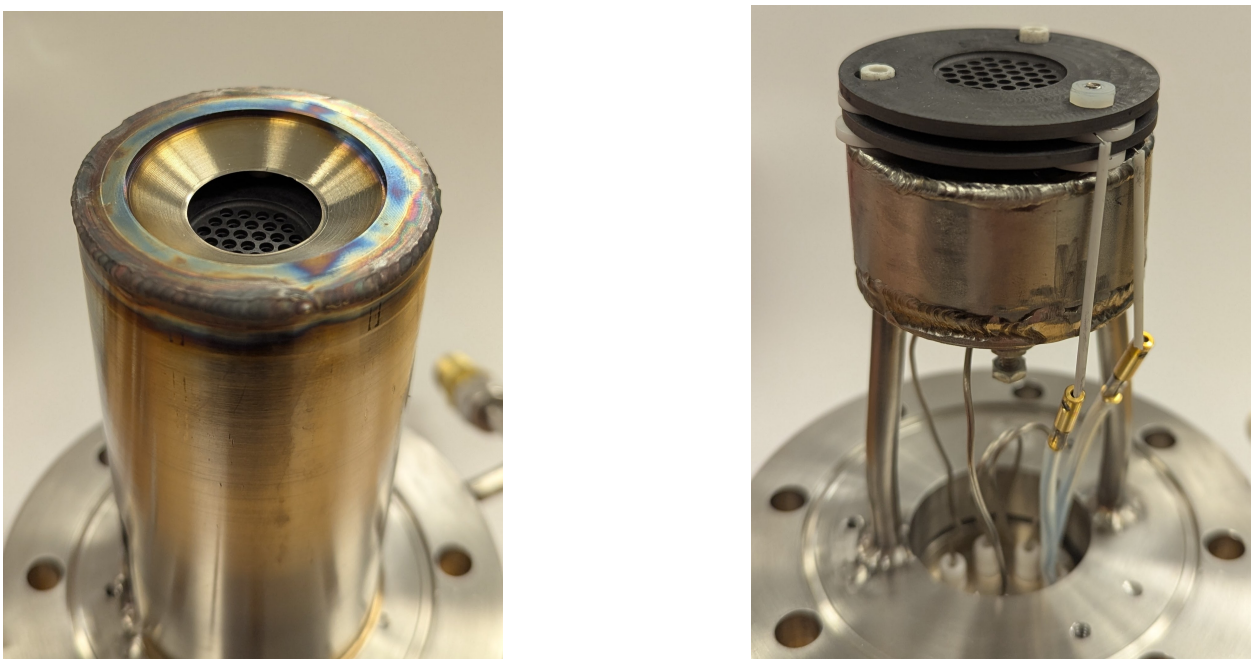

Figure 3.1 The nearly complete ion gun with and without the shell. Note the two grids for ion acceleration. The cylinder houses magnets and the filament for generating a plasma.

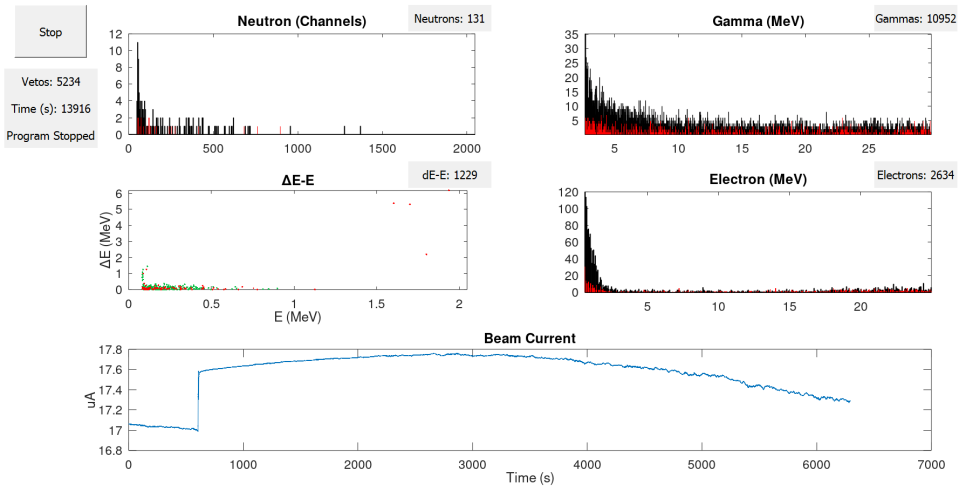

Figure 3.2 Sample of detector data after a long run of the ion beam displayed in the unified GUI. This was deuterium on a hydrated titanium target. Black and green are data points, while red are the vetoed data points (events in coincidence with the veto panel).

nuclear reactions are similar. Thus, a better understanding of these reactions will lead to greater insight into the nuclear processes within the Sun.

Bibliography

[1] H. Kragh, "The Source of Solar Energy, ca. 1840–1910: From Meteoric Hypothesis to Radioactive Speculations," European Physical Journal H **41,** 365–394 (2016).

[2] A. S. Eddington, "The Internal Constitution of the Stars," The Observatory **43,** 353–357 (1920).

[3] University of Southampton, "Nuclear Fusion (Phys3002 Course Notes, Chapter 11),", https://www.southampton.ac.uk/~ab1u06/teaching/phys3002/course/11_fusion.pdf, n.d., accessed: 2025-09-18.

[4] B. Ricci, V. Berezinsky, S. Degl'Innocenti, G. Fiorentini, and M. Lissia, "Helioseismic Constraints to the Central Solar Temperature and Neutrino Fluxes," Physics Letters B **407,** 155–160 (1997).

[5] C. E. Rolfs, *Cauldrons in the Cosmos* (The University of Chicago Press, 1988).

[6] E. E. Salpeter, "Electron Screening and Thermonuclear Reactions," Australian Journal of Physics **7,** 373–388 (1954).

[7] G. Gamow, "Zur Quantentheorie des Atomkernes," Zeitschrift für Physik **51,** 204–212 (1928).

[8] V. S. Dhillon, "PHY213: The Physics of Stellar Interiors – Occurrence of Fusion Reactions,", https://vikdhillon.staff.shef.ac.uk/teaching/phy213/phy213_fusion2.html, university of Sheffield teaching webpage, accessed 2026-04-30.

[9] H. A. Bethe, "Energy Production in Stars," Physical Review **55,** 434–456 (1939).

[10] F. C. Frank, "Hypothetical Alternative Energy Sources for the 'Second Meson' Events," Nature **160,** 525–527 (1947).

[11] L. W. Alvarez *et al.*, "Catalysis of Nuclear Reactions by μ Mesons," Physical Review **105,** 1127–1128 (1957).

[12] S. E. Jones, J. Rafelski, W. A. Perkins, J. M. Worlock, and H. Winkler, "Observation of Cold Nuclear Fusion in Condensed Matter," Nature **321,** 127–133 (1986).

[13] M. Lipoglavšek, S. Markelj, M. Mihovilovic, T. Petrovic, S. Štajner, M. Vencelj, , and J. Vesic, "Observation of electron emission in the nuclear reaction between protons and deuterons," Phys. Lett. B (2017).

[14] K. Czerski, R. Dubey, M. Kaczmarski, A. Kowalska, N. Targosz-Ślęczka, G. Das Haridas, and M. Valat, "Indications of electron emission from the deuteron-deuteron threshold resonance," Physical Review C 109 (2024).

[15] S. Forbes, P. L. Hagelstein, and F. Metzler, "Energetic Ion Emission from TiDx in Low-Energy Ion Beam Experiments," J. Condensed Matter Nucl. Sci. (2024).

[16] L. Holmlid and S. Olafsson, "Decay of muons generated by laser-induced processes in ultra-dense hydrogen H(0)," Heliyon **5,** e01864 (2019).

[17] K. Hansen and J. Engelen, "Are claims of cheap muon production correct?," Energy, Sustainability and Society **13,** 24 (2023).

[18] K. Czerski, A. Huke, A. Biller, P. Heide, M. Hoeft, and G. Ruprecht, "Enhancement of the electron screening effect for d+d fusion reactions in metallic environments," EPL (2001).

[19] N. D. Cook, *Models of the Atomic Nucleus: Unification Through a Lattice of Nucleons* (Springer, Berlin, Heidelberg, 2006).

[20] CAEN S.p.A., "V1720: 8 Input Channel 12-bit 250 MS/s Digitizer,", 2025, accessed: 2026-07-09.

[21] T. Martin, "Development of a High-Energy Electron Spectrometer For the Study of Lepton-Catalyzation in Nuclear Fusion Reactions," Undergraduate Senior Thesis, Brigham Young University (2024).

[22] C.-H. Lee and J. Seon, "Energy calibration of charged particle detectors using discrete and continuous energy spectra of 137Cs," Applied Radiation and Isotopes **168,** 109484 (2021).

[23] R. Chakravarty, A. Dash, and M. R. A. Pillai, "Availability of yttrium-90 from strontium-90: A nuclear medicine perspective," Cancer Biotherapy & Radiopharmaceuticals **27,** 621–641 (2012).

[24] K. Buchtela, "Cobalt-60," in *Encyclopedia of Analytical Science*, second edition ed. (Elsevier, 2005).

[25] J. McKenna, J. Ellsworth, and D. Rees, "Charged Particle Spectrometer for LNAR," In *Bulletin of the American Physical Society*, APS Four Corners Section Meeting (2008), presented at the APS Four Corners Section Meeting (4CS).

[26] G. F. Knoll, *Radiation Detection and Measurement*, 4 ed. (Wiley, 2010), p. 400.

[27] G. by Physino, "Geant4 Example Application with Rich features and Small footprints,", 2023, used Am241 mac file found on page.

Index

Appendix A

Calibrating the dE-E Charged-Particle Spectrometer

Included here is the ΔE-E detector NIM module circuit and the code to process the collected data. Also included is the code implemented in Geant4 via Gears. The data was saved in a root file that was then processed by the Python file.

A.1 Python Code For Analysis

Listing A.1 Python code to fit calibration data to simulation data.

```
import uproot
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import linregress
import statistics as st
from scipy.optimize import curve_fit


```

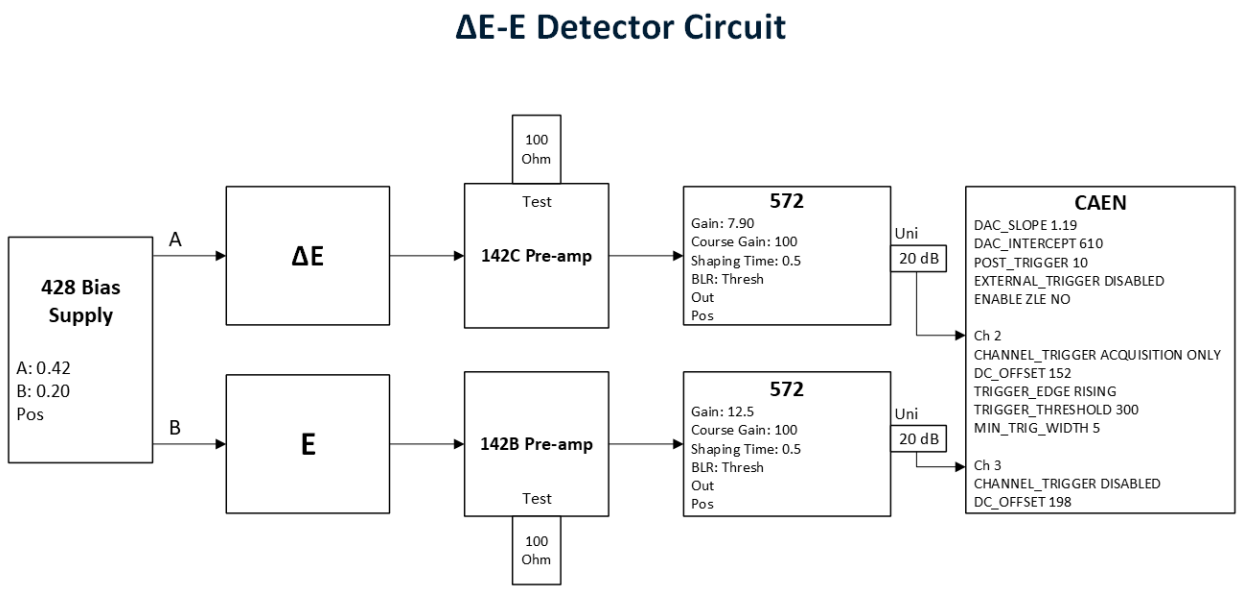

Figure A.1 ΔE-E detector circuit

```
#%% Get the Simulation Results

# Open the ROOT file
with uproot.open("panda2.root") as file: # Originally "800kSamples.
    root"
    tree = file["t"]  # replace with your actual TTree name

    # Load branches into a pandas DataFrame
    df = tree.arrays(["et"], library="pd")  # pd = pandas

    # Save to CSV
    #df.to_csv("output.csv", index=False)

# Save as a np array
temp = df.to_numpy()
dESim = np.array([])
ESim = np.array([])

# Remove empty cells and assign data to dE and E
for a in temp[:,0]:
    if len(a)==3 and a[1]>0:
        dESim = np.append(dESim,a[1])
        ESim = np.append(ESim,a[2])

# Plot
```

```
plt.figure()
plt.scatter(ESim,dESim,s=1,c="red")
plt.show()
plt.xlabel("E")
plt.ylabel("dE")


# Find slope of line
ESim=ESim[ESim<2500]
dESim=dESim[dESim>1000]
statsSim = linregress(ESim,dESim)
slopeSim = statsSim.slope
interceptSim = statsSim.intercept
print(f"Slope: {statsSim.slope}")
print(f"Intercept: {statsSim.intercept}")
print(f"R-squared: {statsSim.rvalue**2}")




#%% Get the Experimental Results

df2 = pd.read_csv("dEEData.csv")
temp = df2.to_numpy()
dE = np.array([])
E = np.array([])

for b in temp:
    if b[1] > 600 and b[1]<2047:
        dE = np.append(dE,b[1])
        E = np.append(E,b[0])




#dE = dE*2 ## REMOVE THIS LATER TODO
   ####################################




# Find slope of line
stats = linregress(E,dE)
slope = stats.slope
intercept = stats.intercept
```

```
print(f"Slope: {stats.slope}")
print(f"Intercept: {stats.intercept}")
print(f"R-squared: {stats.rvalue**2}")


# Plot the line on top of the data
plt.figure()
x = [0,2000]
y = [intercept,2000*slope+intercept]
plt1 = plt.scatter(E,dE,s=1,c="red")
plt2 = plt.plot(x,y)


# Find the transformation parameters
Enew = E
dEnew = (dE*slopeSim/slope - intercept*slopeSim/slope + interceptSim)
print(f"mAdjust = {slopeSim/slope}\nbAdjust =  {- intercept*slopeSim/
    slope + interceptSim}")


#%% Plot the result
plt.figure()
p1 = plt.scatter(Enew,dEnew,s=1,c="blue")
p2 = plt.scatter(ESim,dESim,s=1,c="red")
plt.xlabel("E (keV)")
plt.ylabel("dE (keV)")
plt.legend(["Data","Simulation"])


#%% Plot the total of Enew and dEnew

tot = Enew+dEnew
totSim = ESim+dESim

plt.figure()
counts, bins, patches = plt.hist(tot,256)
#totSimHist = plt.hist(totSim,100)
plt.show()
plt.xlabel("Total Energy (keV)")
plt.ylabel("Counts")
#plt.legend(["Calibration Data", "Simulation Data"])

#%% Fit the data to a gaussian


def gauss(x,A,mu,sigma):
```

```
    return A*np.exp(-(x-mu)**2 / (2*sigma**2))

centers = 0.5*(bins[1:]+bins[:-1])

p0 = [counts.max(),centers[np.argmax(counts)],np.std(tot)]

popt,pcov = curve_fit(gauss,centers,counts,p0=p0)
A,mu,sigma = popt
print(A)
print(mu)
print(sigma)

fit = gauss(centers,A,mu,sigma)

plt.figure()
plt.plot(centers,counts,centers,fit,'r','linewdith'==7)
plt.legend(["Calibration Data", "Gaussian Fit"])
plt.xlim([min(centers),max(centers)])
plt.xlabel("Total Energy (keV)")
plt.ylabel("Counts")
plt.show()

fwhm = 2*np.sqrt(2*np.log(2))*sigma
print("\nFWHM")
print(fwhm)
```

A.2 Geant4 Gears Code for Simulation

A.2.1 Main Macro File

This file is the main macro file for the simulation. It is run by entering "gears mainCalibration.mac" in a terminal at the file's location.

Listing A.2 Macro file for Am241 on dEE calibration simulation.

```
# print macro commands on screen
/control/verbose 1

# geometry must be specified before /run/initialize
/geometry/source detectors.tg
```

```
6
7  # Starts the simulation and ensures that there are no overlapping
       volumes
8  /run/initialize
9  /geometry/test/run
10
11 # Prepares to be saved as a WRL file
12 /vis/open VRML2FILE
13 /vis/viewer/zoom 1
14 /vis/viewer/set/background white ! ! 0
15 /vis/drawVolume
16
17 # Save to a root file
18 # Open and edit by putting 'root filename.root' in the terminal
19 # See https://root-forum.cern.ch/t/view-contents-of-root-file-like-a-
       csv-viewer/51604 for more information
20 /analysis/setFileName data1.root
21
22 # Starts the data collection and trajectory storing
23 # Trajectories can be rich or smooth
24 /vis/scene/add/trajectories rich
25 /vis/scene/endOfEventAction accumulate
26
27 # Runs an external .mac file
28 /control/execute am241Source.mac
29
30 # Generates [number] particles from the General Particle Source (gps)
31 /run/beamOn 1000000
32
33
34
35 # Run this .mac file by entering "gears [filename].mac" in the
       command terminal. It will print out any errors
36
37 # Reading the root file after data collection:
38 #   1.  Enter "root [filename].root" in the command terminal
39 #   2. Use the following commands in the root session:
40 #       a. t->Show(0)
41 #           i. This will show all the variables stored and their
       names (see link above for more details)
42 #       b. t->Draw("name")
43 #           i. This will draw a graph for the counts of the given
       variable
44 #       c. For more specific drawings:
45 #           i. t->Draw("[y-axis variable]:[x-axis variable]","vlm==[
       copy number]")
```

```
#                1) The volume copy number can be specified in the tg
    file
#            ii. You can also specify a particle type:
#                1) t->Draw("q", "vlm == 1&&pdg == 11")
#                    a) Pdg number comes from https://pdg.lbl.gov
    /2007/reviews/montecarlorpp.pdf
#            iii. To view the values on the graphs instead of the
    histogram bars
#                1) t->Draw("[variable]", "[conditions]", "TEXT")
#            iii. Add more conditions as needed, for example, "q < 40"
#                1) Remember the "&&" if there are multiple conditions


# To fit a line in Root
# 1. t->Draw("graph","","goff")
# 2. TGraph *g = newTGraph(t->GetSelectedRows(), t->GetV2(), t->GetV1
    ())
# 3. g->Draw("AP")
# 4. g->Fit("pol1") (For linear)
#   a. Chi2 is how bad the fit is
#   b. NDf is the data points minus the fitted parameters (2 for
    linear)
#   c. p0 is the first parameter (constant terminal
#   d. p1 is the slope parameter


#More Root Tips: https://physino.xyz/gears/tutorials/physics/
```

A.2.2 Am-241 Source Macro File

This next file is to simulate an Am-241 source with all of its possible particles and their frequency. This file was originally from Physino [27].

Listing A.3 Macro file that has accurate energies and intensities for the various products of Am-241

```
# See help manual https://www.fe.infn.it/u/paterno/Geant4_tutorial/
    slides_further/GPS/GPS_manual.pdf
# for gps help



# This declares the variable [name] and gives it a value to use later
```

```
/control/alias srcpos "0 0 2 cm"


# gps stands for General Particle Source


# alpha ray
## alpha1
# [intensity] is from 0 to 1. Gives percent chance of happening versus other gps sources
/gps/source/intensity 0.8445 # 84.45%

# Set the type of particle. Options include alpha, gamma, e-, and more
# You can also make a custom particle via
# /gps/particle/ion
# /gps/ion Z A Q E (atomic number, atomic mass, charge, excitation energy)
/gps/particle alpha


# General particles default to emitting from a point, but this can be changed
# Volume, Plane, Beam, Surface (see manual for details on each)

# define a shape that is slightly larger than <physics_volume_name>
/gps/pos/type Volume
/gps/pos/shape Cylinder
/gps/pos/radius 1.1 cm #0.2 cm
/gps/pos/halfz 11 micrometer #0.05 cm
# place it to a position so that it can fully contain <physics_volume_name>
/gps/pos/centre {srcpos}
# GPS will generate particles only in <physics_volume_name>
/gps/pos/confine Am

# Change the angular distribution type
# Iso (isotropic) emits particles in every direction
# Other options are cos, planar, beam1d, etc.
# User defined is an option (see manual)
/gps/ang/type iso

# Optional: you can set the angle restrictions
#/gps/ang/mintheta [theta] deg      # Min polar angle
#/gps/ang/maxtheta [theta] deg     # Max polar angle (90 degrees for hemisphere)
```

```
#/gps/ang/minphi [phi] deg        # Azimuthal angle range (0 degrees)
#/gps/ang/maxphi [phi] deg      # Full azimuthal angle (360 degrees to cover all around)

# Sets the energy distribution type
# Lin, Pow, Gauss, Brem are a few options
# Mono sets only one energy
/gps/ene/type Mono
/gps/ene/mono  5.48556 MeV


##alpha2
# Each particle after the first will use
# /gps/source/add instead of intensity.
# [number] is the probability that this particle will appear each event
/gps/source/add  0.1323 # 13.23%
/gps/particle alpha

# define a shape that is slightly larger than <physics_volume_name>
/gps/pos/type Volume
/gps/pos/shape Cylinder
/gps/pos/radius 1.1 cm #0.2 cm
/gps/pos/halfz 11 micrometer #0.05 cm
# place it to a position so that it can fully contain <physics_volume_name>
/gps/pos/centre {srcpos}
# GPS will generate particles only in <physics_volume_name>
/gps/pos/confine Am

/gps/ang/type iso
/gps/ene/type Mono
/gps/ene/mono  5.44286 MeV


# gamma ray
## gamma 1
/gps/source/add 0.0240 # 2.40%
/gps/particle gamma

# define a shape that is slightly larger than <physics_volume_name>
/gps/pos/type Volume
/gps/pos/shape Cylinder
/gps/pos/radius 1.1 cm #0.2 cm
/gps/pos/halfz 11 micrometer #0.05 cm
```

```
89  # place it to a position so that it can fully contain <
        physics_volume_name>
90  /gps/pos/centre {srcpos}
91  # GPS will generate particles only in <physics_volume_name>
92  /gps/pos/confine Am
93
94  /gps/ang/type iso
95  /gps/ene/type Mono
96  /gps/ene/mono 26.3446 keV
97
98
99  ## gamma 2
100 /gps/source/add 0.00121 # 0.121%
101 /gps/particle gamma
102
103 # define a shape that is slightly larger than <physics_volume_name>
104 /gps/pos/type Volume
105 /gps/pos/shape Cylinder
106 /gps/pos/radius 1.1 cm #0.2 cm
107 /gps/pos/halfz 11 micrometer #0.05 cm
108 # place it to a position so that it can fully contain <
        physics_volume_name>
109 /gps/pos/centre {srcpos}
110 # GPS will generate particles only in <physics_volume_name>
111 /gps/pos/confine Am
112
113 /gps/ang/type iso
114 /gps/ene/type Mono
115 /gps/ene/mono 33.1963 keV
116
117
118 ## gamma 3
119 /gps/source/add 0.3578 # 35.78%
120 /gps/particle gamma
121
122 # define a shape that is slightly larger than <physics_volume_name>
123 /gps/pos/type Volume
124 /gps/pos/shape Cylinder
125 /gps/pos/radius 1.1 cm #0.2 cm
126 /gps/pos/halfz 11 micrometer #0.05 cm
127 # place it to a position so that it can fully contain <
        physics_volume_name>
128 /gps/pos/centre {srcpos}
129 # GPS will generate particles only in <physics_volume_name>
130 /gps/pos/confine Am
131
```

```
/gps/ang/type iso
/gps/ene/type Mono
/gps/ene/mono 59.5409 keV


# Np X-ray ray
## Np-L_l
/gps/source/add 0.00848 # 0.848%
/gps/particle gamma

# define a shape that is slightly larger than <physics_volume_name>
/gps/pos/type Volume
/gps/pos/shape Cylinder
/gps/pos/radius 1.1 cm #0.2 cm
/gps/pos/halfz 11 micrometer #0.05 cm
# place it to a position so that it can fully contain <
   physics_volume_name>
/gps/pos/centre {srcpos}
# GPS will generate particles only in <physics_volume_name>
/gps/pos/confine Am

/gps/ang/type iso
/gps/ene/type Mono
/gps/ene/mono 11.89 keV

## Np-L_alpha
/gps/source/add 0.1303 # 13.03 %
/gps/particle alpha

# define a shape that is slightly larger than <physics_volume_name>
/gps/pos/type Volume
/gps/pos/shape Cylinder
/gps/pos/radius 1.1 cm #0.2 cm
/gps/pos/halfz 11 micrometer #0.05 cm
# place it to a position so that it can fully contain <
   physics_volume_name>
/gps/pos/centre {srcpos}
# GPS will generate particles only in <physics_volume_name>
/gps/pos/confine Am

/gps/ang/type iso
/gps/ene/type Mono
/gps/ene/mono 13.90 keV


## Np-L_beta_eta
```

```
/gps/source/add 0.1886 # 18.86 %
/gps/particle e-

# define a shape that is slightly larger than <physics_volume_name>
/gps/pos/type Volume
/gps/pos/shape Cylinder
/gps/pos/radius 1.1 cm #0.2 cm
/gps/pos/halfz 11 micrometer #0.05 cm
# place it to a position so that it can fully contain <
   physics_volume_name>
/gps/pos/centre {srcpos}
# GPS will generate particles only in <physics_volume_name>
/gps/pos/confine Am

/gps/ang/type iso
/gps/ene/type Mono
/gps/ene/mono 17.81 keV


## Np-L_beta_eta
/gps/source/add 0.0481 # 4.81 %
/gps/particle e-

# define a shape that is slightly larger than <physics_volume_name>
/gps/pos/type Volume
/gps/pos/shape Cylinder
/gps/pos/radius 1.1 cm #0.2 cm
/gps/pos/halfz 11 micrometer #0.05 cm
# place it to a position so that it can fully contain <
   physics_volume_name>
/gps/pos/centre {srcpos}
# GPS will generate particles only in <physics_volume_name>
/gps/pos/confine Am

/gps/ang/type iso
/gps/ene/type Mono
/gps/ene/mono 20.82 keV


# list all sources
/gps/source/list
```

A.2.3 Geometry File For Simulation

This code is the geometry file to create the simulated materials and their locations in the simulation.

```
:volu hall BOX 3*cm 3*cm 3*cm G4_Galactic // Define materials

// Define detector volumes
// Note on TUBE: the third dimension is the half-width of the
   detectors!
:volu detector1(S) TUBE 0*mm 0.977*cm 12.5*micrometer G4_Si // First
   detector (inner radius, outer radius, thickness). Adding "(S)"
   makes it a detector!
:volu detector2(S) TUBE 0*mm 0.977*cm 65.0*micrometer G4_Si // Second
    detector
:volu Am TUBE 0*mm 1*cm 0.05*micrometer G4_Am
:volu aluminum TUBE 0*mm 1*cm 0.015*micrometer G4_Al
:volu back TUBE 0*mm 1*cm 1000*micrometer G4_Al

// Place Am in the hall volume with the aluminum
:place Am 3 hall r000 0 0 2*cm
:place aluminum 4 hall r000 0 0 1.9999935*cm
:place back 5 hall r000 0 0 2.100005*cm

// Define rotation matrix (identity rotation in this case)
:rotm r000 0 0 0

// Place detector1. Distance from d1 is 1 mm
:place detector1(S) 1 hall r000 0*cm 0*cm -1000*micrometer

// Place detector2. Distance from d1 to d2 is 8 mm
:place detector2(S) 2 hall r000 0*cm 0*cm -9000*micrometer

// Set colors
:color detector1(S) 1 0 1
:color detector2(S) 0 0 1

// Hide hall volume in visualization
:vis hall OFF
```

Listing A.4 Geometry file for the two silicon parts of the dEE.

Appendix B

Calibrating the High Energy Electron Detector

Included here is the NIM module circuit for the electron detector to the CAEN.

We used Cs-137 as a beta and conversion electron source to calibrate the energy of this detector. We digitized the samples via the CAEN and processed it with Octave. We calibrated our energy to the curved peak, which should be at 624 keV, then increased our gain (adjusting the calibration

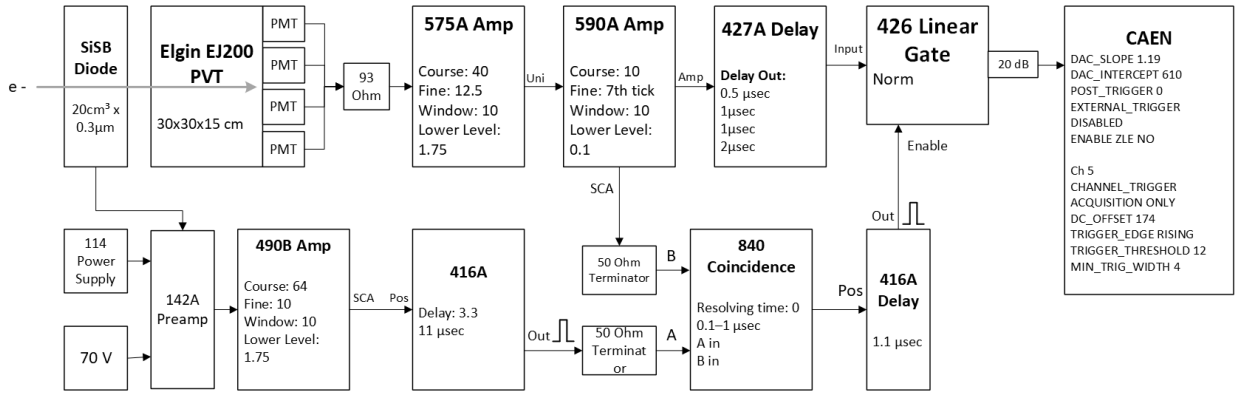

Figure B.1 The detection and signal processing circuit for the high energy electron detector

accordingly) to have a range of 20 MeV. We implemented the calibration factor and multiplied by the factor we reduced the gain by. The total factor we multiplied by was $(0.624/390) \cdot 8.33 \approx 0.0122$, where 0.624 was the peak in MeV, 390 was the peak in Channels originally, and 8.33 was the factor we reduced the gain by to achieve a range of 20 MeV.

B.1 Octave Code for Processing High Energy Electron Data

Edited code originally made by Taylor Martin [21].

```
% Code by Isaac Willden
% For the high energy electron detector
% Can also be used for other detectors (like the VETO). Just change
    the channel
% Started 4 April 2025

clear all; close all; clc;

% Open the folder and find the files. Select all the files you want
    to use

dirs = uigetdir('','Click on directory so file names show');

files = cellstr(uigetfile([dirs,'\textbackslash{}*.*'],'Highlight all
    files of interest', 'MultiSelect', 'on'));
ii=0;

% Reads the files and stores the data into the event array

for fCntr = 1 : length(files) %get the number of data files
    fileName = [dirs, '/', char(files(fCntr))];

    %create an array of data file names
    fid = fopen(fileName,'r');
    fseek(fid,0,'eof');
    fileLength = ftell(fid);
    fseek(fid,0,'bof');

    while ftell(fid) < fileLength - 1
        ii = ii+1;
        event(ii) = fEventRead8(fid);
    end
end
```

```
fclose('all');


% Plotting

% Constants
cha = 1; % Offset by +1 from the CAEN

% Cs 137 for electron calibration
mElec = 0.624/390 * 8.33;


% Find the maximum for each event

% .ch accesses the specific channels for the event.
for j = 1: ii-1
    peak = max(event(j).ch(:,cha))*mElec;
    peaks(:,j) = peak;
end

% Makes a histogram of the peaks
figure
hist(peaks,500)
title("Electron Energy Histogram")
```

B.2 Octave Code for fEventRead8 Function

This function is used in the electron detector code as well as most of the other programs for processing data.

```
function event = fEventRead8(fid)
%capture CAEN header values  per UM3244-DT5720 User Manual rev.10 p27
event.header = fread(fid, 4, 'uint32');
    %CAEN DATA EVENT format, Standar Mode, Normal Format (4 32-bit
       words):
    % 31302928 27262524 23222120 19181716 15141312 1110 9 8  7 6 5 4
       3 2 1 0
    %  1 0 1 0  * * * *  * * * *  * *  EVENT SIZE   * * * *  * * * *
       * * * *
    %    RESERVED     0          RESERVED                      *
       CHANNEL MASK *
    % RESERVED           * * * *  * EVENT COUNTER   * * * *  * * * *
       * * * *
```

```
    %  * * * * * * * * * * * TRIGGER TIME TAG  *  * * * * * * * * * * * *
event.patA = bitand(event.header(1),4026531840);   %1111 0000 0000 0000 0000 0000 0000 0000
event.Size = bitand(event.header(1),268435455);    %0000 1111 1111 1111 1111 1111 1111 1111
event.BoardID = bitand(event.header(2),4160749568);%1111 1000 0000 0000 0000 0000 0000 0000
event.ChannelMask = bitand(event.header(2),255);   %0000 0000 0000 0000 0000 0000 1111 1111
event.Counter = bitand(event.header(3),16777215);  %0000 0000 1111 1111 1111 1111 1111 1111
event.TrigTime = event.header(4);                  % all 32 bits

%Determine channel count
ChCnt = 0;
if bitand(event.ChannelMask,1) == 1
    ChCnt = ChCnt+1;
end
if bitand(event.ChannelMask,2) == 2
    ChCnt = ChCnt+1;
end
if bitand(event.ChannelMask,4) == 4
    ChCnt = ChCnt+1;
end
if bitand(event.ChannelMask,8) == 8
    ChCnt = ChCnt+1;
end
if bitand(event.ChannelMask,16) == 16
    ChCnt = ChCnt+1;
end
if bitand(event.ChannelMask,32) == 32
    ChCnt = ChCnt+1;
end
if bitand(event.ChannelMask,64) == 64
    ChCnt = ChCnt+1;
end
if bitand(event.ChannelMask,128) == 128
    ChCnt = ChCnt+1;
end

%Determine waveform length in 16 bit (12 bits used) words
length = ((event.Size - 4)/ChCnt)  *2;
%length = 1248

```

```
%read channel data enabled in channel mask, leaving no empty waveform channels
for i = 1:8  %4
  if(bitand(event.ChannelMask,(2^(i-1))) ~= 0) %only used channels
    %event.ch(:,i) = fread(fid, length, 'uint16');
    event.ch(:,i) = fread(fid, length, 'uint16') - 2047;
  end
end
end
```

B.3 MATLAB/Octave Code For Generating the Kurie Plot

Generative AI was used to learn about and write the code for this. Note that this code was run after the code for processing was run to generate the "peaks" array.

```
%% Kurie Plot

% - Kurie Plot for Sr-90/Y-90 endpoint calibration -

% Recompute histogram (same as your existing code)
numBins = 2047;
centers = linspace(0, 2047*m, numBins);
counts = hist(peaks, centers);

% - Fermi function approximation -
% For a rough Kurie plot, F(Z,E) ~ 1 is often used as a first approximation
% for low-Z detectors. If you want the full Fermi function, let me know.
Z = 39; % Y-90 daughter nucleus
% Simple approximation: F ~ 1 (acceptable for a calibration purpose)
F = ones(size(centers));

% - Endpoint energy assignment -
E_endpoint_MeV = 2.28; % Y-90 endpoint in MeV
bin_endpoint = 165;      % bin index you're assigning to the endpoint

% Convert bin centers to energy using linear scaling
E = centers * (E_endpoint_MeV / centers(bin_endpoint));

% - Kurie plot quantity -
% K(E) = sqrt(counts / (F * p^2))
% For relativistic electrons: p^2 = E^2 + 2*E*m_e, m_e = 0.511 MeV
```

```
m_e = 0.511; % MeV
p2 = E.^2 + 2*E*m_e;

% Mask out zero or negative counts to avoid sqrt of bad values
mask = counts > 0 & E > 0;
K = sqrt(counts(mask) ./ (F(mask) .* p2(mask)));
E_plot = E(mask);

% - Plot Kurie plot -
figure;
plot(E_plot, K, 'w.', 'MarkerSize', 8);
xlabel('Energy (MeV)');
ylabel('Kurie Function K(E)');
title('Kurie Plot - Y-90');
set(gca, 'FontSize', 20);
xlim([0, 3]);

% Fit region approaching the end
fit_low=1.8;
fit_high=2.3;

fit_mask = E_plot >= fit_low & E_plot <= fit_high;
p = polyfit(E_plot(fit_mask), K(fit_mask), 1);

% Extrapolate to find x-intercept (the endpoint)
E_fine = linspace(fit_low, 2.5, 1000);
K_fit  = polyval(p, E_fine);
E_zero = -p(2) / p(1); % x-intercept = endpoint estimate

hold on;
plot(E_fine, K_fit, 'r-', 'LineWidth', 2);
xline(E_zero, 'b-', sprintf('E_0 = %.3f MeV', E_zero), 'LineWidth',
   1.5);
legend('Data', 'Linear fit', 'Endpoint', 'Location', 'northeast');

fprintf('Estimated endpoint: %.4f MeV\n', E_zero);
fprintf('Residual from Y-90 true endpoint (2.28 MeV): %.4f MeV\n',
   E_zero - 2.28);

%% Uncertainty

% - Endpoint uncertainty from polyfit -
% Get covariance matrix of fit coefficients
[p, S] = polyfit(E_plot(fit_mask), K(fit_mask), 1);

% Uncertainty on slope (p(1)) and intercept (p(2))
```

```
sigma_p = sqrt(diag(inv(S.R' * S.R)) * S.normr^2 / S.df);
sigma_slope     = sigma_p(1);
sigma_intercept = sigma_p(2);

% Endpoint = -p(2)/p(1), propagate uncertainty
E_zero = -p(2) / p(1);
sigma_E0 = E_zero * sqrt((sigma_slope/p(1))^2 + (sigma_intercept/p(2)
   )^2);

fprintf('Endpoint: %.4f +- %.4f MeV\n', E_zero, sigma_E0);
```

Appendix C

The Veto Detector

Included here is the NIM module circuit for the Veto. The code to process the Veto detector is very similar to the high energy electron detector, but with a different channel number.

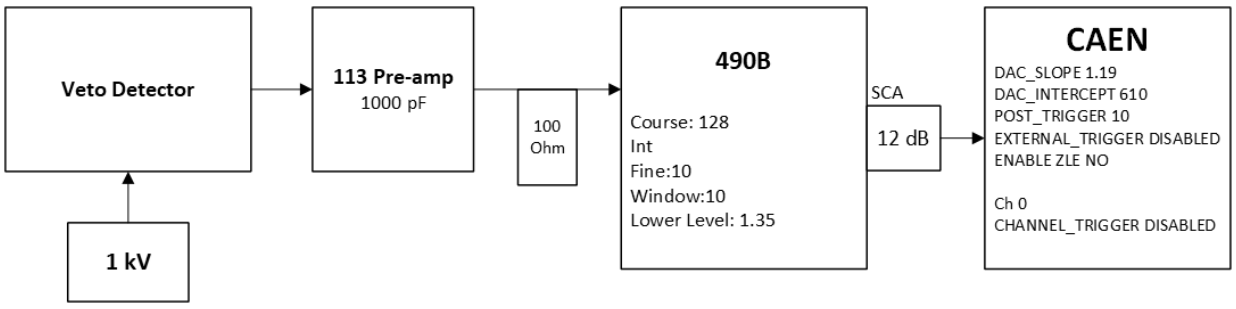

Figure C.1 Veto detector circuit diagram

Appendix D

Neutron Detection

We used various NIM modules to shape and discriminate the pulse shapes for the neutron detector. The pulses are digitized in the CAEN and then processed further in code. The 552 constant fraction discriminator is used to determine if a capture pulse is present. That is used to create a gate for the CAEN to trigger off of. The code then processes the digitized data, plotting heights of the thermal pulses closest to the capture pulse on a histogram. These heights are indicative of the energy level of the neutron.

The thermal pulses are distinguished in the code from the capture pulses through the ratios of the full width at half maximum (FWHM) divided by the height of the pulse. Because the capture pulses are slow processes, they should be wide and therefore have a somewhat larger FWHM/peak ratio than the thermal pulses. They were also distinguished from each other by their relative location to one another.

In the rare case that we observed two or more thermal pulses, the code only considered the one closer to the capture pulse.

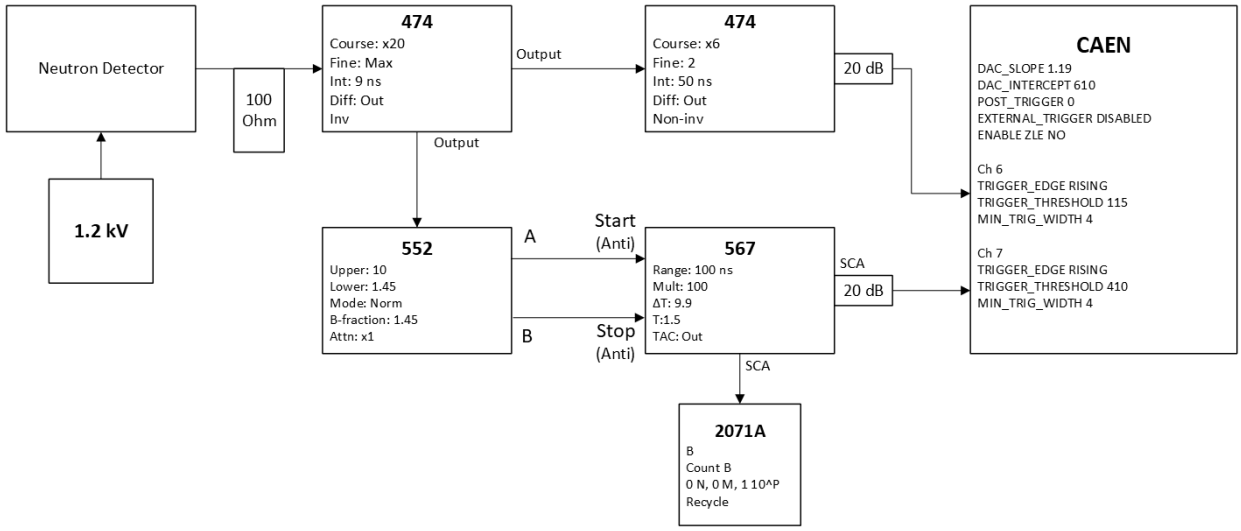

Figure D.1 Neutron detector circuit

D.1 Octave Code For Neutron Processing

```
% Code by Isaac Willden
% For the Neutron Detector
% Can also be used for other detectors (like the VETO). Just change
    the channel

% 21 April 2025
clear all; close all; clc;

% Open and read the file
event = readFile();
ii = length(event);


% Variables
a = 1; % Range of samples to plot (a to b)
b = 9;
ch1 = 7;
ch2 = 8;

% Plot some samples

%figure;
%for i = a:b
%  V = (event(i).ch(:,ch1))';
```

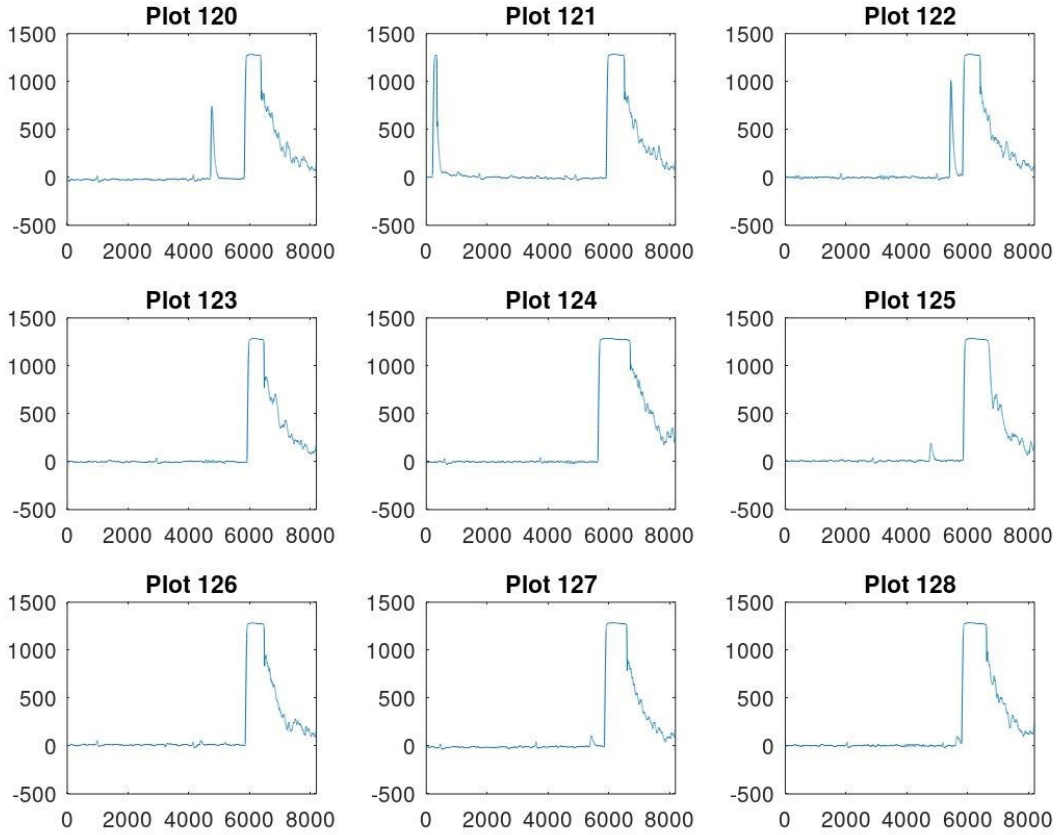

Figure D.2 Samples of waveforms from the neutron detectors. Note the smaller thermal pulses before the large capture pulse. x-axis units in channels. y-axis units in 1/2047 volts

```
%  x = 1:length(V);

%  subplot(ceil((b-a+1)/3),3,(i-a)+1)
%  plot(x,V);
%  xlim([0,length(V)]);
%  title(["Plot ", num2str(i)]);
%end
%pause(0.1)



% Find the Maximum and truncate the data to before the maximum
% Store final events into an array
peaks = neutronProcesser(ii,event,ch1,ch2); %processNeutron(ii,event,
    ch1,ch2,a,b);

% Makes a histogram of the peaks
if length(peaks)~=0
  bins = 255; % bins originally 500. Trying something
  %figure
  %hist(peaks,bins)
  %set(gca, "yscale", "log");
  %title("Log Neutron Peak Histogram")

  figure
  hist(peaks,bins)
  %title("Neutron Peak Histogram")
  xlabel("Channel")
  ylabel("Counts")
  set(gca, "FontSize", 20)
else
  disp("No neutrons found")
end
```

Listing D.1 Code to process neutron waveform data and generate a histogram.

```
% This is an experiment to improve on processNeutron to distinguish
    between thermals and capture pulses
function peaks = neutronProcesser(numEvents, evnts, cha, gate)

  % Constants
  leng = 8192;
  background = 45;
  minPoints = 10;
  minCaptureWidth = 1000;
  minCaptureHeight = 1000;
  captureRatio = 0.1;
```

```
11  thermalRatio = 1;
12  neutronGateThresh = 410;
13  skip = 50;
14
15  veto = 1;
16  vetoThresh = 50;
17
18  % Variables
19  evnt = zeros(1,leng);
20  peaks = [];
21  foundPulses = [];
22
23  % Run through each event
24  for i = 1:numEvents
25
26    % Display the event the code is on. Skip every [skip]
27    if mod(i,skip) == 0
28      disp([num2str(i),"/",num2str(numEvents)])
29    end
30
31    % Find what the neutron gate height is
32    gateHeight = max(evnts(i).ch(:,gate));
33    vetoHeight = max(evnts(i).ch(:,veto));
34
35    % If the height is large enough, process the neutron
36    if gateHeight > neutronGateThresh  && vetoHeight < vetoThresh
37
38      % Store the ith event into an array
39      evnt(:) = (evnts(i).ch(:,cha))';
40
41      % Some variables
42      k = leng; % An index to run through the event waveform
43      thermalFound = false;
44      captureFound = false;
45      tempPulse = [];
46
47      % Filter out events that are just noise
48      if max(evnt) > minCaptureHeight
49
50        % Main while loop to iterate the waveform. Start from right
             and go left
51        while ~thermalFound && k > 0
52
53          % The value of the kth point of the waveform. If it is
               greater than background, start recording
54          point = evnt(k);
```

```
55            if point > background
56              tempPulse(end+1) = point;
57
58            % Once it drops below background, and the recorded pulse is
                at least minPoints wide
59            elseif length(tempPulse) > minPoints
60
61              % Reverse tempPulse to get the height and index. Index is
                  for debugging
62              tempPulse = flip(tempPulse);
63              [peak, index] = max(tempPulse);
64              index += k;
65
66              % Find the half max of the peak. If that is less than
                  background, set it equal to background
67              halfMax = peak/2;
68              if halfMax <= background
69                halfMax = background;
70              end
71
72              % The indices where the temporary array is above the
                  halfmax (stores 0s and 1s)
73              % Then, find the indices where the array goes from 0 to 1
                  and then 1 to 0
74              aboveIndices = tempPulse >= halfMax;
75              left = find(diff(aboveIndices) == 1, 1, "first");
76              right = find(diff(aboveIndices) == -1, 1, "last");
77              width = length(tempPulse);
78
79              % If left and right are empty, just set the fwhm to the
                  width
80              % Otherwise, it is right minus left
81              if isempty(left) && isempty(right)
82                fwhm = width;
83              else
84                fwhm = right - left;
85              end
86
87              % Set the ratio for this pulse
88              ratio = fwhm/peak;
89
90              %i
91              %ratio
92              %peak
93              %width
94              %k
```

```
95
96              % if the capture gate has beend found, and the ratio is
                    less than the required ratio to be a thermal
97              if captureFound && ratio < thermalRatio
98                % A thermal has been found. Set its height as a peak
99                peaks(i) = peak;
100               thermalFound = true;
101
102               %i
103               %width
104               %k
105               %thermalFound
106
107               foundPulses(end+1) = i;
108
109             % Otherwise, if the pulse is wide enough and ratio large
                    enough
110             elseif width > minCaptureWidth && ratio > captureRatio &&
                     peak > minCaptureHeight
111               % A capture pulse has been found. Record this index for
                      debugging
112               captureFound = true;
113               %foundPulses(end+1) = i;
114             end
115
116             % Empty the array
117             tempPulse = [];
118           else
119
120             % Empty the array
121             tempPulse = [];
122           end
123
124           % Move one index to the left
125           k -= 1;
126         end
127       end
128     end
129   end
130
131   % Make sure to remove any zero peaks
132   peaks = peaks(peaks~=0);
133
134
135 %{
136 j=1;
```

```
endj = length(foundPulses)
while j<endj
  figure;
  for i = foundPulses(1:endj)
    V = (evnts(i).ch(:,cha))';
    x = 1:length(V);

    if mod(j,25)>0
      subplot(5,5,mod(j,25))
      plot(x,V)
      xlim([0,length(V)])
      title(["Plot ", num2str(i)])
    elseif mod(j,25) == 0
      subplot(5,5,25)
      plot(x,V)
      xlim([0,length(V)])
      title(["Plot ", num2str(i)])
    end

    j+=1
  end

  foundPulses = foundPulses(endj-1:end);
  pause(0.1)
end
%}


%V = (evnts(650).ch(:,cha))';
%x = 1:length(V);
%plot(x,V)


  %{
  disp("CaptureRatio mean, then std, then max, then min")
  mean(captureRatios)
  std(captureRatios)
  max(captureRatios)
  min(captureRatios)

  disp("ThermalRatio mean, then std, then max, then min")
  mean(thermalRatios)
  std(thermalRatios)
  max(thermalRatios)
  min(thermalRatios)
  %}
```

```
183
184 endfunction
```

Listing D.2 neutronProcesser function to process neutron waveforms to find the height of the thermal pulses.

Appendix E

Ion Beam Current Detection System

Included here is the circuit and code for the beam current detection. The vacuum chamber was grounded and the target holder was isolated from ground using Teflon and connected directly to the electrometer.

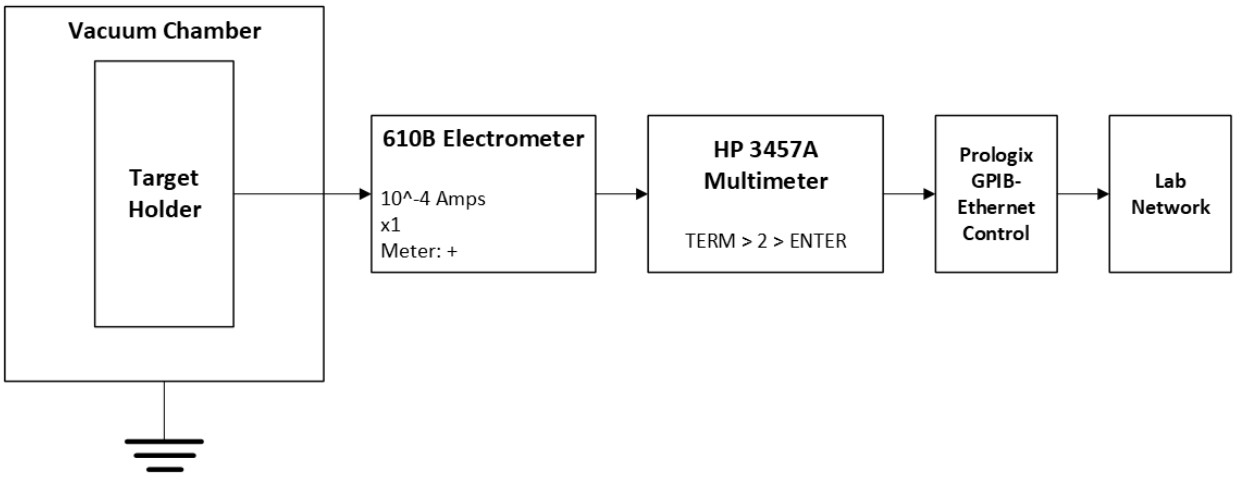

Figure E.1 The setup to measure the current on the target holder and send that signal to the lab computers

E.1 Octave Code To Read the Current

```
1
2  % hpReadCurrent
3  % Reads the current when the meter is set to the Amp setting and
       finds the
4  % average of the data
5  % Isaac Willden
6  % 24 Jan 2025
7  clear; close all;
8
9  % Before starting
10 % Make sure the HP meter is set to terminal 2 (TERM > 2 > ENTER)
11 % Make sure electrometer is set to Amps
12
13 % Configure c connection
14 c = connectPrologix();
15
16 % Variables
17 maxTime = 1800;
18 timerHandle = tic;
19 i = 1;
20 time = 0;
21
22 %For loop to display graph
23 while time < maxTime
24     % Pause and collect data
25     pause(0.7);
26     data = char(read(c));
27     data = strsplit(data);
28     Itemp = cellfun(@str2double,data);
29
30     % The code collected multiple samples at once. Take the average
31     aveITemp = mean(Itemp(~isnan(Itemp)));
32     I(i) = aveITemp;
33
34     % If the data isn't NaN, then plot it
35     if ~isnan(I(i))
36       time = toc(timerHandle);
37       t(i) = time;
38
39       plot(t,I,'-b')
40       pause(0.01)
41
42       title('Amps Graph')
```

```
43        ylabel('Amps') % Add 10^-x based on the scale of the
              electrometer
44        xlabel('Time (s)') % Figure out actual timing
45        hold on
46
47        i+=1;
48      else
49        I = I(~isnan(I));
50      end
51  end
52  hold off
53
54  % Close connection
55  clear c;
56
57  % Find the total average current
58  format long
59  Iave = mean(I)
```

E.1.1 The connectPrologix Function

Here is the code for the connectPrologix function that connects to the HP Multimeter via the GPIB-Ethernet connector.

```
1   % Configures the prologix connection
2   function [c] = connectPrologix()
3
4   % Load the instrument control package to get access to tcpclient
5   pkg load instrument-control
6
7   % Parameters
8   ipAddress = '192.168.1.156'; % IP of Prologix GPIB-Ethernet
        controller
9   port = 1234; % Default Prologix port
10
11  % Create TCP/IP object
12  c = tcpclient(ipAddress, port);
13  set(c, "Timeout", 10);
14  configureTerminator(c, 'LF');
15
16  % Set the address and have it beep so we know it's working
17  write(c, "++addr 6\n");
18  write(c, "BEEP\n");
19  pause(0.2);
```

```
20
21 % Set the proper modes to read commands
22 write(c, '++mode 1\n');
23 write(c, '++auto 1\n');
24 write(c, 'W4\n');
25
26 endfunction
```

Appendix F

The Gamma Detector Circuit

Included here is the gamma NIM module circuit to acquire data via the CAEN. We used similar code to the high energy electron detector to process the waveforms and plot the histogram.

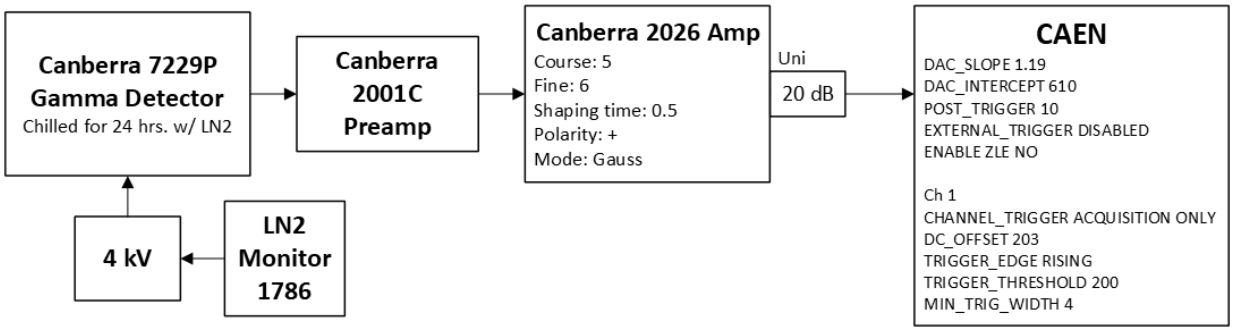

Figure F.1 Gamma detector circuit

Appendix G

Unified GUI For All Detectors

Included here is the code for the unified GUI written in Octave.

```
% One Program
% Takes all the detectors, plots them onto one screen while analyzing
    the data
% Started 6 May 2025

clear all; close all; clc;

 % Calibration data

 % Cobalt60 data for gamma calibration (E = m*x+b)
peak1Dig = 107.5;
peak1MeV = 1.1732;
peak2Dig = 122;
peak2MeV = 1.3325;

m = (peak2MeV-peak1MeV)/(peak2Dig-peak1Dig);
b = peak1MeV-m*peak1Dig;


% Am 241 for dEE Calibration
mdE = 3/1000; % TESTING!! %3.85;%4.995;%3.6617;  % Slope for
   calibtrating dE
bdE = 0/1000; %-224.38;%293.647;%236.288;
mE = 1/1000;

```

```
24
25 % Cs 137 for electron calibration
26 mElec = 0.624/390 * 7.63;
27
28
29
30   % Set up the figure and attributes
31
32 % Make the main figure
33 plotFont = 23;
34 fontSize = 16;
35 f = figure("name", "One Program");
36
37 % Stop button
38 % array after "Position" is formatted like [x-position,y-position,x-
       size,y-size]
39 global stop = false;
40 stop_btn = uicontrol("Style", "pushbutton", "String", "Stop", "Units"
       , "normalized", "Position", [0.02,0.85,0.07,0.1], "FontSize",
       fontSize);
41 set(stop_btn, "Callback", @(src,event) stop_callback());
42 function stop_callback()
43   disp("Ending program")
44   global stop;
45   stop = true;
46 end
47
48
49 % Veto count and status boxes
50 status = uicontrol("Style", "text", "String", "Vetos: \n\n Time (s):
       \n\nStatus: ", "Units", "normalized", "Position", [0.01, 0.68,
       0.09, 0.15], "FontSize", fontSize);
51 nCounter = uicontrol("Style", "text", "String", "Neutrons: ", "Units"
       , "normalized", "Position", [0.4, 0.9, 0.08, 0.05], "FontSize",
       fontSize);
52 gammaCounter = uicontrol("Style", "text", "String", "Gammas: ", "
       Units", "normalized", "Position", [0.85, 0.9, 0.08, 0.05], "
       FontSize", fontSize);
53 edeCounter = uicontrol("Style", "text", "String", 'dE-E: ', "Units",
       "normalized", "Position", [0.4, 0.605, 0.08, 0.05], "FontSize",
       fontSize);
54 electronCounter = uicontrol("Style", "text", "String", "e-: ", "Units
       ", "normalized", "Position", [0.85, 0.605, 0.08, 0.05], "FontSize"
       , fontSize);
55
56
```

```
57
58    % Initialize the plots
59
60  subplot(3,2,1)
61  title("Neutron (Channels)")
62  set(gca, "FontSize", plotFont)
63  subplot(3,2,2)
64  title("Gamma (MeV)")
65  set(gca, "FontSize", plotFont)
66  subplot(3,2,4);
67  title("Electron (MeV)")
68  set(gca, "FontSize", plotFont)
69
70  %Set up the live plot for dE-E
71  subplot(3,2,3);
72  eh = plot(0,0,"k.", "MarkerSize", 10);
73  title('\DeltaE-E', "Interpreter", "tex", "FontSize", plotFont)
74  xlabel("E (MeV)", "FontSize", plotFont)
75  ylabel('\DeltaE (MeV)', "Interpreter", "tex", "FontSize", plotFont)
76  vmax = 2048;
77  axis([0 vmax*mE 0 (vmax*mdE+bdE)]) %% TEST Make sure axes work right
78  set(gca, "FontSize", plotFont)
79
80  % Live plot for the electrometer/beam current
81  axCou = subplot(3,1,3);
82  couPlot = plot(0,0);
83  t = [];
84  C = [];
85  ci = 1;
86  cskip = 10;
87  title('Beam Current', 'FontSize', plotFont)
88  ylabel('uA', "FontSize", plotFont) % The title and scale here depends
        on what the Beam Current is set to
89  xlabel('Time (s)', "FontSize", plotFont)
90  set(gca, "FontSize", plotFont)
91
92  % Set the figure to full screen
93  screen = get(0, "screensize");
94  screen(1) = 1;
95  screen(2) = 1;
96  screen(4) -= 217;
97  set(f, "position", screen);
98
99
100
101   % Some Constants
```

```
102
103 % Channels
104 vet = 1;
105 dE = 3;
106 E = 4;
107 elec = 6;
108 g = 2;
109 n = 7;
110 ngate = 8;
111
112 % Other
113 file = "";
114
115 % These Thresholds Should Match What's in byuCAENcorderConfig.txt
116 vetoThreshold = 400;
117 nThresh = 0;
118 ngateThresh = 410;
119 gammaThresh = 200;
120 eThresh = 50;
121 electronThresh = 90;
122
123
124     % Some Variables
125
126 vetoCount = 0;
127 nCount = 0;
128 edeCount = 0;
129 gammaCount = 0;
130 electronCount = 0;
131
132 skip = 1; % How many times the plotting should skip per loop. Higher
        numbers make it run quicker
133 filenum = 1;
134 time = 0;
135 elecTime = 0;
136 oldTime = 0;
137 clockSpeed = 125*10^6; % Should be right based on my research and
        testing
138 loopQ = false;
139 tempEvent = [];
140
141 % Prepare to store peaks into these arrays
142 neutronPeaks = [];
143 gammaPeaks = [];
144 dEPeaks = [];
145 EPeaks = [];
```

```
146 elecPeaks = [];
147
148 % Get the directory
149 dirs = uigetdir('','Click on directory so file names show');
150
151
152 % EVENTS PER FILE
153 % KEEP THIS UPDATED WITH THE TXT DOCUMENT
154 eventsPerFile = 1000;
155
156
157   % Main While Loop
158 c = connectPrologix();
159 timerHandle = tic;
160 minCurrent = 18;
161 skip = ceil(eventsPerFile/2);
162 while eventsPerFile ~= 0
163
164     % Set the eventsPerFile to 0 to end the loop
165     if stop
166        eventsPerFile = 0;
167     end
168
169     % Update the electrometer coulomb graph
170     data = char(read(c));
171     data = strsplit(data);
172     Ctemp = cellfun(@str2double,data);
173     aveCTemp = mean(Ctemp(~isnan(Ctemp)));
174     C(ci) = aveCTemp*100; %check this!
175
176     % Beep if the current gets too low. Alarm mechanism
177     if C(ci) < minCurrent
178       write(c, "BEEP\n");
179       pause(0.1);
180     end
181
182     % If the data exists and isn't NaN
183     if ~isnan(C(ci))
184       elecTime = toc(timerHandle);
185       t(ci) = elecTime;
186
187       % Plot every cskip samples
188       if mod(ci,cskip) == 0
189          pause(0.01)
190          set(couPlot, "XData", t)
191          set(couPlot, "YData", C)
```

```
192           drawnow;
193         end
194
195         ci+=1;
196       else
197         C = C(~isnan(C));
198       end
199
200
201   % Open and read the file
202   [event,filenum] = combReadFile(filenum, dirs, eventsPerFile);
203
204   % Make sure the event structure is a structure we can use (this
        implies that we have new data)
205   if isstruct(event)
206
207     % For loop to loop through events in the structure
208     for i = 1:length(event)
209
210
211       % Update the electrometer coulomb graph (Needs to be done in
            and out of the loop to ensure constant reading)
212       data = char(read(c));
213       data = strsplit(data);
214       Ctemp = cellfun(@str2double,data);
215       aveCTemp = mean(Ctemp(~isnan(Ctemp)));
216       C(ci) = aveCTemp*100; % 100 converts from milivolts to volts,
            so
217
218       % Beep if the current gets too low. Alarm mechanism
219       if C(ci) < minCurrent
220          write(c, "BEEP\n");
221          pause(0.1);
222       end
223
224       % If the data exists and isn't NaN
225       if ~isnan(C(ci))
226         elecTime = toc(timerHandle);
227         t(ci) = elecTime;
228
229         % Plot every cskip samples
230         if mod(ci,cskip) == 0
231           pause(0.01)
232           set(couPlot, "XData", t)
233           set(couPlot, "YData", C)
234           %set(couPlot, 'xlim', [t(ci)-10,t(ci)+20]) % TESTING!!
```

```
235             drawnow;
236           end
237
238           ci+=1;
239         else
240           C = C(~isnan(C));
241         end
242
243
244         % Update the clock time (clock loops back from 2^32 to 2^31
                when it reaches there)
245         newTime = event(i).TrigTime/clockSpeed;
246         if oldTime < newTime
247           if ~loopQ
248             dt = newTime - time;
249           else
250             dt = newTime - oldTime;
251           end
252         else
253           dt = ((2^32-1)/clockSpeed - oldTime) + (newTime - (2^31-1)/
                clockSpeed);
254           loopQ = true;
255         end
256         time += dt;
257         oldTime = newTime;
258
259         % Update the counters
260         if mod(i,skip) == 0
261           set(status, "string", sprintf("Vetos: %d\n\nTime (s): %d\n\n%
                s", vetoCount, round(time), "Processing Data"));
262           set(nCounter, "string", sprintf("Neutrons: %d", nCount));
263           set(gammaCounter, "string", sprintf("Gammas: %d", gammaCount)
                );
264           set(edeCounter, "string", sprintf('dE-E: %d', edeCount));
265           set(electronCounter, "string", sprintf("e-: %d",
                electronCount));
266         end
267
268         % Get the veto event
269         veto = (event(i).ch(:,vet))';
270
271         % Get the size of the event array to find the number of
              channels
272         sizeEvents = size(event(i).ch);
273         numCh = sizeEvents(2);
274
```

```
275
276         % Processes as long as the veto is below a certain threshold
277         if max(veto) < vetoThreshold
278
279           % Neutron processing
280           if numCh >= n
281             gateMax = max((event(i).ch(:,ngate)));
282
283             if gateMax > ngateThresh
284               neutron = (event(i).ch(:,n))';
285               nPeak = combinedNeutronProcesser(neutron);
286
287               if nPeak > nThresh
288                 pause(0.05)
289                 neutronPeaks(end+1) = nPeak;
290                 neutronPeaks = neutronPeaks(neutronPeaks~=0);
291                 nCount += 1;
292               end
293             end
294
295             % Skip every few plots for efficiency
296             if mod(i,skip) == 0 && length(neutronPeaks) > 0
297               pause(0.1)
298               subplot(3,2,1)
299               hist(neutronPeaks, 500)
300               title("Neutron (Channels)")
301               set(gca, "FontSize", plotFont)
302             end
303           end
304
305           % Gamma processing
306           if numCh >= g
307             gamma = (event(i).ch(:,g))';
308             maxGamma = max(gamma);
309
310             if maxGamma > gammaThresh
311               baseline = mode(gamma);
312               peak = maxGamma - baseline;
313               gammaPeaks(end+1) = m*peak+b;
314               gammaCount +=1;
315             end
316
317             if mod(i,skip) == 0 && length(gammaPeaks) > 0
318               subplot(3,2,2);
319               centers = linspace(0,2048*m+b,2048);
320               counts = hist(gammaPeaks,centers);
```

```
321             bar(centers, counts)
322             title("Gamma (MeV)")
323             xlim([0 max(gammaPeaks)]);
324             set(gca, "FontSize", plotFont)
325             pause(0.05)
326           end
327         end
328
329         % Electron processing
330         if numCh >= elec
331           electron = (event(i).ch(:,elec))';
332
333           if max(electron) > electronThresh
334             elecPeaks(end+1) = max(electron)*mElec;
335             electronCount +=1;
336           end
337
338           if mod(i,skip) == 0 && length(elecPeaks) > 0
339             pause(0.05)
340             subplot(3,2,4);
341             hist(elecPeaks, 500);
342             title("Electron (MeV)")
343             xlim([0 25]);
344             set(gca, "FontSize", plotFont)
345           end
346         end
347
348         % dE-E processing
349         if numCh >= E
350           Energy = (event(i).ch(:,E))';
351
352           if max(Energy) > eThresh
353             EPeaks(end+1) = max(Energy)*mE;
354
355             deltaEnergy = (event(i).ch(:,dE))';
356             dEPeaks(end+1) = max(deltaEnergy)*mdE+bdE;
357
358             edeCount +=1;
359           end
360
361           if mod(i,skip) == 0
362             set(eh, "XData", EPeaks)
363             set(eh, "YData", dEPeaks)
364             set(eh, "Color", [0,0.75,0.25]);
365             drawnow;
366           end
```

```
367         end
368
369       else
370         % Since the signal was vetoed, up the count
371         vetoCount += 1;
372       end
373
374     end
375
376   else
377     % Set the status box to read "No New Data"
378     set(status, "string", sprintf("Vetos: %d\n\nTime (s): %d\n\n%s",
          vetoCount, round(time), "No New Data"));
379   end
380
381    pause(0.01)
382 end
383
384
385 % Plot everything one more time after the program stopped
386 if length(EPeaks) > 0
387   set(eh, "XData", EPeaks)
388   set(eh, "YData", dEPeaks)
389   set(eh, "Color", [0,0.75,0.25]);
390   drawnow;
391 end
392
393 if length(elecPeaks) > 0
394   subplot(3,2,4);
395   hist(elecPeaks, 500);
396   title("Electron (MeV)")
397   xlim([0 25]);
398   set(gca, "FontSize", plotFont)
399 end
400
401 if length(gammaPeaks) > 0
402   subplot(3,2,2);
403   centers = linspace(0,2048*m+b,2048);
404   counts = hist(gammaPeaks,centers);
405   bar(centers, counts)
406   title("Gamma (MeV)")
407   xlim([0 max(gammaPeaks)]);
408   set(gca, "FontSize", plotFont)
409 end
410
411 if length(neutronPeaks) > 0
```

```
412   subplot(3,2,1)
413   hist(neutronPeaks, 500)
414   title("Neutron (Channels)")
415   set(gca, "FontSize", plotFont)
416 end
417
418
419 % Write a beamCurrent csv file to save for later
420 if length(C) > 0
421   data = [t',C'];
422   csvwrite('beamCurrent.csv',data);
423 end
424
425
426 % Set the status and counter boxes one more time
427 set(status, "string", sprintf("Vetos: %d\n\nTime (s): %d\n\n%s",
      vetoCount, round(time), "Program Stopped"));
428 set(nCounter, "string", sprintf("Neutrons: %d", nCount));
429 set(gammaCounter, "string", sprintf("Gammas: %d", gammaCount));
430 set(edeCounter, "string", sprintf('dE-E: %d', edeCount));
431 set(electronCounter, "string", sprintf("Electrons: %d", electronCount
      ));
432
433
434 % Clear the connection to the electrometer
435 clear c;
```

Listing G.1 Unified GUI Code.

Appendix H

Using CAD-Based Files in Geant4 Gears

To aid in future Geant4 simulations, the author figured out, using various open source tools, how to convert STEP files to GDML files, which can be read directly in the .mac file (instead of using a .tg file). Geant4 must be compiled correctly with GDML compatibility (and XERCESC) in order for it to work.

Here are the steps to follow to convert the file and run the simulation:

1. Change the .step file variable in "convert.py" to the name of your step file
2. Open an anaconda prompt and navigate to the folder of your step file
3. Activate the geo environment with the command "conda activate geo"
4. Run your .mac file with the command "gears [name].mac" (include the line /geometry/source output.gdml before /run/initialize)

This appendix also includes the convert.py and example main.mac files.

H.1 Converting Python Script

This is the code primarily written by AI that converts STEP files to GDML files. To use, one must change the STEP_FILE to the name of the step file they wish to simulate. Change the PART_MAT variable in order to change the material used.

Listing H.1 CAD to GDML conversion code.

```
import pyg4ometry as pg4
from pyg4ometry.geant4 import PhysicalVolume

# ============================================================
# CONFIG
# ============================================================

STEP_FILE   = "sample.step"
OUTPUT_GDML = "output.gdml"

PART_MAT = "G4_Al"

WORLD_SIZE_MM = 200  # 10 m cube

# ============================================================
# READ STEP
# ============================================================

print("\nReading STEP file...")

reader = pg4.pyoce.Reader(STEP_FILE)

free_shapes = list(reader.freeShapes())

print(f"\nFound {len(free_shapes)} top-level shape(s):")

for i, label in enumerate(free_shapes):

    name = pg4.pyoce.pythonHelpers.get_TDataStd_Name_From_Label(label
       )

    print(f"  {i+1}: '{name}'")

if len(free_shapes) == 0:
    raise RuntimeError("No shapes found in STEP file.")

```

```
root_name = pg4.pyoce.pythonHelpers.get_TDataStd_Name_From_Label(
    free_shapes[0]
)

print(f"\nUsing root shape: '{root_name}'")

# ============================================================
# CONVERT STEP -> GEANT4 REGISTRY
# ============================================================

print("\nConverting STEP to Geant4 geometry...")

reg = pg4.convert.oce2Geant4(
    reader.shapeTool,
    root_name,
    materialMap={},
    labelToSkipList=[],
    meshQualityMap={},
    oceName=True
)

# ============================================================
# RENAME LOGICAL VOLUMES CLEANLY
# ============================================================

print("\nRenaming logical volumes...")

new_lv_dict = {}

part_counter = 0

for old_name, lv in reg.logicalVolumeDict.items():

    if old_name == "wl":
        new_lv_dict["wl"] = lv
        continue

    new_name = f"part_{part_counter}"

    print(f"  '{old_name}' -> '{new_name}'")

    lv.name = new_name

    new_lv_dict[new_name] = lv

    part_counter += 1
```

```
82
83  reg.logicalVolumeDict = new_lv_dict
84
85  # ============================================================
86  # MATERIALS
87  # ============================================================
88
89  print("\nAssigning materials...")
90
91  galactic = pg4.geant4.MaterialPredefined(
92      "G4_Galactic",
93      reg
94  )
95
96  part_material = pg4.geant4.MaterialPredefined(
97      PART_MAT,
98      reg
99  )
100
101 for name, lv in reg.logicalVolumeDict.items():
102
103     if name == "wl":
104         lv.material = galactic
105     else:
106         lv.material = part_material
107
108 # ============================================================
109 # REPLACE WORLD VOLUME WITH LARGE BOX
110 # ============================================================
111
112 world_lv = reg.logicalVolumeDict["wl"]
113
114 print("\nOriginal world solid:")
115 print(world_lv.solid)
116
117 big_world = pg4.geant4.solid.Box(
118     "bigWorldSolid",
119     WORLD_SIZE_MM,
120     WORLD_SIZE_MM,
121     WORLD_SIZE_MM,
122     reg
123 )
124
125 world_lv.solid = big_world
126
127 print(f"\nWorld replaced with {2*WORLD_SIZE_MM} mm cube.")
```

```
128
129 # ============================================================
130 # REMOVE AUTO-GENERATED DAUGHTERS
131 # ============================================================
132
133 print("\nRemoving auto-generated daughter placements...")
134
135 world_lv.daughterVolumes = []
136
137 # ============================================================
138 # EXPLICITLY PLACE IMPORTED PARTS
139 # ============================================================
140
141 print("\nCreating explicit physical placements...")
142
143 placed_count = 0
144
145 for name, lv in reg.logicalVolumeDict.items():
146
147     if name == "wl":
148         continue
149
150     print(f"  Placing {name} at origin")
151
152     PhysicalVolume(
153         [0, 0, 0],    # rotation
154         [0, 0, 0],    # translation (mm)
155         lv,
156         f"{name}_pv",
157         world_lv,
158         reg
159     )
160
161     placed_count += 1
162
163 print(f"\nPlaced {placed_count} physical volume(s).")
164
165 # ============================================================
166 # DIAGNOSTIC: BOUNDING BOXES
167 # ============================================================
168
169 print("\nBounding boxes:")
170
171 for name, lv in reg.logicalVolumeDict.items():
172
173     if name == "wl":
```

```
174             continue
175
176         try:
177
178             mesh = lv.solid.mesh()
179
180             bb = mesh.getBoundingBox()
181
182             print(f"\n{name}")
183             print(f"  xmin = {bb[0]}")
184             print(f"  xmax = {bb[1]}")
185             print(f"  ymin = {bb[2]}")
186             print(f"  ymax = {bb[3]}")
187             print(f"  zmin = {bb[4]}")
188             print(f"  zmax = {bb[5]}")
189
190         except Exception as e:
191
192             print(f"\n{name}")
193             print("  Failed bounding box extraction")
194             print(f"  {e}")
195
196 # ============================================================
197 # OPTIONAL: TEST TESS CLOSURE
198 # ============================================================
199
200 print("\nTesting tessellated solid closure:")
201
202 for name, lv in reg.logicalVolumeDict.items():
203
204     if name == "wl":
205         continue
206
207     print(f"\n{name}")
208
209     try:
210         print("  Solid type:", type(lv.solid))
211
212         if hasattr(lv.solid, "isClosed"):
213             print("  Closed:", lv.solid.isClosed())
214         else:
215             print("  Closure test unavailable")
216
217     except Exception as e:
218         print("  Closure test failed")
219         print(f"  {e}")
```

```
220
221 # ============================================================
222 # OPTIONAL: VTK VIEWER
223 # ============================================================
224
225 print("\nLaunching VTK viewer...")
226
227 try:
228
229     viewer = pg4.visualisation.VtkViewer()
230
231     viewer.addLogicalVolume(
232         reg.getWorldVolume()
233     )
234
235     viewer.view()
236
237 except Exception as e:
238
239     print("\nVTK viewer failed:")
240     print(e)
241
242 # ============================================================
243 # WRITE GDML
244 # ============================================================
245
246 print("\nWriting GDML...")
247
248 writer = pg4.gdml.Writer()
249
250 writer.addDetector(reg)
251
252 writer.write(OUTPUT_GDML)
253
254 print(f"\nDone.")
255 print(f"GDML written to: {OUTPUT_GDML}")
```

H.2 Main Macro File

The example Gears .mac file that uses a GDML file for the geometry.

```
1 # print macro commands on screen
2 /control/verbose 1
3
```

```
# geometry must be specified before /run/initialize
/geometry/source output.gdml #det.tg

# Starts the simulation and ensures that there are no overlapping
   volumes
/run/initialize
/geometry/test/run

# Prepares to be saved as a WRL file
/vis/open VRML2FILE
/vis/viewer/zoom 1
/vis/viewer/set/background white ! ! 0
/vis/drawVolume

# Starts the data collection and trajectory storing
# Trajectories can be rich or smooth
/vis/scene/add/trajectories rich
/vis/scene/endOfEventAction accumulate

# Protons
/gps/source/intensity 1
/gps/particle proton
/gps/pos/centre 0 0 2 cm
/gps/direction 0 0 1
/gps/ang/type iso
/gps/ene/type Mono
/gps/ene/mono  3 MeV

# Generates [number] particles from the General Particle Source (gps)
/run/beamOn 200
```

Appendix I

Repairing Old CAD Files

The lab computers have RSDOC files that, due to an unfortunate DesignSpark update, are unusable. This chapter outlines the development of the procedure and code used to extract the 3D files and repair them for future use.

For these examples, the original file will be called "example.rsdoc" and will exist in the main directory.

I.1 Extracting and Converting the SAB Files

1. Rename the file "example.zip"
2. Select the file, then "Extract all"
3. Enter the extracted folder, then go to **SpaceClaim -> Geometry** and verify that there are SAB files
    (a) If there are multiple RSDOCs, do steps 1-3 for all of them
4. Go back to the main directory and create a file called **convert_all.py**.

5. Copy and paste the "Convert All" code listed later in this appendix, then run it.

    (a) This should convert all of the SAB files into SAT files and organize them under **SAT Files**

I.2 SAT to STEP Conversion

1. Enter the **SAT Files** directory

2. Highlight all the SAT files and drag them into **ABViewer** (30 day trial). Wait for them to load

3. Create a file called **auto_stp.py** and copy and paste the code from later in this appendix

4. Open a command terminal in the same directory

    (a) Check to see if **pyautogui** is installed via the command "pip show pyautogui"

    (b) If it is not installed, use the command "pip install pyautogui"

5. Save a test file in ABViewer to the desired directory. This will set where the STEP files are saved

6. Run the script in the command terminal with the command "python auto_stp.py"

    (a) Have the very first file selected in ABViewer. Make sure to click into ABViewer immediately after you start the script. This will set it as the active window for the python script to function

7. End the script by selecting the command prompt and pressing **Ctrl+C** or by closing the command prompt

(a) The Python code will automatically save the files in order. It will not automatically stop when it loops around to the first file again.

I.3 Importing to Spaceclaim

Most of the STEP files should now work to be dragged straight into SpaceClaim. The position information is not usually preserved very well, so the objects may need to be realigned. There will likely be some STEP files that do not load and lead to errors in SpaceClaim. Move those files to a separate folder for futher repair.

Follow these steps to repair those broken STEP files one at a time:

1. Open **FreeCAD**
2. Press **Ctrl+N** to create a new document
3. Drag a STEP file into **FreeCAD** or press **Ctrl+O** to choose the file
4. Press **Ctrl+A** then **Ctrl+E** to export it. Save it as a STEP file
   (a) This resulting STEP file should work in SpaceClaim now
5. Repeat as needed for the remaining

A similar Python script can be made to automate this process, but there are usually not very many corrupted STEP files to justify this.

I.4 Convert All Python Code

The code for **convert_all.py** used in this process.

```python
import os
import subprocess

BASE_DIR = os.path.abspath("")
CONVERTER = os.path.join(BASE_DIR, "SabSatConverter.exe")
OUTPUT_DIR = os.path.join(BASE_DIR, "SAT Files")

os.makedirs(OUTPUT_DIR, exist_ok=True)

def find_all_sab_files(base_dir):
    sab_files = []
    for root, _, files in os.walk(base_dir):
        for f in files:
            if f.lower().endswith(".sab"):
                sab_files.append(os.path.join(root, f))
    return sab_files


def get_grandparent_name(path):
    # Conversion / Grandparent / ... / file.sab
    parts = os.path.normpath(path).split(os.sep)

    # Find "Conversion" and take next folder as grandparent
    try:
        idx = parts.index("Conversion")
        return parts[idx + 1]
    except ValueError:
        return "Unknown"


def main():
    if not os.path.exists(CONVERTER):
        raise FileNotFoundError(f"Converter not found: {CONVERTER}")

    sab_files = find_all_sab_files(BASE_DIR)

    print(f"FOUND {len(sab_files)} .sab files\n")

    counters = {}

    for sab in sab_files:
        # figure out grandparent folder name
        rel_path = os.path.relpath(sab, BASE_DIR)
        parts = rel_path.split(os.sep)

        grandparent = parts[0] if len(parts) > 1 else "Unknown"
```

```
47
48          counters.setdefault(grandparent, 0)
49          counters[grandparent] += 1
50          idx = counters[grandparent]
51
52          output_name = f"{grandparent}-{idx}.sat"
53          output_path = os.path.join(OUTPUT_DIR, output_name)
54
55          cmd = [
56              CONVERTER,
57              "-i", sab,
58              "-o", output_path
59          ]
60
61          print(f"Converting: {sab}")
62          print(f"       -> {output_path}")
63
64          result = subprocess.run(cmd, capture_output=True, text=True)
65
66          if result.returncode != 0:
67              print(" FAILED:")
68              print(result.stderr)
69          else:
70              print(" Done\n")
71
72
73  if __name__ == "__main__":
74      main()
```

I.5 Auto STEP Code

This is the Python code that automatically saves the SAT files as STEP files in ABViewer. The sleep times may need to be adjusted based on the speed of the computer.

```
1  import pyautogui
2  import time
3
4  print("Starting in 5 seconds...")
5  time.sleep(5)
6
7  while True:
8      pyautogui.hotkey('ctrl', 's')
9      time.sleep(3)
```

```
10
11      pyautogui.press('tab')
12      time.sleep(0.1)
13
14      pyautogui.press('s')
15      time.sleep(0.1)
16
17      pyautogui.press('enter')
18      time.sleep(1)
19
20      pyautogui.hotkey('ctrl', 'tab')
21      time.sleep(1)
```

Appendix J

Designing the Filament Energized Nuclear Ion Current Source (FENICS)

Since a higher current ion beam would be needed, a new ion gun would need to be fabricated. The goal was for this to reach on the order of 20 mA of ion current, a thousandfold increase from the original ion gun. This was dubbed the Filament Energized Nuclear Ion Current Source or FENICS. Its design is based on the DC25 by Oxford used in Forbes' research [15]. In the development, the device was designed in a CAD software before submitting the parts to be machined. Simultaneously, the ions were simulated first with MATLAB, then via IBSimu, a C-based software that solves for the trajectory of charged particles. This aided in the design and validation for the FENICS. Generative AI was used to aid the development of the code in this appendix.

J.1 CAD Design

SpaceClaim was used to design the 3D models for the FENICS. See figures J.1, J.2, and J.3.

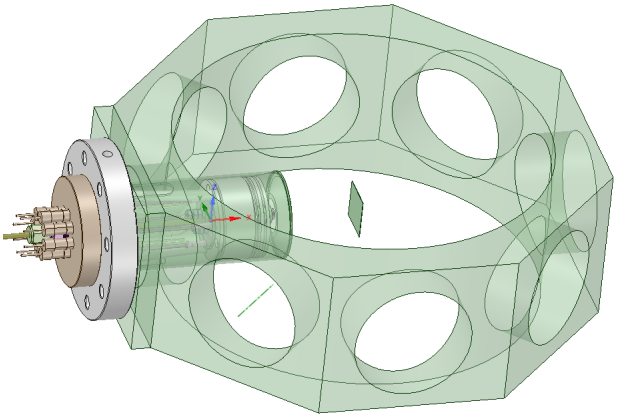

Figure J.1 The 3D CAD model of the FENICS. Note the octagonal shell of the vacuum chamber with the square target in the center.

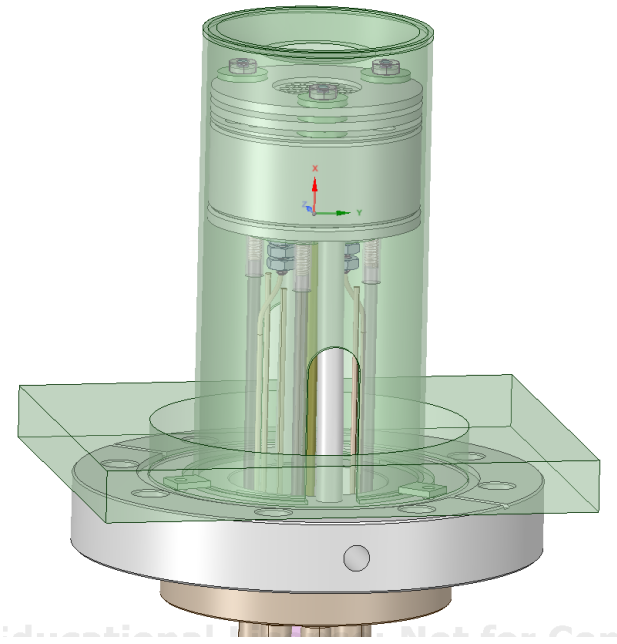

Figure J.2 A closer view of the 3D model of FENICS.

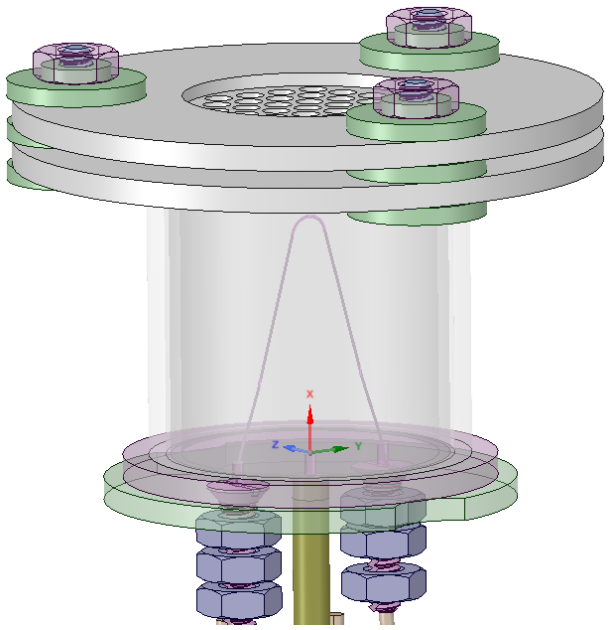

Figure J.3 A zoomed-in view of the FENICS 3D model without the shell. Note the two grids at the top and the filament below. The filament is surrounded by a cylindrical anode that is insulated by glass.

J.2 Simulating Ion Trajectories in Electromagnetic Fields

In preparation for assembling a high-current ion gun, the author worked with generative AI to write a simulation for ion trajectories in electric and magnetic fields. This involves the magnetic fields from 9 neodymium magnets in a ring around a cylinder held at a high positive voltage. Inside the cylinder is a curved filament held at a high negative voltage. This filament emits electrons that accelerate to the outer cylinder, ionizing atoms along the way.

The conclusion from this simulation was that the orientation of the bar magnets makes no significant difference to the trajectory of the ion.

J.3 MATLAB Code For Simulation

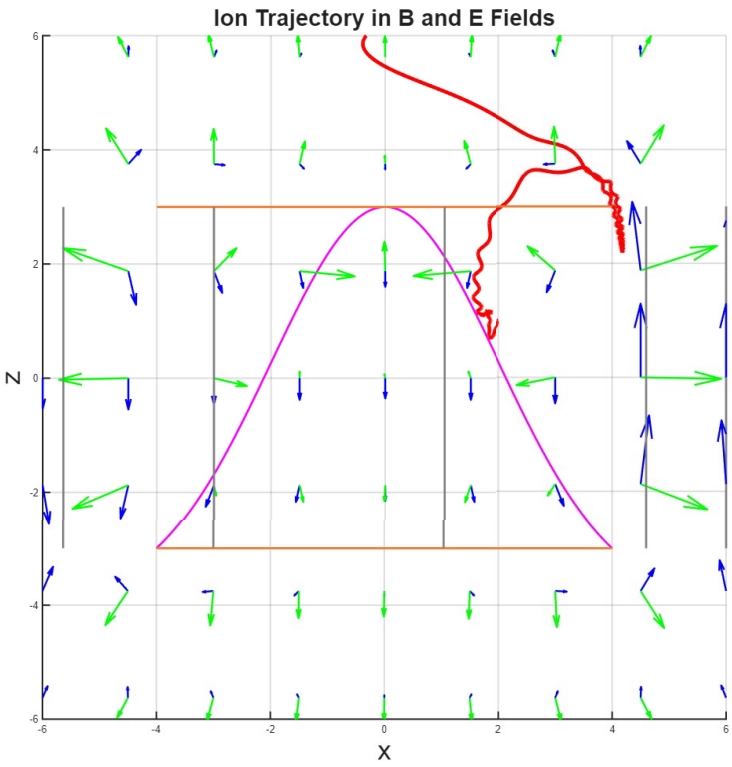

Figure J.4 Ion simulation of a positive ion in the ion gun. The gray lines are the center lines of the bar magnets. Orange is the top and bottom of the cylinder. Magenta is the curved filament. Green arrows signify electric field vectors and blue arrows are magnetic field vectors. The red line is the trajectory of the ion, which starts at (x,y,z)=(2,0,1). Although it does not exit the cylinder upwards properly, the accelerator grid (not simulated) would likely have propelled it much faster to the exit, preventing as much of a winding path.

```
%% Code written with assistance from generative AI
%% Ion in a magnetic and electric field (Arbitrary units)

clear; clc; close all

% Parameters (normalized)
q = 1;            % charge
m = 2;            % mass (2 amu -> 2)

% Initial conditions
x0 = 2;
y0 = 0;
z0 = 1;

vx0 = 0;        % small transverse velocity
vy0 = 0;
vz0 = 0;          % mostly along +z

state0 = [x0 y0 z0 vx0 vy0 vz0];


%% Solve ODE

% Time span
tspan = [0 50];

% Integrate Lorentz force
opts = odeset('RelTol',1e-6,'AbsTol',1e-8); % Lower tolerances later!
[t,state] = ode45(@(t,s) lorentzODE(t,s,q,m), tspan, state0, opts);

x = state(:,1);
y = state(:,2);
z = state(:,3);

% Final position
xf = x(end);
yf = y(end);
zf = z(end);


%% Plot particle trajectory


hold on
plot3(x,z,y,'r','LineWidth',2)

```

```
grid on
axis equal
xlabel('x')
ylabel('z')
zlabel('y')
title('Ion Trajectory')

% Plot final point as a red ball
plot3(xf, zf, yf, 'ro', ...
      'MarkerSize', 10, ...
      'MarkerFaceColor', 'r')
hold off

%% Plot stuff (magnets, cylinder, etc) and fields
hold on
plotStuff()
plotFieldVectors()
hold off



%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Local functions (MATLAB allows these below the script)
%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function dsdt = lorentzODE(~,s,q,m)
    x  = s(1); y  = s(2); z  = s(3);
    vx = s(4); vy = s(5); vz = s(6);

    [Bx,By,Bz] = solenoidField(x,y,z);
    [Ex,Ey,Ez] = eField(x,y,z);

    v = [vx; vy; vz];
    E = [Ex; Ey; Ez];
   %B = [Bx; By; Bz];

    a = (q/m) * (E);% + cross(v,B));

    dsdt = [vx; vy; vz; a];
end

function Edata = precomputeEField(x,y,z)
```

```
89
90      % ---------- Grid ----------
91      [X,Y,Z] = meshgrid(x,y,z);
92
93      Ex = zeros(size(X));
94      Ey = zeros(size(X));
95      Ez = zeros(size(X));
96
97      for i = 1:numel(X)
98          [Ex(i),Ey(i),Ez(i)] = eField(X(i),Y(i),Z(i));
99      end
100
101     % Store everything
102     Edata.Ex = Ex;
103     Edata.Ey = Ey;
104     Edata.Ez = Ez;
105 end
106
107 function [Ex,Ey,Ez] = eField(x,y,z)
108     % Exact finite conducting cylinder + wire + disk via Coulomb
            integration
109     % ---------- Initialize ----------
110     Ex = 0; Ey = 0; Ez = 0;
111
112     % ---------- Voltages ----------
113     Vcyl = 2;       % cylinder voltage
114     Vdisk = 10;     % disk voltage (negative)
115     eps0 = 1;
116
117
118     %{
119     % ---------- Cylinder geometry ----------
120     Rc = 4;
121     L  = 3;
122
123     % ---------- Cylinder discretization ----------
124     Nphi = 80;
125     Nz   = 60;
126
127     phi = linspace(0,2*pi,Nphi);
128     zs  = linspace(-L,L,Nz);
129
130     dphi = phi(2)-phi(1);
131     dz   = zs(2)-zs(1);
132
133
```

```
134     % ---------- Cylinder Coulomb integral ----------
135     sigma_cyl = 1;
136     Ccyl = sigma_cyl / (4*pi*eps0);
137
138     for i = 1:Nphi
139         for k = 1:Nz
140             xs = Rc*cos(phi(i));
141             ys = Rc*sin(phi(i));
142             zs_s = zs(k);
143
144             dA = Rc * dphi * dz;
145
146             rx = x - xs;
147             ry = y - ys;
148             rz = z - zs_s;
149
150             r3 = (rx^2 + ry^2 + rz^2)^(3/2) + 1e-12;
151
152             Ex = Ex + Ccyl * rx / r3 * dA;
153             Ey = Ey + Ccyl * ry / r3 * dA;
154             Ez = Ez + Ccyl * rz / r3 * dA;
155         end
156     end
157
158     % ---------- Normalize cylinder to Vcyl ----------
159     rtest = Rc * 0.95;
160     [Er,~,~] = eField_raw(rtest,0,0,Rc,L);
161
162     Vcalc = abs(Er) * Rc;
163     scale_cyl = Vcyl / Vcalc;
164
165     Ex = scale_cyl * Ex;
166     Ey = scale_cyl * Ey;
167     Ez = scale_cyl * Ez;
168     %}
169
170     % ---------- Add wire contribution ----------
171     [Exw,Eyw,Ezw] = wireField(x,y,z);
172
173     Ex = Ex + Exw;
174     Ey = Ey + Eyw;
175     Ez = Ez + Ezw;
176
177     % ---------- Add disk contribution ----------
178     [Exd,Eyd,Ezd] = diskField(x,y,z,Vdisk);
179
```

```
    Ex = Ex + Exd;
    Ey = Ey + Eyd;
    Ez = Ez + Ezd;
end

function [Ex,Ey,Ez] = eField_raw(x,y,z,Rc,L)
    % Same as above but without normalization
    eps0 = 1;
    sigma = 1;
    C = sigma / (4*pi*eps0);

    Nphi = 80;
    Nz   = 60;

    phi = linspace(0,2*pi,Nphi);
    zs  = linspace(-L,L,Nz);

    dphi = phi(2)-phi(1);
    dz   = zs(2)-zs(1);

    Ex = 0; Ey = 0; Ez = 0;

    for i = 1:Nphi
        for k = 1:Nz
            xs = Rc*cos(phi(i));
            ys = Rc*sin(phi(i));
            zs_s = zs(k);

            dA = Rc * dphi * dz;

            rx = x - xs;
            ry = y - ys;
            rz = z - zs_s;

            r3 = (rx^2 + ry^2 + rz^2)^(3/2) + 1e-12;

            Ex = Ex + C * rx / r3 * dA;
            Ey = Ey + C * ry / r3 * dA;
            Ez = Ez + C * rz / r3 * dA;
        end
    end
end

function [Ex,Ey,Ez] = diskField(x,y,z,Vdisk)
    % Finite conducting disk via Coulomb surface integration

```

```
226     % ---------- Geometry ----------
227     Rd = 3.5;      % disk radius
228     Dd = 5;      % disk z-position (above cylinder)
229
230     % ---------- Discretization ----------
231     Nr   = 40;
232     Nphi = 80;
233
234     r   = linspace(0,Rd,Nr);
235     phi = linspace(0,2*pi,Nphi);
236
237     dr   = r(2)-r(1);
238     dphi = phi(2)-phi(1);
239
240     % ---------- Constants ----------
241     eps0 = 1;
242     sigma = -1;    % unnormalized negative surface charge
243     C = sigma / (4*pi*eps0);
244
245     % ---------- Initialize ----------
246     Ex = 0; Ey = 0; Ez = 0;
247
248     % ---------- Coulomb integral ----------
249     for i = 1:Nr
250         for j = 1:Nphi
251             xs = r(i)*cos(phi(j));
252             ys = r(i)*sin(phi(j));
253             zs = Dd;
254
255             dA = r(i) * dr * dphi;
256
257             rx = x - xs;
258             ry = y - ys;
259             rz = z - zs;
260
261             r3 = (rx^2 + ry^2 + rz^2)^(3/2) + 1e-12;
262
263             Ex = Ex + C * rx / r3 * dA;
264             Ey = Ey + C * ry / r3 * dA;
265             Ez = Ez + C * rz / r3 * dA;
266         end
267     end
268
269     % ---------- Normalize disk to Vdisk ----------
270     % Sample point just above disk center
271     ztest = Dd + 0.1;
```

```
    rtest = 0;

    [~,~,Ez_test] = diskField_raw(rtest,0,ztest,Rd,Dd);

    Vcalc = abs(Ez_test) * Rd;
    scale = Vdisk / Vcalc;

    Ex = scale * Ex;
    Ey = scale * Ey;
    Ez = scale * Ez;
end

function [Ex,Ey,Ez] = diskField_raw(x,y,z,Rd,Dd)
    eps0 = 1;
    sigma = -1;
    C = sigma / (4*pi*eps0);

    Nr = 40;
    Nphi = 80;

    r = linspace(0,Rd,Nr);
    phi = linspace(0,2*pi,Nphi);

    dr = r(2)-r(1);
    dphi = phi(2)-phi(1);

    Ex = 0; Ey = 0; Ez = 0;

    for i = 1:Nr
        for j = 1:Nphi
            xs = r(i)*cos(phi(j));
            ys = r(i)*sin(phi(j));
            zs = Dd;

            dA = r(i)*dr*dphi;

            rx = x - xs;
            ry = y - ys;
            rz = z - zs;

            r3 = (rx^2 + ry^2 + rz^2)^(3/2) + 1e-12;

            Ex = Ex + C * rx / r3 * dA;
            Ey = Ey + C * ry / r3 * dA;
            Ez = Ez + C * rz / r3 * dA;
        end
```

```
    end
end

function [Ex,Ey,Ez] = wireField(x,y,z)
    % Negatively charged curved wire

    % ---------- Geometry ----------
    Rc = 4;
    z0 = -3;          % bottom of cylinder
    A  = 6;           % height rise (goes near top)
    sigma = Rc/2;     % Gaussian width

    % ---------- Charge ----------
    lambda = -1;      % negative line charge
    eps0 = 1;
    C = lambda / (4*pi*eps0);

    % ---------- Discretization ----------
    Nx = 200;
    xs = linspace(-Rc, Rc, Nx);
    dx = xs(2)-xs(1);

    % ---------- Initialize ----------
    Ex = 0; Ey = 0; Ez = 0;

    % ---------- Line integral ----------
    for i = 1:Nx
        xw = xs(i);
        yw = 0;
        zw = z0 + A*exp(-xw^2/(2*sigma^2));

        rx = x - xw;
        ry = y - yw;
        rz = z - zw;

        r3 = (rx^2 + ry^2 + rz^2)^(3/2) + 1e-12;

        Ex = Ex + C * rx / r3 * dx;
        Ey = Ey + C * ry / r3 * dx;
        Ez = Ez + C * rz / r3 * dx;
    end
end

function [Bx,By,Bz] = solenoidField(x,y,z)
    % Ring of 9 finite solenoids via exact Biot-Savart
    % All solenoids aligned with z-axis
```

```
    % Ring centered at origin

    % ---------- Solenoid parameters ----------
    R = 2;          % solenoid radius
    L = 3;          % half-length
    Nloops = 60;    % loops per solenoid
    I = -5;          % current
    mu0 = 1;        % normalized units (mu0 / 4pi absorbed below)

    % ---------- Ring parameters ----------
    Nsol = 9;       % number of solenoids
    A = 6;          % radius of solenoid ring

    % ---------- Discretization ----------
    zs = linspace(-L, L, Nloops);
    dphi = linspace(0, 2*pi, 90);
    dphi_step = dphi(2) - dphi(1);

    % ---------- Initialize field ----------
    Bx = 0; By = 0; Bz = 0;

    % Biot-Savart constant
    C = mu0 * I / (4*pi);

    % ---------- Loop over solenoids ----------
    for n = 1:Nsol
        theta = 2*pi*(n-1)/Nsol;

        % Solenoid center in x-y plane
        x0 = A*cos(theta);
        y0 = A*sin(theta);

        % ---------- Loop over current loops ----------
        for k = 1:length(zs)
            zk = zs(k);

            % ---------- Loop over loop segments ----------
            for m = 1:length(dphi)-1
                phi = 0.5*(dphi(m) + dphi(m+1));

                % Source point (shifted solenoid)
                xs = x0 + R*cos(phi);
                ys = y0 + R*sin(phi);
                zs_seg = zk;

                % dl vector
```

```
                dl = R * [-sin(phi); cos(phi); 0] * dphi_step;

                % Vector to field point
                rx = x - xs;
                ry = y - ys;
                rz = z - zs_seg;

                r3 = (rx^2 + ry^2 + rz^2)^(3/2) + 1e-12;

                % Biot-Savart contribution
                dB = C * cross(dl, [rx; ry; rz]) / r3;

                Bx = Bx + dB(1);
                By = By + dB(2);
                Bz = Bz + dB(3);
            end
        end
    end
end

function plotFieldVectors()

    % ---------- Grid (x-z slice, y = 0) ----------
    [x,~,z] = meshgrid( ...
        linspace(-12,12,17), ...
        linspace(-12,12,17),...
        linspace(-15,15,19));
    y = zeros(size(x));

    % ---------- Allocate fields ----------
    Bx = zeros(size(x));
    By = zeros(size(x));
    Bz = zeros(size(x));

    Ex = zeros(size(x));
    Ey = zeros(size(x));
    Ez = zeros(size(x));

    % ---------- Compute fields ----------
    for i = 1:numel(x)
        [Bx(i),By(i),Bz(i)] = solenoidField(x(i),y(i),z(i));
        [Ex(i),Ey(i),Ez(i)] = eField(x(i),y(i),z(i));
    end

    % ---------- Normalize (direction only, optional) ----------
    %Bmag = sqrt(Bx.^2 + Bz.^2);
```

```
456     %Emag = sqrt(Ex.^2 + Ez.^2);
457
458     %Bmag(Bmag==0) = 1;
459     %Emag(Emag==0) = 1;
460
461     %Bx = Bx ./ Bmag;
462     %Bz = Bz ./ Bmag;
463
464     %Ex = Ex ./ Emag;
465     %Ez = Ez ./ Emag;
466
467     % ---------- Plot ----------
468     hold on
469
470     %quiver(x,z,Bx,Bz,0.8,'b','LineWidth',1.8,'MaxHeadSize',3)
471     %quiver3(x,z,y,Bx,Bz,By,0.8,'b','LineWidth',1.8,'MaxHeadSize',3)
472
473     quiver(x,z,Ex,Ez,0.8,'g','LineWidth',2,'MaxHeadSize',3)
474     %quiver3(x,z,y,Ex,Ez,Ey,0.8,'g','LineWidth',1.8,'MaxHeadSize',3)
475
476 end
477
478 function plotStuff()
479 hold on
480
481 % Solenoid ring parameters
482 Nsol = 9;
483 A = 6;            % ring radius
484 zmin = -3;
485 zmax = 3;
486
487 for n = 1:Nsol
488     theta = 2*pi*(n-1)/Nsol;
489     xs = A*cos(theta);
490     ys = A*sin(theta);
491
492     % Vertical line along z at solenoid center
493     plot3( ...
494         xs*ones(1,2), ...
495         [zmin zmax], ...
496         ys*ones(1,2), ...
497         'y', 'LineWidth', 2 ...
498     )
499 end
500
501
```

```
Rc = 4;                          % cylinder radius
phi = linspace(0,2*pi,200);

xc = Rc*cos(phi);
yc = Rc*sin(phi);

% Bottom ring (z = zmin)
plot3(xc, zmin*ones(size(phi)), yc, ...
      'c', 'LineWidth', 2)

% Top ring (z = zmax)
plot3(xc, zmax*ones(size(phi)), yc, ...
      'c', 'LineWidth', 2)


% ---------- Plot disk ring ----------
Rd = 3.5;       % disk radius
Dd = 5;       % disk z-position

xd = Rd*cos(phi);
yd = Rd*sin(phi);

plot3(xd, Dd*ones(size(phi)), yd, ...
      'Color', [1 0.5 0], 'LineWidth', 2)    % orange



% -------------- Wire filament ------------------
x0 = 4;           % half-length
z_min = -3;
z_max = 3;

Nx = 400;
xw = linspace(-x0, x0, Nx);

% Gaussian width (controls curvature)
sigma = x0/2;

% Raw Gaussian
g = exp(-xw.^2/(2*sigma^2));

% Normalize Gaussian so:
% g(0) = 1, g(+/-x0) = 0
g0 = exp(-x0^2/(2*sigma^2));
g = (g - g0) / (1 - g0);

```

```
% Construct z(x)
zw = z_min + (z_max - z_min)*g;

% y is zero (x-z plane)
yw = zeros(size(xw));

% ---------- Plot ----------
plot3(xw, zw, yw, 'm', 'LineWidth', 2)



view(0,90)
xlim([-6 6])
ylim([-6 6])
zlim([-6 6])
hold off
end
```

Listing J.1 Snippet of the ion trajectory simulation code in MATLAB.

J.4 IBSimu For Ion Optics Simulation

Included here is the Ion Beam Simulator (IBSimu) code and results used to validate the design.

J.4.1 Code

This is the .cpp file for the ion simulation. It involves setting up the geometries, boundary conditions, magnetic field, and other parameters for simulating the given ion in this environment.

```
/*! \file nsimp_plasma3d.cpp
 *  \brief 3D plasma simulation with NSIMP Bohm-injection and automatic
 *         multi-hole aperture detection from the mesh.
 *
 *  Physics overview
 *  ================
 *  - The NSIMP solver models bulk electrons as a Boltzmann fluid:
 *        n_e(x) = n_0 * exp[ e*phi(x) / (k*Te) ]
 *    This self-consistently raises the plasma potential a few volts above
 *    the most-positive electrode, producing the correct potential gradient
```

```
 *    for extraction (ions are NOT repelled by the screen grid).
 *  - D+ ions are injected ONLY at the plasma-vacuum boundary (upstream face
 *    of the screen grid, x = plasma_bdy_x) with the Bohm velocity in +x
 *    and a small Maxwellian transverse spread at Ti.
 *  - NO explicit electron particles are added.  The NSIMP Boltzmann model
 *    handles electron space charge analytically.
 *  - Injection positions are detected automatically from the mesh: every
 *    PURE_VACUUM node in the x-slice at plasma_bdy_x is inside a grid hole
 *    and eligible for injection.  This works for any hole pattern (hex,
 *    square, irregular) without hard-coding coordinates.
 *  - The plasma meniscus emerges self-consistently from the Poisson solve.
 *
 *  Parameters to tune
 *  ==================
 *    plasma_bdy_x  - x of the upstream (plasma-facing) face of grid1.stl
 *    n_plasma      - bulk plasma density; sets beam current
 *    Te_eV         - electron temperature; sets Bohm velocity & sheath width
 */

// ============================================================
// STANDARD LIBRARY
// ============================================================
#include <cstdlib>
#include <sstream>
#include <fstream>
#include <iomanip>
#include <cmath>
#include <vector>
#include <array>
#include <random>
#include <filesystem>
#include <ctime>

// ============================================================
// IBSIMU
// ============================================================
#include "epot_bicgstabsolver.hpp"
```

```
48 #include "epot_umfpacksolver.hpp"
49 #include "epot_gssolver.hpp"
50 #include "epot_mgsolver.hpp"
51
52 #include "particledatabase.hpp"
53 #include "geometry.hpp"
54 #include "stl_solid.hpp"
55 #include "solid.hpp"
56 #include "func_solid.hpp"
57
58 #include "epot_efield.hpp"
59 #include "meshvectorfield.hpp"
60
61 #include "ibsimu.hpp"
62 #include "error.hpp"
63
64 #include "particlediagplotter.hpp"
65 #include "geomplotter.hpp"
66 #include "config.h"
67
68 #ifdef GTK3
69 #include "gtkplotter.hpp"
70 #endif
71
72 using namespace std;
73
74
75 // ============================================================
76 // SIMULATION PARAMETERS  - edit these before each run
77 // ============================================================
78
79 static const char* NOTE      = "Testing";
80
81 static const char* STL_GRID1 = "STLs/grid1.stl";
82 static const char* STL_GRID2 = "STLs/grid2.stl";
83
84 // Electrode voltages [V]
85 static const double V_GRID1    =  100.0;   // screen / extraction
      grid
86 static const double V_GRID2    = -1000.0;  // accelerator grid
87 static const double V_FILAMENT =  -50.0;   // filament bias
88 static const double V_ANODE    =  100.0;   // anode
89
90 // x-coordinate of the upstream (plasma-facing) face of grid1.stl [m
      ].
91 // The NSIMP solver treats x < plasma_bdy_x as the plasma region.
```

```
92  // Injection nodes are detected from the mesh at this x-plane.
93  static const double plasma_bdy_x = 0.026; //0.0265; //0.02654;
94
95  static const std::string SIM_NAME =
96      std::filesystem::path(__FILE__).stem().string();
97
98
99  // ============================================================
100 // FUNCTIONAL SOLIDS
101 // ============================================================
102
103 bool square_plate(double x, double y, double z)
104 {
105     const double xc             = 0.086;
106     const double yc             = 0.0;
107     const double zc             = 0.0;
108     const double half_side      = 0.0254 * 0.5;
109     const double half_thickness = 10.0e-4 * 0.5;
110     return (   fabs(x - xc) <= half_thickness
111             && fabs(y - yc) <= half_side
112             && fabs(z - zc) <= half_side );
113 }
114
115 bool ring_electrode(double x, double y, double z)
116 {
117     const double xc             = 0.0375;
118     const double half_thickness = 1e-3;
119     const double r_inner        = 0.005;
120     const double r_outer        = 0.026;
121     double r = std::sqrt(y*y + z*z);
122     return (   fabs(x - xc) <= half_thickness
123             && r >= r_inner
124             && r <= r_outer );
125 }
126
127
128 // ============================================================
129 // MAGNETIC FIELD - 9-magnet dipole ring
130 // ============================================================
131
132 static double compute_dipole_moment()
133 {
134     const double B_face = 0.02;
135     const double L      = 0.5 * 0.0254;
136     const double d      = L / 2.0;
137     return B_face * (d * d * d) / (2.0e-7);
```

```
138 }
139
140 static void dipole_field(const Vec3D& p_field,
141                          const Vec3D& p_mag,
142                          double       m_moment,
143                          double&      Bx,
144                          double&      By,
145                          double&      Bz)
146 {
147     const double mu0_over_4pi = 1e-7;
148     double rx = p_field[0] - p_mag[0];
149     double ry = p_field[1] - p_mag[1];
150     double rz = p_field[2] - p_mag[2];
151     double r2 = rx*rx + ry*ry + rz*rz;
152     if (r2 < (5e-3 * 5e-3)) { Bx = By = Bz = 0.0; return; }
153     double r       = std::sqrt(r2);
154     double r5      = r2 * r2 * r;
155     double m_dot_r = m_moment * rx;
156     double factor  = mu0_over_4pi / r5;
157     Bx = factor * (3.0 * m_dot_r * rx - m_moment * r2);
158     By = factor * (3.0 * m_dot_r * ry);
159     Bz = factor * (3.0 * m_dot_r * rz);
160 }
161
162 MeshVectorField build_bfield_magnets(const Geometry& geom,
163                                      double magnet_x,
164                                      double ring_radius = 0.75 *
                                            0.0254,
165                                      int    n_magnets   = 9)
166 {
167     double m_moment = compute_dipole_moment();
168     ibsimu.message(1) << "Magnet dipole moment : " << m_moment << " A
            *m^2\n";
169     ibsimu.message(1) << "Magnet ring x-pos    : " << magnet_x << " m
            \n";
170
171     std::vector<Vec3D> mag_pos;
172     mag_pos.reserve(n_magnets);
173     for (int k = 0; k < n_magnets; k++)
174     {
175         double phi = 2.0 * M_PI * k / n_magnets;
176         mag_pos.push_back(Vec3D(magnet_x,
177                                 ring_radius * std::sin(phi),
178                                 ring_radius * std::cos(phi)));
179     }
180
```

```
    bool fout[3] = { true, true, true };
    MeshVectorField bfield(geom.geom_mode(), fout,
                           geom.size(), geom.origo(), geom.h());
    Int3D  sz  = geom.size();
    Vec3D  org = geom.origo();
    double h   = geom.h();
    for (int ix = 0; ix < sz[0]; ix++)
    for (int iy = 0; iy < sz[1]; iy++)
    for (int iz = 0; iz < sz[2]; iz++)
    {
        Vec3D  p_field(org[0]+ix*h, org[1]+iy*h, org[2]+iz*h);
        double Bx=0, By=0, Bz=0, dBx, dBy, dBz;
        for (const Vec3D& pm : mag_pos)
        {
            dipole_field(p_field, pm, m_moment, dBx, dBy, dBz);
            Bx+=dBx; By+=dBy; Bz+=dBz;
        }
        bfield.set(ix, iy, iz, Vec3D(Bx, By, Bz));
    }
    ibsimu.message(1) << "Magnetic field grid filled.\n";
    return bfield;
}


// ============================================================
// HELPERS
// ============================================================

static std::string today_string()
{
    std::time_t now = std::time(nullptr);
    std::tm* t = std::localtime(&now);
    char buf[16];
    std::strftime(buf, sizeof(buf), "%Y-%m-%d", t);
    return buf;
}

static void archive_pngs()
{
    namespace fs = std::filesystem;
    fs::create_directories("Pictures");
    for (auto& entry : fs::directory_iterator("."))
    {
        if (!entry.is_regular_file()) continue;
        if (entry.path().extension() != ".png") continue;
```

```
226         fs::path dest = fs::path("Pictures") / entry.path().filename
               ();
227         if (fs::exists(dest))
228         {
229             std::string stem = entry.path().stem().string();
230             int idx = 0;
231             do {
232                 dest = fs::path("Pictures") /
233                        (stem + "_" + std::to_string(idx) + ".png");
234                 idx++;
235             } while (fs::exists(dest));
236         }
237         fs::rename(entry.path(), dest);
238     }
239 }
240
241 static void append_csv(const std::string& sim_name,
242                        const std::string& stl_grid1,
243                        const std::string& stl_grid2,
244                        double v_grid1,    double v_grid2,
245                        double v_filament, double v_anode,
246                        double transmission_pct,
247                        double mean_energy_eV,
248                        double stddev_energy_eV,
249                        double beam_2sigma_m,
250                        const std::string& note)
251 {
252     std::string csv_path = "data.csv";
253     bool write_header = !std::filesystem::exists(csv_path);
254     std::ofstream csv(csv_path, std::ios::app);
255     if (write_header)
256         csv << "FileName,Date,STL_Grid1,STL_Grid2,"
257                "V_Grid1(V),V_Grid2(V),V_Filament(V),V_Anode(V),"
258                "Transmission(%),MeanEnergy(eV),StdDevEnergy(eV),"
259                "BeamWidth_2sigma(m),Note\n";
260     csv << sim_name    << "," << today_string() << ","
261         << stl_grid1   << "," << stl_grid2      << ","
262         << v_grid1     << "," << v_grid2        << ","
263         << v_filament  << "," << v_anode        << ","
264         << std::fixed << std::setprecision(2)
265         << transmission_pct << "," << mean_energy_eV << ","
266         << stddev_energy_eV << ","
267         << std::setprecision(6)
268         << beam_2sigma_m << "," << note << "\n";
269 }
270
```

```
271
272 // ============================================================
273 // SIMULATION
274 // ============================================================
275
276 void test(int argc, char **argv)
277 {
278     // ------------------------------------------------------------
279     // Domain
280     // ------------------------------------------------------------
281     const double xmin = plasma_bdy_x;
282     const double xmax = 0.1;
283     const double ymin = -0.013;
284     const double ymax =  0.013;
285     const double zmin = -0.013;
286     const double zmax =  0.013;
287     const double h    =  3e-4;
288
289     int nx = (int)std::round((xmax - xmin) / h) + 1;
290     int ny = (int)std::round((ymax - ymin) / h) + 1;
291     int nz = (int)std::round((zmax - zmin) / h) + 1;
292
293     Geometry geom(MODE_3D, Int3D(nx, ny, nz), Vec3D(xmin, ymin, zmin)
        , h);
294
295
296     // ------------------------------------------------------------
297     // Solids
298     // ------------------------------------------------------------
299     STLSolid *grid1 = new STLSolid(STL_GRID1);
300     Vec3D grid1_pos(0.0, 0.0, 0.0);
301     grid1->translate(grid1_pos);
302     geom.set_solid(7, grid1);
303
304     STLSolid *grid2 = new STLSolid(STL_GRID2);
305     grid2->translate(grid1_pos);
306     geom.set_solid(8, grid2);
307
308     STLSolid *filament = new STLSolid("STLs/filament.stl");
309     filament->translate(Vec3D(0.0, 0.0, 0.0));
310     geom.set_solid(9, filament);
311
312     STLSolid *shell = new STLSolid("STLs/inverted_cone.stl");
313     shell->translate(Vec3D(0.0, 0.0, 0.0));
314     geom.set_solid(10, shell);
315
```

```
316     geom.set_solid(11, new FuncSolid(square_plate));
317
318     STLSolid *anode = new STLSolid("STLs/anode.stl");
319     anode->translate(Vec3D(0.0, 0.0, 0.0));
320     geom.set_solid(12, anode);
321
322     const double magnet_x = grid1_pos[0] - 5e-3;
323
324
325     // ------------------------------------------------------------
326     // Boundary conditions
327     // ------------------------------------------------------------
328     geom.set_boundary(1,  Bound(BOUND_NEUMANN,   0.0));
329     geom.set_boundary(2,  Bound(BOUND_NEUMANN,   0.0));
330     geom.set_boundary(3,  Bound(BOUND_NEUMANN,   0.0));
331     geom.set_boundary(4,  Bound(BOUND_NEUMANN,   0.0));
332     geom.set_boundary(5,  Bound(BOUND_NEUMANN,   0.0));
333     geom.set_boundary(6,  Bound(BOUND_NEUMANN,   0.0));
334
335     geom.set_boundary(7,  Bound(BOUND_DIRICHLET, V_GRID1));
336     geom.set_boundary(8,  Bound(BOUND_DIRICHLET, V_GRID2));
337     geom.set_boundary(9,  Bound(BOUND_DIRICHLET, V_FILAMENT));
338     geom.set_boundary(10, Bound(BOUND_DIRICHLET,   0.0));
339     geom.set_boundary(11, Bound(BOUND_DIRICHLET, V_GRID2 - 9.0));
340     geom.set_boundary(12, Bound(BOUND_DIRICHLET, V_ANODE));
341
342     geom.build_mesh();
343
344
345     // ------------------------------------------------------------
346     // Geometry export
347     // ------------------------------------------------------------
348     {
349         ofstream gf("geometry.dat");
350         Int3D sz = geom.size(); Vec3D o = geom.origo(); double gh =
                geom.h();
351         for (int ix = 0; ix < sz[0]; ix++)
352         for (int iy = 0; iy < sz[1]; iy++)
353         for (int iz = 0; iz < sz[2]; iz++)
354         {
355             uint32_t raw      = geom.mesh(ix, iy, iz);
356             uint32_t nodetype = raw & SMESH_NODE_ID_MASK;
357             uint32_t id       = raw & SMESH_BOUNDARY_NUMBER_MASK;
358             if (nodetype == SMESH_NODE_ID_PURE_VACUUM) continue;
359             gf << (o[0]+ix*gh) << " " << (o[1]+iy*gh) << " "
                    << (o[2]+iz*gh) << " " << id << "\n";
360
```

```
361         }
362         cout << "geometry.dat written\n";
363     }
364
365
366     // ------------------------------------------------------------
367     // Physical constants
368     // ------------------------------------------------------------
369     const double e_charge = 1.602176634e-19;
370     const double m_D      = 2.0 * 1.66053906660e-27;  // D+ mass [kg]
371     const double J_to_eV  = 1.0 / e_charge;
372
373
374     // ------------------------------------------------------------
375     // Plasma model parameters
376     // ------------------------------------------------------------
377     // Te_eV: electron temperature [eV].
378     //   Sets the Bohm velocity, sheath width, and meniscus curvature
        .
379     //   2-5 eV is typical for a filament discharge in D2.
380     //   Higher Te -> faster Bohm velocity, deeper meniscus, stronger
         focusing.
381     const double Te_eV = 3.0;
382
383     // Ti_eV: D+ ion temperature [eV].
384     //   Only affects the transverse thermal spread at injection.
385     //   0.03-0.1 eV is typical.
386     const double Ti_eV = 0.07;
387
388     // n_plasma: bulk plasma density at the boundary [m^-3].
389     //   Directly sets the Bohm current and hence total beam current.
390     //   Filament discharge in a small D2 chamber: ~10^15 to 10^17 m
        ^-3.
391     //   Tune this so the simulated beam current matches your
           hardware.
392     const double n_plasma   = 1.0e16; //FOR DIAGNOSIS. ORIGINALLY 16!
393     const double rho_plasma = e_charge * n_plasma;   // [C/m^3]
394
395     // Bohm velocity for D+: v_B = sqrt(k*Te / m_D)
396     const double v_bohm = std::sqrt(Te_eV * e_charge / m_D);
397
398     // Bohm current density: j_B = e * n * v_B  [A/m^2]
399     const double j_bohm = e_charge * n_plasma * v_bohm;
400
401
402     // ------------------------------------------------------------
```

```
403     // Auto-detect injection nodes from mesh
404     // ------------------------------------------------------------
405     // Scan the x-slice of the mesh at x = plasma_bdy_x.
406     // Every PURE_VACUUM node in that slice sits inside a grid hole
        and is a
407     // valid injection point.  This works automatically for any
        number of
408     // holes in any arrangement - hex, square, or irregular.
409     //
410     // Note: injection positions are discretised to the mesh spacing
        h.
411     // At h = 0.3 mm, each ~1 mm-radius hole has ~35 nodes, which
        gives
412     // adequate sampling for space-charge deposition.
413     {
414         int ix_inj = static_cast<int>(std::round((plasma_bdy_x - xmin
            ) / h));
415         if (ix_inj < 0 || ix_inj >= geom.size()[0])
416             throw std::runtime_error("plasma_bdy_x is outside the
                domain.");
417         ibsimu.message(1) << "Scanning mesh at ix = " << ix_inj
418                           << " (x = " << (xmin + ix_inj*h)*1e3 << "
                              mm) "
419                           << "for injection nodes...\n";
420     }
421
422     // Collect vacuum nodes at the injection plane (done once, before
         the loop).
423     std::vector<std::pair<double,double>> hole_nodes;   // (y, z)
        pairs
424     {
425         Int3D sz  = geom.size();
426         Vec3D org = geom.origo();
427         int ix_inj = static_cast<int>(std::round((plasma_bdy_x - xmin
            ) / h));
428
429         for (int iy = 0; iy < sz[1]; iy++)
430         for (int iz = 0; iz < sz[2]; iz++)
431         {
432             uint32_t raw      = geom.mesh(ix_inj, iy, iz);
433             uint32_t nodetype = raw & SMESH_NODE_ID_MASK;
434             if (nodetype == SMESH_NODE_ID_PURE_VACUUM)
435                 hole_nodes.push_back({ org[1] + iy*h,
436                                        org[2] + iz*h });
437         }
438
```

```
439         if (hole_nodes.empty())
440             throw std::runtime_error(
441                 "No vacuum nodes found at plasma_bdy_x. "
442                 "Check that plasma_bdy_x matches the upstream face of
                        grid1.stl.");
443
444         // Open area derived directly from the mesh.
445         double open_area = hole_nodes.size() * h * h;   // [m^2]
446         double beam_current_est = j_bohm * open_area;
447
448         ibsimu.message(1) << "=== Hole detection results ===\n";
449         ibsimu.message(1) << "  Vacuum nodes found : " << hole_nodes.
                size() << "\n";
450         ibsimu.message(1) << "  Open area          : " << open_area*1
                e6 << " mm^2\n";
451         ibsimu.message(1) << "  Te                 : " << Te_eV    <<
                " eV\n";
452         ibsimu.message(1) << "  n_plasma           : " << n_plasma
                << " m^-3\n";
453         ibsimu.message(1) << "  v_Bohm (D+)        : " << v_bohm   <<
                " m/s\n";
454         ibsimu.message(1) << "  j_Bohm             : " << j_bohm   <<
                " A/m^2\n";
455         ibsimu.message(1) << "  Est. beam current  : " <<
                beam_current_est*1e3 << " mA\n\n";
456
457         // Write hole node positions to a diagnostic file so you can
                verify
458         // the detection looks correct before committing to a long
                run.
459         {
460             std::ofstream hf("hole_nodes.dat");
461             hf << "# y[m]  z[m]  (vacuum nodes at x = plasma_bdy_x)\n
                    ";
462             for (auto& [y, z] : hole_nodes)
463                 hf << y << " " << z << "\n";
464             cout << "hole_nodes.dat written (" << hole_nodes.size()
465                  << " nodes)\n";
466         }
467     }
468
469     // Open area and beam current - computed from detected nodes, not
            assumed.
470     const double open_area    = hole_nodes.size() * h * h;
471     const double beam_current = j_bohm * open_area;
472
```

```
473
474     // ------------------------------------------------------------
475     // NSIMP solver
476     // ------------------------------------------------------------
477     EpotBiCGSTABSolver solver(geom);
478
479     // InitialPlasma: on the very first solve, x < plasma_bdy_x is
480     // initialised with a flat (plasma-like) potential to give the
        solver
481     // a sensible starting guess before the Boltzmann space-charge
        kicks in.
482     InitialPlasma initp(AXIS_X, plasma_bdy_x);
483     solver.set_nsimp_initial_plasma(&initp);
484
485     // Boltzmann plasma model.
486     // rhoe   = electron charge density at the plasma potential [C/m
        ^3].
487     //          Quasi-neutrality => rhoe = e * n_plasma.
488     // Te_eV  = electron temperature [eV].
489     //          IBSimu applies: rho_e(x) = rhoe * exp( e*phi(x) / (k*
        Te) )
490     // rhoi   = ion charge density vector (one entry per species).
491     //          For single-species D+ plasma: rhoi[0] = rhoe (quasi-
        neutral).
492     // Ei     = ion energy at the sheath edge [eV].
493     //          Bohm criterion: Ei[0] = Te (half-energy convention in
         IBSimu).
494     {
495         std::vector<double> rhoi_vec = { rho_plasma };
496         std::vector<double> Ei_vec   = { Te_eV };
497         solver.set_nsimp_plasma(rho_plasma, Te_eV, rhoi_vec, Ei_vec);
498     }
499
500     EpotField       epot(geom);
501     MeshScalarField scharge(geom);
502     EpotEfield      efield(epot);
503
504     field_extrpl_e efldextrpl[6] = {
505         FIELD_EXTRAPOLATE, FIELD_EXTRAPOLATE,
506         FIELD_EXTRAPOLATE, FIELD_EXTRAPOLATE,
507         FIELD_EXTRAPOLATE, FIELD_EXTRAPOLATE
508     };
509     efield.set_extrapolation(efldextrpl);
510
511     MeshVectorField bfield = build_bfield_magnets(geom, magnet_x);
512
```

```
513
514     // ------------------------------------------------------------
515     // Particle database
516     // ------------------------------------------------------------
517     ParticleDataBase3D pdb(geom);
518
519     bool pmirror[6] = { false, false, false, false, false, false };
520     pdb.set_mirror(pmirror);
521     pdb.set_polyint(true);
522
523
524     // ------------------------------------------------------------
525     // Injection parameters
526     // ------------------------------------------------------------
527     // Distribute num_particles uniformly across all detected hole
        nodes.
528     // IQ_per_particle is set so the total current weight exactly
        equals
529     // the Bohm current through the detected open area.
530     const int    num_particles   = 50000; // For testing
531     const double IQ_per_particle = beam_current / num_particles;
532
533     ibsimu.message(1) << "  num_particles    : " << num_particles <<
        "\n";
534     ibsimu.message(1) << "  IQ_per_particle  : " << IQ_per_particle*1
        e9 << " nA\n\n";
535
536     // Target plate geometry (must match square_plate())
537     const double target_xc        = 0.086;
538     const double target_half_side = 0.0254 * 0.5;
539     const double target_half_th   = 15.0e-4 * 0.5;
540
541     // 10 iterations for meniscus convergence.
542     const size_t N_ITER = 8; // Very high for testing!
543
544
545     // ------------------------------------------------------------
546     // Self-consistent iteration loop
547     // ------------------------------------------------------------
548     for (size_t i = 0; i < N_ITER; i++)
549     {
550         ibsimu.message(1) << "\n=== Iteration " << i+1 << " / "
551                           << N_ITER << " ===\n";
552
553         solver.solve(epot, scharge);
554         efield.recalculate();
```

```
555         pdb.clear();
556
557
558         // -- Bohm injection from detected hole nodes
                -------------------
559         //
560         // For each macroparticle we pick a random node from
                hole_nodes
561         // and add a small sub-cell jitter (uniform within +-h/2) so
562         // particles don't all stack on mesh lattice positions.
563         // This smooths the space-charge deposition without changing
                the
564         // total current or injection area.
565         //
566         // Only D+ ions are added.  Electrons are handled by NSIMP.
567         {
568             std::mt19937 rng(42 + i * 1000);
569             std::uniform_int_distribution<int>    pick(0, (int)
                    hole_nodes.size()-1);
570             std::uniform_real_distribution<double> jitter(-0.5*h,
                    0.5*h);
571
572             const double kTi_J  = Ti_eV * e_charge;
573             const double v_th_i = std::sqrt(kTi_J / m_D);
574             std::normal_distribution<double> g_trans(0.0, v_th_i);
575
576             for (int p = 0; p < num_particles; ++p)
577             {
578                 // Choose a random hole node and apply sub-cell
                        jitter.
579                 auto [y_node, z_node] = hole_nodes[pick(rng)];
580                 double y_inj = y_node + jitter(rng);
581                 double z_inj = z_node + jitter(rng);
582
583                 // vx = Bohm velocity (+x, into extraction gap)
584                 // vy, vz = small thermal spread at ion temperature
585                 pdb.add_particle(IQ_per_particle, 1.0 /*q/e*/, 2.0 /*
                        amu*/,
586                                  ParticleP3D(0.0,
587                                              plasma_bdy_x, v_bohm,
588                                              y_inj,        g_trans(
                                                    rng),
589                                              z_inj,        g_trans(
                                                    rng)));
590             }
591         }
```

```
592
593         pdb.iterate_trajectories(scharge, efield, bfield);
594
595
596         // -- Trajectory export
                ------------------------------------------
597         {
598             ofstream tf("trajectories.dat");
599             for (size_t j = 0; j < pdb.size(); j++)
600             {
601                 Particle3D &part = pdb.particle(j);
602                 if (part.traj_size() < 2) continue;
603                 for (size_t k = 0; k < part.traj_size(); k++)
604                 {
605                     ParticleP3D &pt = part.traj(k);
606                     tf << pt[0] << " " << pt[1] << " " << pt[2] << "
                            "
607                        << pt[3] << " " << pt[4] << " " << pt[5] << "
                            "
608                        << pt[6] << "\n";
609                 }
610                 tf << "\n";
611             }
612         }
613
614
615         // -- Plot
                ------------------------------------------------------------
616         // Equipotential lines cover both positive (plasma bulk,
                meniscus)
617         // and negative (accel gap) regions so the full potential
                structure
618         // is visible in a single PNG.
619         archive_pngs();
620         std::string png_name = SIM_NAME + "_iter_" + std::to_string(i
                ) + ".png";
621
622         std::vector<double> eqlines = {
623             // Plasma bulk and meniscus region
624             110.0, 108.0, 106.0, 104.0, 103.0, 102.5,
625             102.0, 101.5, 101.0, 100.8, 100.6, 100.4,
626             100.2, 100.1, 100.0,
627
628             // Just below screen potential
629              99.9,  99.8,  99.6,  99.4,  99.2,  99.0,
630              98.0,  96.0,  94.0,  92.0,  90.0,
```

```
631
632             80.0,  70.0,  60.0,  50.0,  40.0,
633             30.0,  20.0,  10.0,   5.0,   2.0,   1.0,
634
635              0.0,
636
637             -1.0,  -2.0,  -5.0, -10.0, -25.0,
638            -50.0, -100.0, -250.0, -500.0, -1000.0
639         };
640
641         const int y_mid_index = static_cast<int>(std::round((0.0 -
               ymin) / h));
642
643         GeomPlotter gp_xy(geom);
644         gp_xy.set_size(1024, 768);
645         gp_xy.set_epot(&epot);
646         gp_xy.set_view(VIEW_XY, y_mid_index);
647         gp_xy.set_eqlines_manual(eqlines);
648         gp_xy.set_bfield(&bfield);
649         gp_xy.set_particle_database(&pdb);
650         gp_xy.plot_png(png_name.c_str());
651
652
653         // -- Beam diagnostics
               ---------------------------------------------
654         size_t hit_target = 0;
655         size_t total_ions = 0;
656         double E_sum = 0, E2_sum = 0, r_sum = 0, r2_sum = 0;
657
658         for (size_t j = 0; j < pdb.size(); j++)
659         {
660             Particle3D &part = pdb.particle(j);
661             if (part.q() <= 0.0) continue;
662             total_ions++;
663             if (part.traj_size() < 2) continue;
664
665             ParticleP3D &pt = part.traj(part.traj_size() - 1);
666             double x=pt[1], vx=pt[2], y=pt[3], vy=pt[4], z=pt[5], vz=
                   pt[6];
667
668             bool hit = fabs(x - target_xc) <= target_half_th
669                     && fabs(y)              <= target_half_side
670                     && fabs(z)              <= target_half_side;
671             if (!hit) continue;
672
673             hit_target++;
```

```
674             double v2 = vx*vx + vy*vy + vz*vz;
675             double E  = 0.5 * m_D * v2 * J_to_eV;
676             E_sum  += E;  E2_sum += E*E;
677             double r = std::sqrt(y*y + z*z);
678             r_sum  += r;  r2_sum += r*r;
679         }
680
681         double transmission_pct = total_ions > 0
682             ? 100.0 * (double)hit_target / total_ions : 0.0;
683         double mean_E   = hit_target > 0 ? E_sum / hit_target : 0.0;
684         double stddev_E = hit_target > 1
685             ? std::sqrt(std::max(0.0, E2_sum/hit_target - mean_E*
                    mean_E)) : 0.0;
686         double r_mean    = hit_target > 0 ? r_sum  / hit_target :
                0.0;
687         double r_var     = hit_target > 0 ? r2_sum / hit_target -
                r_mean*r_mean : 0.0;
688         double beam_2sig = r_var > 0.0 ? 2.0 * std::sqrt(r_var) :
                0.0;
689
690         std::cout << "\n=== Beam diagnostics (iter " << i << ") ===\n
                "
691                   << "Injected ions : " << total_ions << "\n"
692                   << "Hit target    : " << hit_target  << "\n"
693                   << "Transmission  : " << std::fixed << std::
                         setprecision(1)
694                   << transmission_pct << " %\n"
695                   << "Mean energy   : " << mean_E   << " eV\n"
696                   << "StdDev energy : " << stddev_E << " eV\n"
697                   << "Beam width    : " << std::setprecision(4)
698                   << beam_2sig*1e3 << " mm (2-sigma)\n\n";
699
700         if (i == N_ITER - 1)
701         {
702             append_csv(SIM_NAME,
703                        STL_GRID1, STL_GRID2,
704                        V_GRID1, V_GRID2, V_FILAMENT, V_ANODE,
705                        transmission_pct, mean_E, stddev_E, beam_2sig,
706                        NOTE);
707             std::cout << "Results appended to " << SIM_NAME << ".csv\
                    n";
708         }
709     }
710 }
711
712
```

```
// ============================================================
// ENTRY POINT
// ============================================================

int main(int argc, char **argv)
{
    bool success = true;
    try
    {
        ibsimu.set_message_threshold(MSG_VERBOSE, 1);
        test(argc, argv);
        std::cout << "3D plasma simulation completed successfully.\n";
    }
    catch (Error &e)          { e.print_error_message(std::cerr); success = false; }
    catch (std::exception &e) { std::cerr << "Exception: " << e.what() << "\n"; success = false; }

    const char* payload = success
        ? "{\\\"content\\\": \\\" IBSimu simulation finished successfully!\\\"}"
        : "{\\\"content\\\": \\\" IBSimu simulation FAILED!\\\"}";

    std::string cmd =
        std::string("curl -s -X POST -H \"Content-Type: application/json\" -d \"")
        + payload + "\" "
        "https://discord.com/api/webhooks/1511476594147987618/"
        "Z35IoBrTWzyO_j4IxUUB9iCDxKJhxA_S9vIvzdU7_kf9uepciABHuKRYcEukKybc6Hhe";
    system(cmd.c_str());

    return success ? 0 : 1;
}
```

Listing J.2 Snippet of the IBSimu Ion Trajectory Code (First 50 lines).

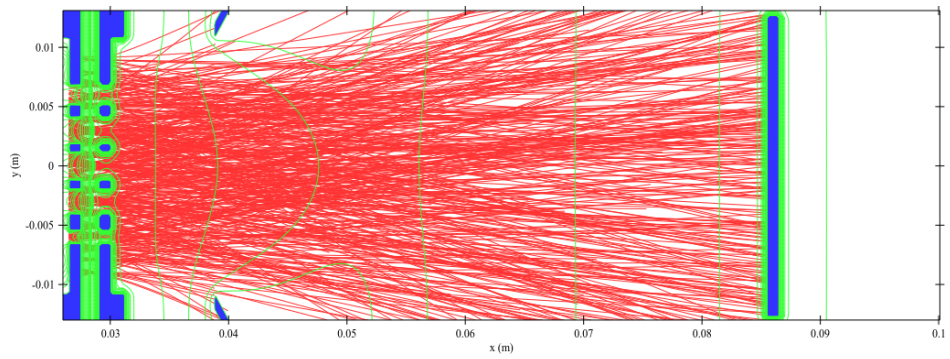

Figure J.5 Results from the IBSimu simulation. Note that the potential lines in green are not evenly spaced in voltage, but demonstrate the general potential shape.

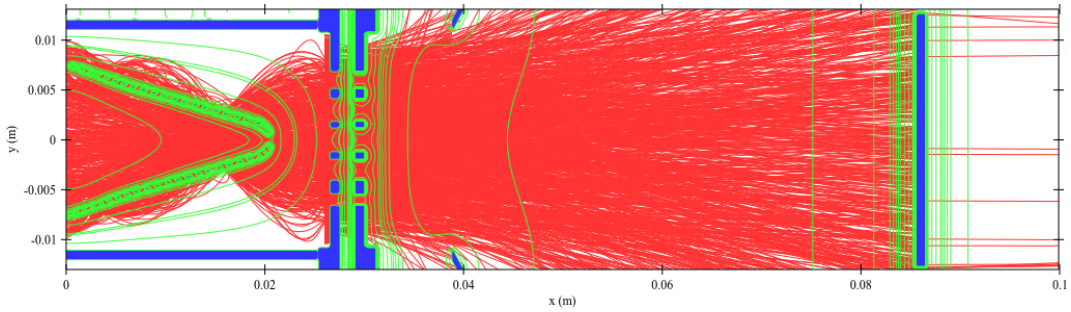

Figure J.6 IBSimu results showing the filament and anode on the left. The filament is held at -50 V and the anode is at 100 V. IBSimu is not designed to simulate this plasma in the chamber, but it is meant to simulate the extracted ions on the right. So having the filament and anode there is unnecessary but demonstrates the setup.

J.4.2 Results

The grid on the left is set at 100 V, while the grid on the right is at -1000 V. When a particle in the plasma enters the extraction region between the two grids, it's accelerated at about 1100 eV towards the target. This is demonstrated in figures J.5 and J.6.

This assumes that a plasma will correctly form around the filament and properly extract between the grids. However, what this demonstrates is that the grids are effective at generating an ion beam from a plasma. Though there is some spread in the beam, a large fraction of the ions (about 40% according to the simulation) hit the target with about 1.1 keV of kinetic energy. This

simulation isn't powerful enough to determine the beam current accurately, and that will require assembly and experimental validation. However, the computational approach helped to verify the general optics of the system.

Appendix K

Optical Spectroscopy

Optical spectroscopy was used in an attempt to identify the species of gas (especially quantity of H+ versus H2+ and other species) in a 0.4 mTorr RF hydrogen plasma. Included here is python code originally written by Claude (AI) to analyze the resulting spectrum, comparing it to NIST and other databases. The result was a CSV file and a PNG file displaying an image of the results as seen in figure K.1.

Listing K.1 Snippet of the plasma optical spectrum analyzing code.

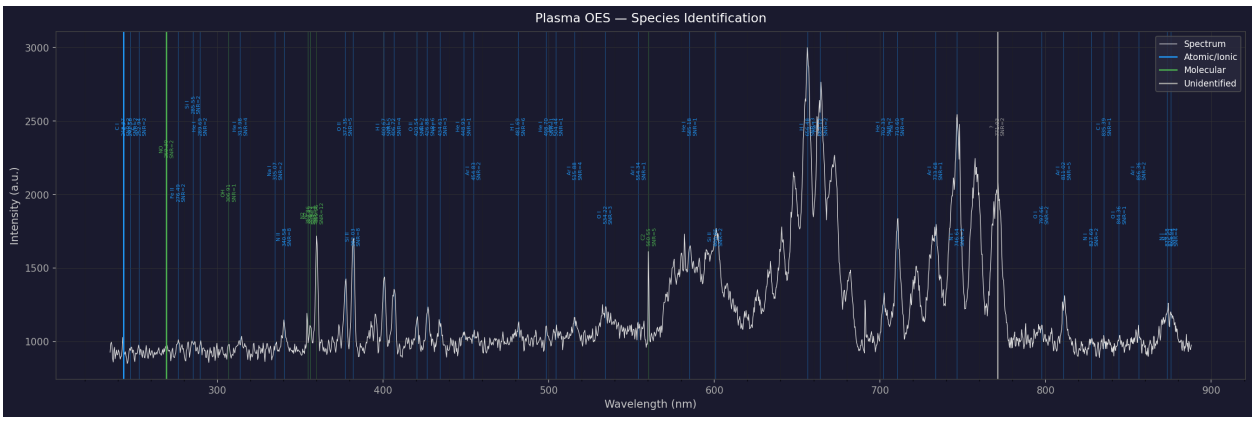

Figure K.1 Sample test using the optical spectrum from the Hydrogen plasma. Note the high-density lines and unlikely identifications. This code seems to be flawed, leading to further development.

```
"""
plasma_oes_analyzer.py
======================
Plasma Optical Emission Spectroscopy (OES) analyzer.
Identifies atomic/ionic lines (via NIST ASD) and molecular bands
(via a local CSV line list from HITRAN/LIFBASE) in a measured spectrum.

Usage
-----
  python plasma_oes_analyzer.py spectrum.csv

Spectrum CSV format (two columns, header optional):
  wavelength_nm, intensity
  235.10, 0.043
  235.20, 0.087
  ...

Molecular line list CSV format (see README block at end of file):
  molecule, wavelength_nm, intensity_rel, assignment
  OH,       306.36,        1.0,           "A-X (0,0) bandhead"
  ...

Install requirements:
  pip install specutils astroquery scipy numpy matplotlib pandas astropy
"""

import sys
import warnings
import argparse
from pathlib import Path

import numpy as np
import pandas as pd
import matplotlib
matplotlib.use("Agg")          # headless-safe; change to "TkAgg" for interactive
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from scipy.signal import find_peaks, peak_prominences
from scipy.ndimage import uniform_filter1d

warnings.filterwarnings("ignore")   # suppress astroquery HTTP noise


```

```
#
# ------------------------------------------------------------------------

# CONFIG  -  edit these for your experiment
#
# ------------------------------------------------------------------------

CFG = dict(
    # -- Spectrum loading
    # ------------------------------------------------------
    # Your spectrometer CSV: Pixel | Wavelength (nm) | Sum Spectrum | Averaged Spectrum
    spectrum_wavelength_col = 1,        # col 2 "Wavelength (nm)"
    spectrum_intensity_col  = 3,        # col 4 "Averaged Spectrum"
    spectrum_delimiter      = ",",
    # None = auto-detect data start (skips metadata like "Scan Date:,...")
    # Set to an integer to force a fixed number of rows to skip.
    spectrum_skip_rows      = None,
    wavelength_unit         = "nm",     # "nm" or "AA" (Angstrom)

    # -- Peak detection
    # ------------------------------------------------------
    # Smooth the spectrum before peak-finding (reduces noise false positives).
    smooth_window_pts       = 5,        # set to 1 to disable smoothing

    # Minimum prominence as a fraction of the global dynamic range.
    # This is a coarse first-pass filter. The SNR filter below is more meaningful.
    peak_prominence_frac    = 0.01,     # 1 % of (max - floor); lower = more candidates

    # Minimum width in data points (filters out single-pixel spikes).
    peak_min_width_pts      = 2,

    # -- Local SNR filter (the main noise rejection knob)
    # --------------------
    # For each candidate peak, the code estimates the LOCAL noise floor in a
    # window around the peak, then computes:
    #
    #   SNR = (peak_height - local_baseline) / local_noise_std
    #
    # Peaks below peak_min_snr are rejected as noise.
```

```
    #
    # Guidance:
    #   3-5   - keep only clear, unambiguous emission lines
    #   2-3   - include weaker real lines (some noise may slip
       through)
    #   1-2   - very permissive; use if you're missing known lines
    #   None  - disable SNR filter entirely
    peak_min_snr            = 1,

    # Half-width (in data points) of the window used to estimate
       local noise.
    # Should be wide enough to span several resolution elements but
       not so wide
    # that it spans multiple strong lines. ~50-150 pts is usually
       right.
    peak_snr_window_pts     = 50,

    # -- NIST atomic line matching
       ------------------------------------------
    # Elements to query. Use None to query ALL (slow, ~30 s first run
       ).
    # Common plasma contaminants always worth including: H, N, O, C,
       Ar.
    # Add your analyte elements here.
    elements                = ["H", "He", "N", "O", "C", "Ar",
                               "Si", "Fe", "Cu", "Na", "Ca", "Mg", "
                                  Ne", "Kr",'Hg'],
    # Include ionic species (I = neutral, II = singly ionized, etc.)
    ion_stages              = ["I", "II", "III"],

    # -- Wavelength matching tolerance
       ---------------------------------------
    # This is the most important knob for unidentified peaks.
    # A peak at lambda_measured matches a database line at lambda_db
       if
    #   |lambda_measured - lambda_db| <= match_tolerance_nm
    #
    # Typical values by detector quality:
    #   0.1-0.2 nm  - high-resolution, well-calibrated bench
       spectrometer
    #   0.3-0.5 nm  - mid-grade USB/compact spectrometer (e.g. Ocean
       Optics)
    #   0.5-1.0 nm  - lower-precision or poorly-calibrated detector
       <- try this
    #   1.0-2.0 nm  - very low resolution (e.g. broadband detector)
    #
```

```
    # Raising this catches more real lines but also increases false positives.
    # If you have LOTS of unidentified peaks, try 0.8 or 1.0 first.
    match_tolerance_nm      = 5,

    # Minimum NIST relative intensity to include a candidate line.
    # NIST intensities are unitless rankings; 0 = include everything.
    nist_min_intensity      = 0,

    # -- Molecular line list ------------------------------------------
    # Path to your downloaded molecular line list CSV.
    # Set to None to skip molecular matching (or if file doesn't exist).
    molecular_csv           = "molecular_lines.csv",

    # -- Output -------------------------------------------------------------
    output_plot             = "oes_analysis.png",
    output_table            = "oes_results.csv",
    dpi                     = 150,
)
# ------------------------------------------------------------------------------------------


# -- 1. Spectrum loading ------------------------------------------------

def load_spectrum(path: str) -> pd.DataFrame:
    """Load spectrum from CSV/TXT. Returns DataFrame with 'wl' and 'intensity'."""
    path = Path(path)
    if not path.exists():
        sys.exit(f"[ERROR] Spectrum file not found: {path}")

    sep = CFG["spectrum_delimiter"]

    # Auto-detect how many rows to skip: scan until BOTH the wavelength
    # and intensity columns are numeric in the same row.
    skip = CFG["spectrum_skip_rows"]
    if skip is None:
        wl_col  = CFG["spectrum_wavelength_col"]
        int_col = CFG["spectrum_intensity_col"]
```

```
        need    = max(wl_col, int_col) + 1
        with open(path, "r", errors="replace") as fh:
            for i, line in enumerate(fh):
                parts = line.split(sep) if sep else line.split()
                if len(parts) >= need:
                    try:
                        float(parts[wl_col].strip())
                        float(parts[int_col].strip())
                        skip = i   # first row where both cols are numeric
                        break
                    except ValueError:
                        pass
        if skip is None:
            sys.exit("[ERROR] Could not find numeric data in spectrum file. "
                     "Set spectrum_skip_rows manually in CFG.")
        print(f"[INFO] Auto-detected data start at row {skip} "
              f"(skipped {skip} header/metadata rows).")

    try:
        raw = pd.read_csv(
            path,
            sep        = sep,
            header     = None,
            skiprows   = skip,
            engine     = "python",
        )
    except Exception as e:
        sys.exit(f"[ERROR] Could not read spectrum file: {e}")

    wl   = pd.to_numeric(raw.iloc[:, CFG["spectrum_wavelength_col"]], errors="coerce")
    ints = pd.to_numeric(raw.iloc[:, CFG["spectrum_intensity_col"]], errors="coerce")
    mask = wl.notna() & ints.notna()
    wl, ints = wl[mask].values, ints[mask].values

    if CFG["wavelength_unit"].upper() == "AA":
        wl /= 10.0   # Angstrom -> nm

    # sort by wavelength just in case
    order = np.argsort(wl)
    return pd.DataFrame({"wl": wl[order], "intensity": ints[order]})


```

```
# -- 2. Peak detection ------------------------------------------------------

def _local_snr(intensity: np.ndarray, idx: np.ndarray,
               half_win: int) -> np.ndarray:
    """
    For each peak index, estimate SNR = (peak - local_baseline) / local_noise_std.

    local_baseline : median of the window (robust to other peaks in window)
    local_noise_std: MAD-based std of the window, excluding a guard zone
                     around the peak itself so the peak doesn't inflate the noise.
    """
    n = len(intensity)
    snrs = np.zeros(len(idx))
    guard = max(3, half_win // 8)   # exclude +-guard pts around peak

    for k, i in enumerate(idx):
        lo = max(0, i - half_win)
        hi = min(n, i + half_win)
        window = intensity[lo:hi]

        # Exclude the peak's own guard zone from noise estimate
        gi_lo = max(0, i - lo - guard)
        gi_hi = min(len(window), i - lo + guard + 1)
        noise_region = np.concatenate([window[:gi_lo], window[gi_hi:]])

        if len(noise_region) < 4:
            snrs[k] = 0.0
            continue

        baseline = np.median(noise_region)
        # MAD -> robust std estimate (1.4826 * MAD ~= std for Gaussian noise)
        mad = np.median(np.abs(noise_region - baseline))
        noise_std = 1.4826 * mad

        if noise_std < 1e-12:          # flat region - anything above it is real
            snrs[k] = 999.0
        else:
            snrs[k] = (intensity[i] - baseline) / noise_std
```

```
226
227     return snrs
228
229
230 def detect_peaks(spec: pd.DataFrame) -> pd.DataFrame:
231     """Return DataFrame of detected emission peaks with local SNR
            values."""
232     intensity = spec["intensity"].values
233     wl        = spec["wl"].values
234
235     # Smooth for peak-finding only; raw values are reported in output
236     smoothed = uniform_filter1d(intensity, size=CFG["
            smooth_window_pts"])
237
238     # Coarse prominence filter - first pass to reduce candidate set
239     baseline  = np.percentile(smoothed, 10)
240     dynamic   = smoothed.max() - baseline
241     min_prom  = CFG["peak_prominence_frac"] * dynamic
242
243     idx, _ = find_peaks(
244         smoothed,
245         prominence = min_prom,
246         width      = CFG["peak_min_width_pts"],
247     )
248
249     if len(idx) == 0:
250         print("[WARN] No peaks found after prominence filter. "
251               "Try lowering peak_prominence_frac.")
252         return pd.DataFrame(columns=["wl_peak", "intensity_peak",
253                                      "prominence", "local_snr"])
254
255     proms = peak_prominences(smoothed, idx)[0]
256
257     # Local SNR filter - rejects noise bumps that survived prominence
            filter
258     min_snr  = CFG["peak_min_snr"]
259     half_win = CFG["peak_snr_window_pts"]
260     snrs     = _local_snr(smoothed, idx, half_win)
261
262     before = len(idx)
263     if min_snr is not None:
264         keep = snrs >= min_snr
265         idx, proms, snrs = idx[keep], proms[keep], snrs[keep]
266         rejected = before - len(idx)
267         if rejected:
```

```
268             print(f"[INFO] SNR filter (>={min_snr}) rejected {
                    rejected} noise "
269                   f"candidate(s), kept {len(idx)}.")
270
271     if len(idx) == 0:
272         print("[WARN] All peaks rejected by SNR filter. "
273               "Try lowering peak_min_snr or peak_prominence_frac.")
274         return pd.DataFrame(columns=["wl_peak", "intensity_peak",
275                                      "prominence", "local_snr"])
276
277     peaks_df = pd.DataFrame({
278         "wl_peak"        : wl[idx],
279         "intensity_peak" : intensity[idx],
280         "prominence"     : proms,
281         "local_snr"      : np.round(snrs, 1),
282     }).sort_values("wl_peak").reset_index(drop=True)
283
284     print(f"[INFO] {len(peaks_df)} peaks passed all filters "
285           f"(SNR >= {min_snr}).")
286     print(f"       SNR range: {peaks_df['local_snr'].min():.1f} - "
287           f"{peaks_df['local_snr'].max():.1f}")
288     return peaks_df
289
290
291 # -- 3. NIST atomic line data
    ------------------------------------------------
292 #
293 # Bundled from NIST ASD (physics.nist.gov/asd) - air wavelengths
    235-900 nm.
294 # Format: (species, wavelength_nm, relative_intensity)
295 # This covers the most diagnostically important lines for plasma work
    .
296 # Add your own lines by extending BUNDLED_NIST_LINES or placing a
297 # custom nist_extra.csv (same columns: species,wl_nist_nm,
    nist_intensity)
298 # in the working directory.
299 #
300 BUNDLED_NIST_LINES = [
301     # -- Hydrogen (Balmer series)
        ------------------------------------------------
302     ("H I",  656.279, 10000), ("H I",  486.135, 3000), ("H I",
        434.047, 1500),
303     ("H I",  410.174,  800), ("H I",  397.007,  500), ("H I",
        388.905,  350),
```

```
304     # -- Helium
           ------------------------------------------------------------

305     ("He I", 587.562, 5000), ("He I", 667.815, 2000), ("He I",
           706.519, 1500),
306     ("He I", 447.148, 1000), ("He I", 471.314,  500), ("He I",
           501.568,  400),
307     ("He I", 492.193,  300), ("He I", 504.774,  200), ("He I",
           388.865,  150),
308     ("He I", 318.774,  100), ("He I", 294.505,   80),
309     # -- Nitrogen (neutral)
           ---------------------------------------------------
310     ("N I",  742.364, 1000), ("N I",  744.229,  900), ("N I",
           746.831,  700),
311     ("N I",  818.487,  600), ("N I",  820.035,  500), ("N I",
           821.634,  500),
312     ("N I",  824.239,  400), ("N I",  868.028,  300), ("N I",
           871.170,  300),
313     ("N I",  868.340,  250), ("N I",  674.002,  200), ("N I",
           645.846,  150),
314     ("N I",  566.663,  100), ("N I",  493.509,   80), ("N I",
           410.975,   70),
315     ("N I",  346.619,   60), ("N I",  314.852,   50),
316     # -- Nitrogen (singly ionized)
           ------------------------------------------------
317     ("N II", 500.515, 500), ("N II", 504.506, 400), ("N II", 567.956,
            300),
318     ("N II", 570.007, 300), ("N II", 568.020, 200), ("N II", 576.232,
            200),
319     ("N II", 399.500, 100), ("N II", 463.054,  80), ("N II", 344.039,
             70),
320     # -- Oxygen (neutral)
           ---------------------------------------------------
321     ("O I",  777.194, 2000), ("O I",  777.417, 1800), ("O I",
           777.539, 1600),
322     ("O I",  844.636, 1500), ("O I",  844.676, 1400), ("O I",
           844.625, 1200),
323     ("O I",  615.820,  800), ("O I",  616.005,  700), ("O I",
           630.030,  600),
324     ("O I",  636.378,  500), ("O I",  645.441,  400), ("O I",
           700.188,  300),
325     ("O I",  725.682,  200), ("O I",  794.762,  150), ("O I",
           822.178,  100),
326     ("O I",  394.861,   80), ("O I",  436.826,   70), ("O I",
           532.048,   60),
327     ("O I",  319.324,   50), ("O I",  297.266,   40),
```

```
328     # -- Oxygen (singly ionized)
            ------------------------------------------------
329     ("O II", 441.492, 300), ("O II", 407.217, 200), ("O II", 373.808,
            150),
330     ("O II", 418.546, 100),
331     # -- Carbon (neutral)
            ------------------------------------------------------
332     ("C I",  247.856,  800), ("C I",  248.317,  700), ("C I",
            833.515,  600),
333     ("C I",  785.258,  500), ("C I",  711.318,  400), ("C I",
            658.760,  300),
334     ("C I",  505.218,  200), ("C I",  477.176,  150), ("C I",
            426.724,  100),
335     ("C I",  833.643,  600), ("C I",  872.095,  400),
336     # -- Carbon (singly ionized)
            ------------------------------------------------
337     ("C II", 426.726, 500), ("C II", 723.642, 300), ("C II", 657.805,
            200),
338     ("C II", 392.068, 150), ("C II", 251.175, 100),
339     # -- Argon (neutral)
            ------------------------------------------------------
340     ("Ar I", 750.387, 5000), ("Ar I", 763.511, 8000), ("Ar I",
            794.818, 4000),
341     ("Ar I", 800.616, 3500), ("Ar I", 801.479, 3000), ("Ar I",
            810.369, 3000),
342     ("Ar I", 811.531, 7000), ("Ar I", 826.452, 2500), ("Ar I",
            840.821, 2000),
343     ("Ar I", 842.465, 4000), ("Ar I", 852.144, 3000), ("Ar I",
            866.794, 2000),
344     ("Ar I", 714.704, 1000), ("Ar I", 696.543, 2000), ("Ar I",
            706.722,  800),
345     ("Ar I", 727.294, 1500), ("Ar I", 738.398, 2000), ("Ar I",
            703.025,  600),
346     ("Ar I", 667.728,  500), ("Ar I", 641.630,  400), ("Ar I",
            549.589,  300),
347     ("Ar I", 518.775,  200), ("Ar I", 487.990,  150), ("Ar I",
            451.073,  100),
348     ("Ar I", 433.356,   80), ("Ar I", 419.832,   70), ("Ar I",
            394.898,   60),
349     # -- Argon (singly ionized)
            -------------------------------------------------
350     ("Ar II", 480.602, 1000), ("Ar II", 487.986, 800), ("Ar II",
            493.319, 600),
351     ("Ar II", 458.990, 500), ("Ar II", 472.687, 400), ("Ar II",
            476.487, 400),
352     ("Ar II", 514.532, 300),
```

```
353     # -- Silicon (neutral) ------------------------------------------------
354     ("Si I",  251.611, 500), ("Si I",  252.411, 400), ("Si I",  288.158, 800),
355     ("Si I",  390.552, 300), ("Si I",  568.446, 200), ("Si I",  637.138, 150),
356     ("Si I",  674.005, 100),
357     # -- Silicon (singly ionized) -----------------------------------------
358     ("Si II", 385.600, 500), ("Si II", 386.260, 400), ("Si II", 413.089, 300),
359     ("Si II", 597.939, 200),
360     # -- Iron (neutral) ---------------------------------------------------
361     ("Fe I",  371.994, 1000), ("Fe I",  373.713, 900), ("Fe I",  374.556, 800),
362     ("Fe I",  375.823, 700), ("Fe I",  382.043, 600), ("Fe I",  404.581, 500),
363     ("Fe I",  438.354, 400), ("Fe I",  526.954, 300), ("Fe I",  532.804, 200),
364     ("Fe I",  537.149, 150), ("Fe I",  542.970, 100),
365     # -- Iron (singly ionized) --------------------------------------------
366     ("Fe II", 238.204, 500), ("Fe II", 259.940, 400), ("Fe II", 261.187, 300),
367     ("Fe II", 274.932, 200),
368     # -- Copper -----------------------------------------------------------
369     ("Cu I",  324.754, 2000), ("Cu I",  327.396, 1500), ("Cu I",  510.554, 1000),
370     ("Cu I",  515.324,  800), ("Cu I",  521.820,  600), ("Cu I",  578.213,  400),
371     # -- Sodium -----------------------------------------------------------
372     ("Na I",  588.995, 5000), ("Na I",  589.592, 4000), ("Na I",  330.237, 300),
373     ("Na I",  818.326,  200), ("Na I",  819.482,  150),
374     # -- Calcium ----------------------------------------------------------
375     ("Ca I",  422.673, 3000), ("Ca I",  442.543, 1000), ("Ca I",  445.478, 800),
376     ("Ca I",  616.908,  500), ("Ca I",  643.907,  400), ("Ca I",  646.257, 300),
```

```
    ("Ca II", 393.366, 5000), ("Ca II", 396.847, 4000), ("Ca II", 854.209, 3000),
    ("Ca II", 866.214, 2000),
    # -- Magnesium ---------------------------------------------------------
    ("Mg I",  285.213, 3000), ("Mg I",  383.826,  800), ("Mg I", 516.732, 1500),
    ("Mg I",  517.268, 1200), ("Mg I",  518.361, 1000),
    ("Mg II", 279.553, 2000), ("Mg II", 280.271, 1500),
    # -- Aluminum ----------------------------------------------------------
    ("Al I",  308.215, 2000), ("Al I",  309.271, 1500), ("Al I", 394.401, 1000),
    ("Al I",  396.152,  800),
    # -- Titanium ----------------------------------------------------------
    ("Ti I",  365.350, 500), ("Ti I",  399.864, 400), ("Ti I", 453.322, 300),
    ("Ti II", 334.941, 400), ("Ti II", 336.121, 350),
    # -- Chromium ----------------------------------------------------------
    ("Cr I",  357.869, 800), ("Cr I",  359.349, 700), ("Cr I", 360.533, 600),
    ("Cr I",  427.480, 400), ("Cr I",  428.972, 350),
]


def query_nist(wl_min: float, wl_max: float) -> pd.DataFrame:
    """
    Build NIST atomic line DataFrame from the bundled line list,
    filtered to the spectrum range and configured elements.
    Optionally merges a user-supplied nist_extra.csv for additional lines.
    """
    records = []
    elem_set = set(CFG["elements"]) if CFG["elements"] else None
    stage_set = set(CFG["ion_stages"])

    for (species, wl_nm, rel_int) in BUNDLED_NIST_LINES:
        parts = species.split()
        elem, stage = parts[0], parts[1]
        if elem_set and elem not in elem_set:
            continue
        if stage not in stage_set:
            continue
        if not (wl_min - 1 <= wl_nm <= wl_max + 1):
```

```
            continue
        if rel_int < CFG["nist_min_intensity"]:
            continue
        records.append({
            "species"       : species,
            "element"       : elem,
            "ion_stage"     : stage,
            "wl_nist_nm"    : wl_nm,
            "nist_intensity": rel_int,
            "type"          : "atomic",
        })

    nist_df = pd.DataFrame(records)

    # Merge optional user extra lines
    extra = Path("nist_extra.csv")
    if extra.exists():
        ex = pd.read_csv(extra)
        nist_df = pd.concat([nist_df, ex], ignore_index=True)
        print(f"[INFO] Merged {len(ex)} lines from nist_extra.csv")

    print(f"[INFO] {len(nist_df)} atomic/ionic reference lines loaded "
          f"(bundled NIST data, {len(set(r['element'] for _,r in
              nist_df.iterrows()))} elements).")
    return nist_df


# -- 4. Molecular line list
    ------------------------------------------------------

def load_molecular_lines() -> pd.DataFrame:
    """
    Load molecular line list from a CSV file.
    Expected columns (flexible): molecule, wavelength_nm,
       intensity_rel, assignment

    Returns empty DataFrame if file not found (molecular matching
       skipped).
    """
    mol_path = Path(CFG["molecular_csv"])
    if not mol_path.exists():
        print(f"[INFO] No molecular line list found at '{mol_path}'.
            "
              f"Skipping molecular matching.\n"
```

```
452                 f"       Download from HITRAN (hitran.org) or LIFBASE,
                        or use the\n"
453                 f"       bundled example: python plasma_oes_analyzer.py
                        --write-mol-template")
454         return pd.DataFrame()
455
456     df = pd.read_csv(mol_path, comment="#", index_col=False)
457     df.columns = [c.strip().lower() for c in df.columns]
458
459     # normalise column names flexibly
460     wl_col  = next((c for c in df.columns if "wave" in c or "wl" in c
            or "nm" in c), None)
461     mol_col = next((c for c in df.columns if "mol" in c or "species"
            in c), None)
462     asgn_col = next((c for c in df.columns if "assign" in c or "band"
            in c or "trans" in c), None)
463     int_col = next((c for c in df.columns if "int" in c or "rel" in c
            or "strength" in c), None)
464
465     if wl_col is None or mol_col is None:
466         print("[WARN] Molecular CSV must have at least 'molecule' and
                'wavelength_nm' columns.")
467         return pd.DataFrame()
468
469     out = pd.DataFrame()
470     out["wl_mol_nm"]     = pd.to_numeric(df[wl_col], errors="coerce")
471     out["molecule"]      = df[mol_col]
472     out["assignment"]    = df[asgn_col] if asgn_col else ""
473     out["mol_intensity"] = pd.to_numeric(df[int_col], errors="coerce"
            ).fillna(0) if int_col else 0
474     out["type"]          = "molecular"
475     out = out.dropna(subset=["wl_mol_nm"])
476
477     print(f"[INFO] {len(out)} molecular lines loaded from '{mol_path
            }'.")
478     return out
479
480
481 # -- 5. Peak matching
        ------------------------------------------------------------
482
483 def match_peaks(peaks: pd.DataFrame, nist: pd.DataFrame, mol: pd.
        DataFrame) -> pd.DataFrame:
484     """
485     For each detected peak, find all candidate atomic and molecular
            lines
```

```
    within +-match_tolerance_nm. Returns a results table.
    """
    tol = CFG["match_tolerance_nm"]
    rows = []

    for _, peak in peaks.iterrows():
        wl   = peak["wl_peak"]
        ints = peak["intensity_peak"]
        prom = peak["prominence"]

        # Atomic matches
        if not nist.empty:
            mask = (nist["wl_nist_nm"] >= wl - tol) & (nist["
               wl_nist_nm"] <= wl + tol)
            atomic_matches = nist[mask].copy()
        else:
            atomic_matches = pd.DataFrame()

        # Molecular matches
        if not mol.empty:
            mask2 = (mol["wl_mol_nm"] >= wl - tol) & (mol["wl_mol_nm"
               ] <= wl + tol)
            mol_matches = mol[mask2].copy()
        else:
            mol_matches = pd.DataFrame()

        n_atomic = len(atomic_matches)
        n_mol    = len(mol_matches)

        if n_atomic == 0 and n_mol == 0:
            # Unidentified peak
            rows.append({
                "peak_wl_nm"     : round(wl, 3),
                "peak_intensity" : round(ints, 4),
                "prominence"     : round(prom, 4),
                "type"           : "unidentified",
                "species"        : "-",
                "db_wl_nm"       : np.nan,
                "delta_nm"       : np.nan,
                "db_intensity"   : np.nan,
                "assignment"     : "",
                "local_snr"      : peak["local_snr"],
            })
        else:
            # Atomic candidates
            for _, a in atomic_matches.iterrows():
```

```
                rows.append({
                    "peak_wl_nm"     : round(wl, 3),
                    "peak_intensity" : round(ints, 4),
                    "prominence"     : round(prom, 4),
                    "type"           : "atomic",
                    "species"        : a["species"],
                    "db_wl_nm"       : round(a["wl_nist_nm"], 3),
                    "delta_nm"       : round(wl - a["wl_nist_nm"], 3)
                      ,
                    "db_intensity"   : a["nist_intensity"],
                    "assignment"     : "",
                    "local_snr"      : peak["local_snr"],
                })
            # Molecular candidates
            for _, m in mol_matches.iterrows():
                rows.append({
                    "peak_wl_nm"     : round(wl, 3),
                    "peak_intensity" : round(ints, 4),
                    "prominence"     : round(prom, 4),
                    "type"           : "molecular",
                    "species"        : m["molecule"],
                    "db_wl_nm"       : round(m["wl_mol_nm"], 3),
                    "delta_nm"       : round(wl - m["wl_mol_nm"], 3),
                    "db_intensity"   : m["mol_intensity"],
                    "assignment"     : m.get("assignment", ""),
                    "local_snr"      : peak["local_snr"],
                })

    return pd.DataFrame(rows)


# -- 6. Confidence scoring
   ------------------------------------------------------

def score_confidence(results: pd.DataFrame) -> pd.DataFrame:
    """
    Add a simple confidence score based on:
      - Deltalambda distance to database line  (closer = better)
      - Whether the same species has multiple peak hits (multiplet
        confirmation)
    """
    if results.empty:
        return results

    # Count how many distinct peaks matched each species
    species_hits = (
```

```
        results[results["type"].isin(["atomic","molecular"])]
        .groupby("species")["peak_wl_nm"]
        .nunique()
        .rename("n_peaks_matched")
    )
    results = results.join(species_hits, on="species")
    results["n_peaks_matched"] = results["n_peaks_matched"].fillna(0)
       .astype(int)

    def _score(row):
        if row["type"] == "unidentified":
            return "-"
        tol = CFG["match_tolerance_nm"]
        d   = abs(row["delta_nm"]) if not np.isnan(row["delta_nm"])
           else tol
        # proximity score: 1.0 at Delta=0, 0.0 at Delta=tol
        proximity = max(0.0, 1.0 - d / tol)
        # multiplet bonus: log-scaled up to 0.5
        multiplet  = min(0.5, 0.25 * np.log1p(row["n_peaks_matched"])
           )
        score = proximity * 0.7 + multiplet * 0.3
        if score > 0.75:
            return "HIGH"
        elif score > 0.40:
            return "MEDIUM"
        else:
            return "LOW"

    results["confidence"] = results.apply(_score, axis=1)
    return results


# -- 7. Plot
   ------------------------------------------------------------------------


COLORS = {
    "atomic"      : "#2196F3",   # blue
    "molecular"   : "#4CAF50",   # green
    "unidentified": "#9E9E9E",   # grey
}

def make_plot(spec: pd.DataFrame, peaks: pd.DataFrame, results: pd.
   DataFrame):
    fig, ax = plt.subplots(figsize=(18, 6))
    fig.patch.set_facecolor("#1a1a2e")
```

```
    ax.set_facecolor("#1a1a2e")
    for spine in ax.spines.values():
        spine.set_edgecolor("#444")

    # Spectrum
    ax.plot(spec["wl"], spec["intensity"],
            color="#e0e0e0", linewidth=0.7, label="Spectrum", zorder
               =2)

    # Annotate each unique peak
    already_labelled = {"atomic": False, "molecular": False, "
       unidentified": False}
    annotated_peaks  = set()

    for _, row in results.iterrows():
        pk = row["peak_wl_nm"]
        if pk in annotated_peaks:
            continue
        annotated_peaks.add(pk)

        ptype  = row["type"]
        color  = COLORS[ptype]
        label  = row["species"] if row["type"] != "unidentified" else
            "?"
        conf   = row.get("confidence", "")

        # Vertical marker at peak
        ax.axvline(pk, color=color, alpha=0.35, linewidth=0.8, zorder
           =1)

        # Text label (staggered height to reduce overlap)
        y_frac  = 0.55 + 0.30 * (hash(label) % 7) / 6
        y_pos   = spec["intensity"].max() * y_frac
        snr_str = f'SNR={row["local_snr"]:.0f}' if not pd.isna(row.
           get('local_snr', float('nan'))) else ''
        ax.text(pk, y_pos, f"{label}\n{pk:.2f}\n{snr_str}",
                fontsize=5.5, color=color, ha="center", va="bottom",
                rotation=90, clip_on=True, zorder=5)

        # Legend entries (one per type)
        if not already_labelled[ptype]:
            ax.axvline(pk, color=color, linewidth=1.5,
                       label={"atomic":"Atomic/Ionic","molecular":"
                          Molecular","unidentified":"Unidentified"}[
                          ptype])
            already_labelled[ptype] = True
```

```
652
653     ax.set_xlabel("Wavelength (nm)", color="#ccc", fontsize=11)
654     ax.set_ylabel("Intensity (a.u.)",  color="#ccc", fontsize=11)
655     ax.set_title("Plasma OES - Species Identification",
656                  color="white", fontsize=13, pad=10)
657     ax.tick_params(colors="#aaa")
658     ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(5))
659     ax.grid(which="major", color="#333", linewidth=0.5)
660     ax.grid(which="minor", color="#222", linewidth=0.3)
661
662     legend = ax.legend(facecolor="#2a2a3e", edgecolor="#555",
663                        labelcolor="white", fontsize=9, loc="upper
                              right")
664
665     plt.tight_layout()
666     plt.savefig(CFG["output_plot"], dpi=CFG["dpi"], bbox_inches="
           tight")
667     print(f"[INFO] Plot saved -> {CFG['output_plot']}")
668
669
670 # -- 8. Summary report to console
        ------------------------------------------------
671
672 def print_summary(results: pd.DataFrame):
673     if results.empty:
674         print("\n[RESULT] No peaks were matched.")
675         return
676
677     high_conf = results[results["confidence"] == "HIGH"]
678     print("\n" + ""*70)
679     print("  PLASMA OES ANALYSIS - SUMMARY")
680     print(""*70)
681
682     # Confirmed species (HIGH confidence, multiple peaks)
683     confirmed = (
684         high_conf[high_conf["type"].isin(["atomic","molecular"])]
685         .groupby(["species","type"])
686         .agg(matched_peaks_nm=("peak_wl_nm", lambda x: sorted(x.
               unique().tolist())),
687              snr_min=("local_snr", "min"),
688              snr_max=("local_snr", "max"))
689         .reset_index()
690     )
691     if not confirmed.empty:
692         print("\n    CONFIRMED SPECIES (HIGH confidence):")
693         for _, r in confirmed.iterrows():
```

```
694             peaks_str = ", ".join(f"{w:.2f}" for w in r["
                    matched_peaks_nm"])
695             snr_str   = (f"  SNR {r['snr_min']:.0f}-{r['snr_max']:.0f
                    }"
696                          if r['snr_min'] == r['snr_max']
697                          else f"  SNR {r['snr_min']:.0f}-{r['snr_max
                             ']:.0f}")
698             print(f"     [{r['type'].upper():10s}]  {r['species
                    ']:<18}  peaks: {peaks_str}{snr_str}")
699     else:
700         print("\n  No HIGH-confidence species found. Broaden elements
                list or lower thresholds.")
701
702     n_unid = results[results["type"] == "unidentified"]["peak_wl_nm"
            ].nunique()
703     if n_unid:
704         wls = sorted(results[results["type"]=="unidentified"]["
                peak_wl_nm"].unique())
705         print(f"\n  ?  UNIDENTIFIED peaks ({n_unid}): "
706               + ", ".join(f"{w:.2f}" for w in wls))
707         print("     -> Add more elements to CFG['elements'], or check
                molecular CSV.")
708
709     print(f"\n  Full results -> {CFG['output_table']}")
710     print(""*70 + "\n")
711
712
713 # -- 9. Molecular CSV template writer
        ----------------------------------------
714
715 MOLECULAR_TEMPLATE = """\
716 # Molecular line list template for plasma_oes_analyzer.py
717 # Sources:
718 #   HITRAN:  https://hitran.org  (download "line-by-line" data)
719 #   LIFBASE: https://www.me.berkeley.edu/gri_mech/data/lifbase.html
720 #   NIST:    https://webbook.nist.gov
721 #
722 # NOTE: assignment strings that contain commas must be quoted with
        double-quotes.
723 # Common plasma molecular bands (expand with your HITRAN/LIFBASE data
        ):
724 molecule,wavelength_nm,intensity_rel,assignment
725 OH,306.36,1.0,A2S-X2P (0-0) bandhead
726 OH,308.90,0.8,A2S-X2P (0-0) R-branch
727 OH,281.10,0.3,A2S-X2P (1-0) bandhead
728 N2,315.93,1.0,C3P-B3P (0-2) 2nd-pos bandhead
```

```
N2,337.13,1.0,C3P-B3P (0-0) 2nd-pos bandhead
N2,357.69,0.7,C3P-B3P (0-1) 2nd-pos bandhead
N2,380.49,0.5,C3P-B3P (1-3) 2nd-pos bandhead
N2+,391.44,1.0,B2S-X2S (0-0) 1st-neg bandhead
N2+,427.81,0.5,B2S-X2S (0-1) 1st-neg bandhead
CN,358.39,1.0,B2S-X2S (0-0) violet bandhead
CN,386.19,0.8,B2S-X2S (1-1) violet bandhead
CN,388.34,0.6,B2S-X2S (0-1) violet bandhead
C2,469.41,0.6,Swan d3P-a3P (1-0) bandhead
C2,516.52,1.0,Swan d3P-a3P (0-0) bandhead
C2,563.55,0.7,Swan d3P-a3P (0-1) bandhead
NH,336.00,0.5,A3P-X3S (0-0) bandhead
NO,236.30,1.0,gamma A2S-X2P (0-0) bandhead
NO,247.90,0.7,gamma A2S-X2P (1-1) bandhead
"""

def write_mol_template():
    p = Path("molecular_lines.csv")
    p.write_text(MOLECULAR_TEMPLATE)
    print(f"[INFO] Molecular template written -> {p}")
    print("       Edit/extend it with your HITRAN/LIFBASE data, then re-run.")


# -- Main ------------------------------------------------------------------


def main():
    parser = argparse.ArgumentParser(description="Plasma OES species identifier")
    parser.add_argument("spectrum", nargs="?", help="Path to spectrum CSV")
    parser.add_argument("--write-mol-template", action="store_true",
                        help="Write a starter molecular_lines.csv and exit")
    parser.add_argument("--clear-nist-cache", action="store_true",
                        help="Delete nist_cache.csv to force a fresh NIST query")
    args = parser.parse_args()

    if args.write_mol_template:
        write_mol_template()
        return

    if args.clear_nist_cache:
```

```
        cache = Path("nist_cache.csv")
        if cache.exists():
            cache.unlink()
            print("[INFO] NIST cache cleared.")

    if not args.spectrum:
        parser.print_help()
        sys.exit(1)

    # Pipeline
    print(f"\n[INFO] Loading spectrum: {args.spectrum}")
    spec   = load_spectrum(args.spectrum)
    wl_min = spec["wl"].min()
    wl_max = spec["wl"].max()
    print(f"[INFO] Wavelength range: {wl_min:.1f} - {wl_max:.1f} nm
       |  {len(spec)} data points")

    peaks  = detect_peaks(spec)
    if peaks.empty:
        sys.exit("[ERROR] No peaks found. Check your file or lower
           peak_prominence_frac.")

    nist   = query_nist(wl_min, wl_max)
    mol    = load_molecular_lines()

    results = match_peaks(peaks, nist, mol)
    results = score_confidence(results)

    results.to_csv(CFG["output_table"], index=False)
    print(f"[INFO] Results table saved -> {CFG['output_table']}")

    make_plot(spec, peaks, results)
    print_summary(results)


if __name__ == "__main__":
    main()


#
# README / QUICK-START
#
#
# STEP 1 - Generate the molecular line list template:
#   python plasma_oes_analyzer.py --write-mol-template
```

```
#   This creates molecular_lines.csv with common plasma bands pre-filled.
#   Extend it with your HITRAN/LIFBASE downloads.
#
# STEP 2 - Run on your spectrum:
#   python plasma_oes_analyzer.py my_spectrum.csv
#
# STEP 3 - Outputs:
#   oes_analysis.png   -> annotated spectrum plot (atomic=blue, molecular=green)
#   oes_results.csv    -> full peak-by-peak results with confidence scores
#   nist_cache.csv     -> NIST lines cached (delete with --clear-nist-cache)
#
# TUNING TIPS:
#   - Too many false peaks?  Raise  peak_prominence_frac  (e.g. 0.05)
#   - Missing weak lines?    Lower  peak_prominence_frac  (e.g. 0.01)
#   - Wrong wavelength?      Check  match_tolerance_nm   (default 0.3 nm)
#   - Missing an element?    Add it to  elements  list in CFG
#   - Confident an ID is wrong? NIST intensity rankings are relative; always
#     verify with at least 2 lines from the same species (multiplet rule).
#
# MOLECULAR DATA SOURCES:
#   HITRAN:   https://hitran.org/lbl/       (register free, download CSV)
#   LIFBASE:  https://www.me.berkeley.edu/gri_mech/data/lifbase.html
#   NIST WebBook: https://webbook.nist.gov
#   Spectraplot:  https://spectraplot.com    (visual check tool)
#
#
```

Since this code was inaccurate (tested with a simple fluorescent spectrum) and too sensitive, the author utilized ChatGPT to revise it, resulting in more code that led to a new picture shown in figure K.2.

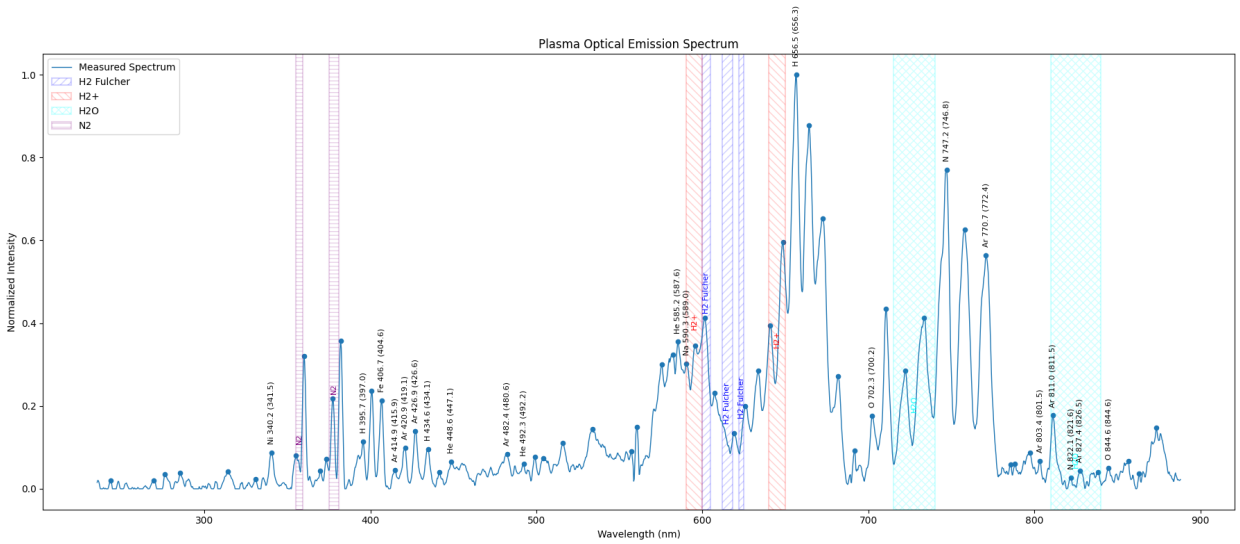

Figure K.2 Hydrogen plasma spectral lines identified by the Python code. This seems better than the previous code, although it is not clear enough to be sure. Further development is required, including testing the code on calibrated spectral lamps.