LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TH-687161

Sensitivity of Inferred Electron Temperature from X-ray Emission of NIF Cryogenic DT Implosions

M. Klem

March 30, 2016

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

UNIVERSITY OF DALLAS

Sensitivity of inferred electron temperature from X-ray emission of NIF cryogenic DT implosions

Submitted in partial fulfillment of the Bachelor of Science Degree

in

Physics

By

Michael Klem

May 2015

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Abstract

The National Ignition Facility (NIF) at the Lawrence Livermore National Laboratory seeks to achieve thermonuclear ignition through inertial confinement fusion. The accurate assessment of the performance of each implosion experiment is a crucial step. Here we report on work to derive a reliable electron temperature for the cryogenic deuterium-tritium implosions completed on the NIF using the x-ray signal from the Ross filter diagnostic. These X-rays are dominated by bremsstrahlung emission. By fitting the x-ray signal measured through each of the individual Ross filters, the source bremsstrahlung spectrum can be inferred, and an electron temperature of the implosion hot spot inferred. Currently, each filter is weighted equally in this analysis. We present work quantifying the errors with such a technique and the results from investigating the contribution of each filter to the overall accuracy of the temperature inference. Using this research, we also compare the inferred electron temperature against other measured implosion quantities to develop a more complete understanding of the hot-spot physics.

Acknowledgments

I would would like to first and foremost thank to my summer internship advisor Tammy Ma. She gave me an incredible chance to spend a wonderful summer in Livermore learning about inertial confinement fusion and to experience the inspiring environment of a national laboratory. Her persistent willingness to answer my many questions, as well as her perpetual cheerfulness made my summer research pleasant as well as interesting.

Thank you to Professors Sally Hicks, Richard Olenick, and Jacob Moldenhauer of the University of Dallas for their years of kind physics instruction. It was only through their lucid teaching that I became prepared to conduct my research. Their enthusiasm for physics has been a great encouragement and advising has long proved to be invaluable.

I am grateful for the financial support of the Lawrence Livermore Summer Student Program.

Lastly, thank you to my parents, Daniel and Karen Klem, for your love and kind support. Thank you for you a lifetime of encouragement, and invaluable advice.

Table of Contents

List of Figures

Introduction

The study of nuclear fusion has been of vital national interest for many years. The potential for nuclear fusion to change the world was first demonstrated in November of 1952 with the detonation of the first hydrogen bomb[1]. The use of fusion energy though has not been purely destructive. The amount of energy released in a fusion reaction is immense, and can not only power the most destructive weapons known to mankind, but could in the future, become a nearly infinite source of clean and safe energy. To this latter effect, the National Ignition Facility was developed at Lawrence Livermore National Laboratory.

Nuclear fusion reactions only occur under extreme temperatures and densities. For this reason it is only observed in the heart of stars and in the detonation of a thermonuclear device. These conditions are required due to the nature of the reaction. Unlike in fission, when a nucleus splits in half, in fusion two nuclei join to form a new element. For this union to occur, the nuclei must overcome the coulomb barrier. The coulomb barrier is the repulsive force a charge's nucleus "feels" as it approaches a second charge's nucleus. This force is due to the coulomb repulsion between the protons in each nuclei. The height of the coulomb barrier for a fusion reaction is on the order of one million electron volts (1 MeV). Under the classical understanding, nuclei in a fusion reaction would be required to have an energy in excess of the height of the coulomb barrier. Without sufficient energy, the nuclei would reach a classical "turning point" and would be repelled away from the other nuclei. An illustration of the coulomb barrier is seen in Figure 1.

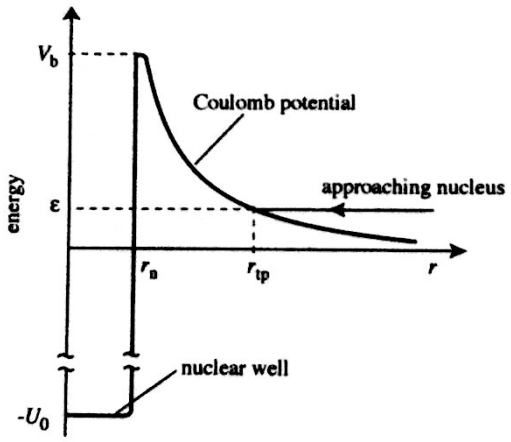


Figure 1: Under the classical understanding, an approaching nucleus must pass over the coulomb potential barrier. By a consideration of quantum mechanics the ability of the nucleus to "tunnel" through the barrier is understood. (needs citation)

However, fusion reaction take place in solar plasma at energies below 1 Mev. This is possible due to quantum tunneling and the uncertainty principle. A nucleus can tunnel through the coulomb barrier because of the uncertainty in the position of the protons in the nucleus. As might be expected, the higher the energy of the nucleus, the more likely it is that the nucleus will tunnel through the potential barrier[3].

While this tunneling effect allows fusion reactions to occur at substantially lower energies than might be expected from a classical consideration, the energy released in fusion reactions is immense. To create self-sustained nuclear fusion, conditions like those found in the center of stars are required. Our nearest star, the sun, has a core temperature of about 15 million degrees kelvin, which is necessary for the proton-proton chain reaction. In a deuterium tritium (DT) reaction (currently being investigated at the NIF), tunneling becomes most probable when the nuclei have 64 keV energies. Such conditions for fusion require an extraordinary environment available

previously only in the heart of stars, and in the midst of thermonuclear detonations. In the hopes of better understanding fusion science and because of it's possible applications to government weapons programs, laboratory controlled fusion projects have been explored since the 1950's. One such approach explored at Lawrence Livermore National Laboratory is inertial confinement fusion (ICF).

In the ICF regime, high-powered lasers are used to compress and heat nuclear fuel to fusion sustainable conditions. While there are two different methods of heating the outside of the target, one being through direct laser irradiance, the second being from indirect irradiance from a laser driven x-ray source, direct and indirect drive respectively, the fundamental theory of the fusion process is the same. In both cases the outside of the target is ablated off by incident radiation. It is blown off in a rocket like expansion [5], which both eject the outside of the fuel capsule and, via action-reaction, implodes the inside of the target. As the target collapses in on itself, the pressure at the core rapidly increases. The work done to compress the volume of the target remains as heat, increasing as the volume decreases. Under these conditions a "hotspot" is formed at the core of the fuel capsule, where, the fusion reaction begins. The NIF currently employs the ICF indirect drive approach in a campaign to achieve thermonuclear ignition. Ignition is a self-sustained fusion reaction inside the fuel capsule when the fusion reaction is sufficiently virulent to allow thermonuclear burn to propagate out from the hot core into the surrounding cooler fuel. Figure 2 illustrates the steps of an ICF implosion and Figure 3 illustrates the difference between direct drive and indirect drive methods.

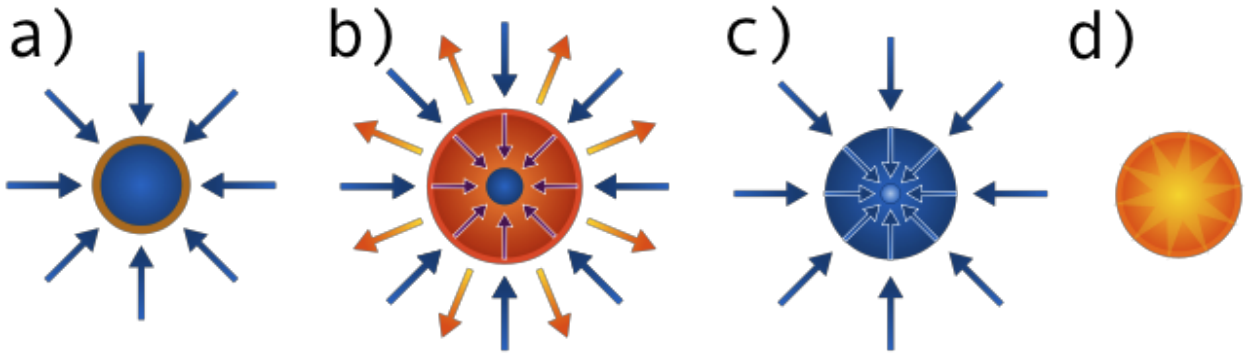


Figure 2. a) the laser light, or X-rays illuminate the outside of the capsule. b) The outside shell is ablated off causing rocket like expansion of the outside of the fuel capsule. “Blowoff” is ejected which the remainder of the capsule in force inwards. c) As the capsule reaches max compression, both the pressure and the temperature massively increase. c) A hotspot is generated at the center of the capsule and fusion begins. (needs citation)

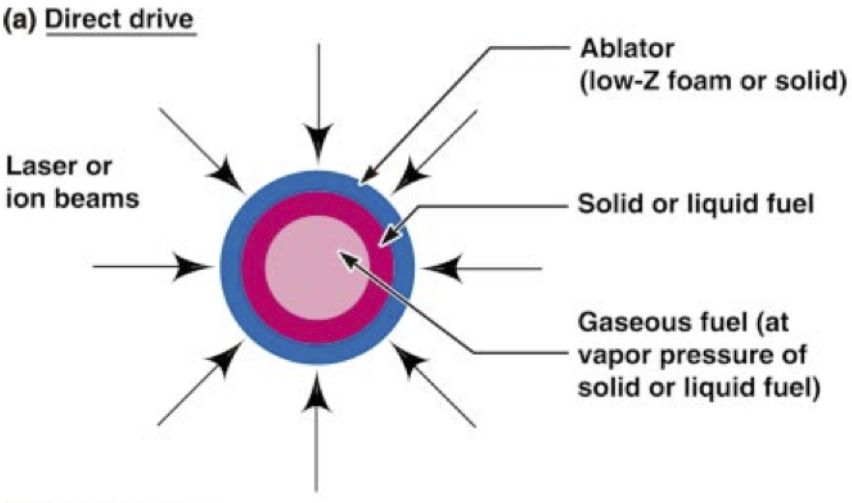


Figure 3. a) In direct drive, laser beams are incident on the outside of the capsule. This method is advantage due to higher energy coupling efficiency and reduces laser plasma interaction effects. (needs citation)

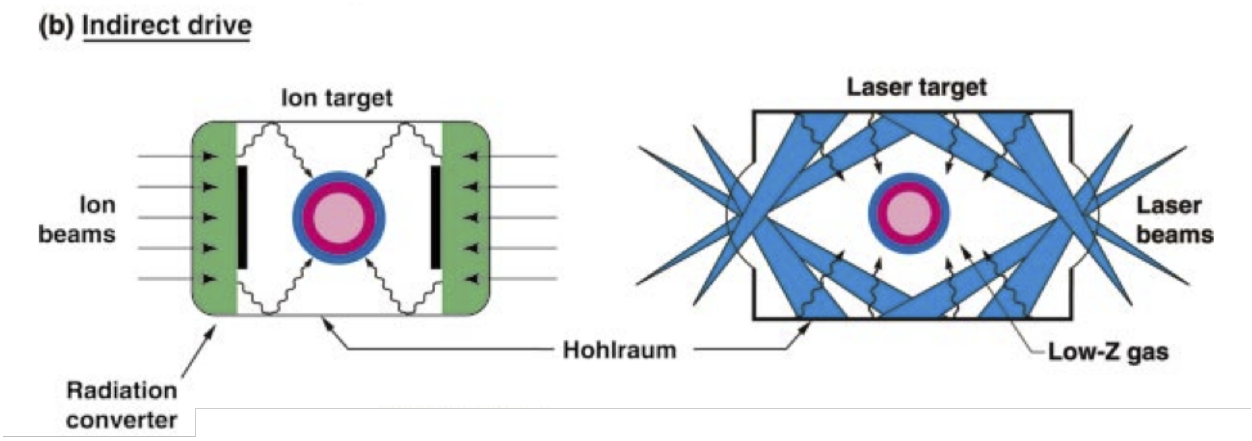


Figure 3. b) In indirect drive, laser beams are incident on the inside of gold capsule called a hohlraum. The hohlraum then emits X-rays which bath the fuel capsule in nearly uniform light. It is advantageous because it relaxes the requirement on beam uniformity and reduces hydrodynamic instability. (needs citation)

While the general theory of ICF indirect drive fusion is simple, there are many fundamental features of ICF, which make achieving ignition a prodigious challenge. These features include laser beam uniformity discrepancies, laser plasma interactions, and hydrodynamic instabilities. If the power delivered by each of the beams is not perfectly identical, the laser can drive an asymmetric implosion. In this case energy can be wasted by translational movement of the target during an implosion or by pushing the target into a non-spherical shape by either pressing too heavily on the top, bottom or sides. Asymmetric drive can be caused by poor beam uniformity or also via laser plasma interactions. These interactions can come in the form of stimulated Raman scattering, stimulated Brillouin scattering, or cross beam energy transfer. In stimulated Brillouin, scattering is caused by laser light scattering off ion sound waves, while stimulated raman scattering is caused by laser light scattering off electron waves. Cross beam energy transfer occurs when energy from laser beam is coupled into that of an intersecting beam. All of these laser plasma interactions can lead to power loss as well as asymmetric drive. In addition to the challenges of beam uniformity discrepancies and laser plasma interactions, hydrodynamic

instabilities can cripple the integrity of an implosion. Hydrodynamic instabilities are instabilities in fluid flow, in this case, plasma. These are important at the interfaces of different layers of the target. As instabilities grow during an implosion, the interfaces of the boundaries degrade and the symmetry of the implosion deteriorates. These instabilities can cause mix within the target, where the material of the outside of the target is mixed in with the nuclear fuel. This mix can make ignition impossible.

For these reasons, and more, ignition has not been achieved at the NIF. As a result strong diagnostic techniques are required to diagnose how close the ignition campaign is to success. Chief among these needed measures is a robust measurement of the temperature of the core of the implosion. The term temperature is however partially ambiguous. Because the entire target is plasma at the time of the implosion, the ions and electrons in the core of the implosion are decoupled. This allows each to have a different thermal distribution. It is currently unknown whether the two are in thermal equilibrium. Thus two temperature measurements are made T_{Ion} and T_{e-}, the ion temperature and the electron temperature. It would be expected that the ions and the electrons would be in thermal equilibrium, but this is not known. An accurate measurement of the electron temperature is needed because currently there is a discrepancy between the electron and ion temperatures.

The ion temperatures are measured using a Neutron Time Of Flight detector (NTOF). The NTOF detector measures the neutron energy spectrum. The energy spectrum is Doppler broadened because the fuel ions acquire a thermal distribution.

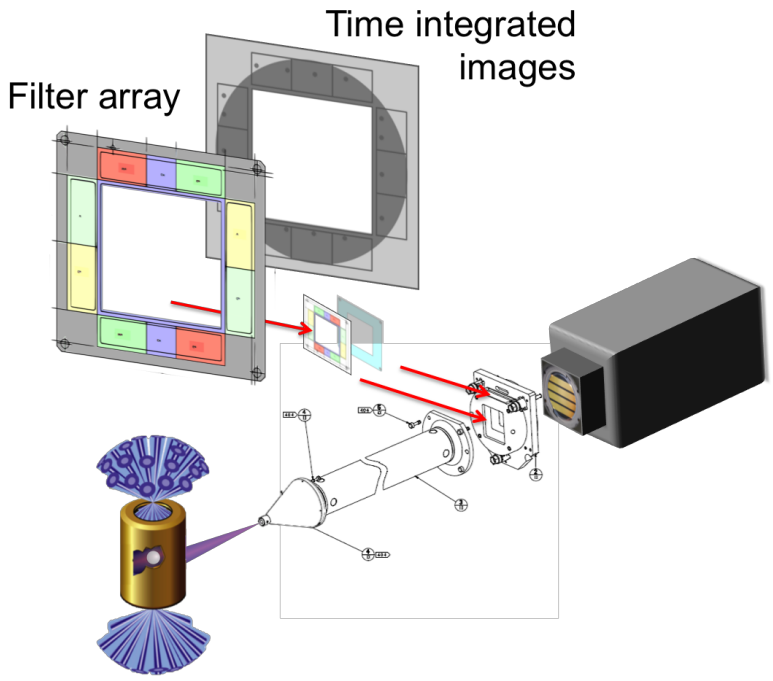


Figure 4: X-rays are emitted from the gold hohlraum, pass through the front stage of the Ross Filter Pair diagnostic, pass through the filter array, and are recorded on the image plate, making time integrated images. (needs citation)

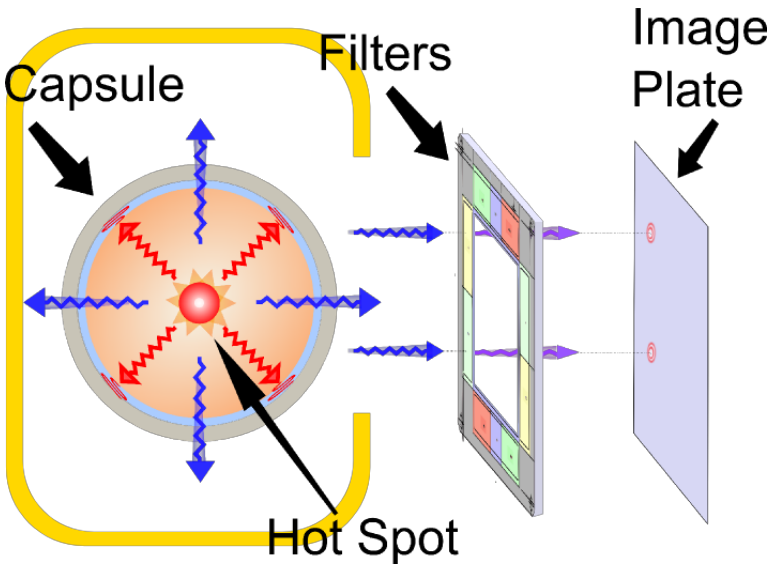


Figure 5: X-rays from the center of the implosion are emitted. Low energy X-rays (red) are attenuated by the ablator shell. Higher energy X-rays pass out of the target, through the hohlraum, filter array, and are recorded on the image plate.

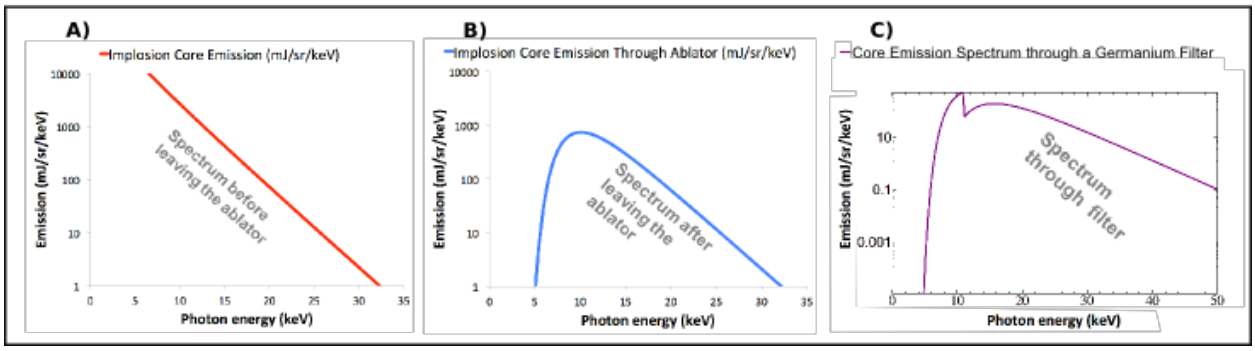


Figure 6: The spectrum changes as the X-rays passes from the center of the implosion to the detector. Initially unattenuated (A), the low energy portion of the spectrum is filtered out by absorption in the ablator as seen in (B). After the X-rays pass through the filter, the spectrum is again attenuated. The sharp drop seen in (C) is the K-edge of the material. It corresponds to an energy just above the binding energy of the K-shell electrons.

The focus of the summer project was the development of a code to calculate the uncertainty in the inferred electron temperatures from the Ross Filter Pair diagnostic. The Ross Filter Pair Diagnostic is a simple device, which measures the x-ray flux coming from a NIF implosion. The diagnostic is divided into twelve sections. Each section has a "view" of the implosion out through the entrance hole of the diagnostic. Each section has two parts, an attenuation filter, and an x-ray image plate. Part of the x-ray flux from the implosion enters the detector, passes through the first stage, then through the x-ray filters, and then is recorded on the x-ray image plate. There are six different types of attenuation filters. These are, vanadium, copper, germanium, molybdenum and two thicknesses of aluminum filters. These are repeated twice in the detector to insure that a signal is recorded through at least one of them. The attenuation filters are chosen carefully for a particular thickness and atomic number, to have a very particular attenuation characteristic.

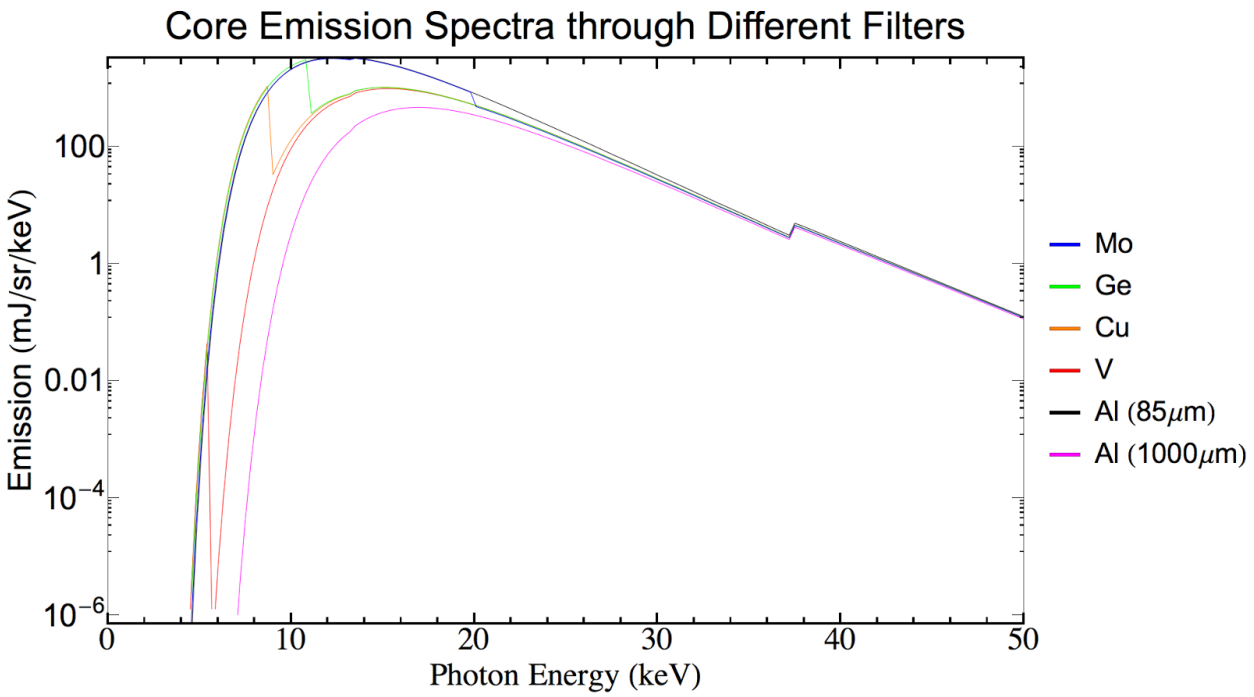


Figure 7: The x-ray spectra through the different filters are each shaped differently by the filter attenuation. This shaping allows the research to inferrer the x-ray spectrum from the signals recorded on the image plate.

The front stage prepares the X-rays for attenuation and detection. While the detector's image plate is recessed from the site of the implosion, the front stage has a long "snout" which protrudes out of the target chamber wall to within centimeters of the implosion. It is designed to capture a small solid angle of the x-ray flux coming from the implosion. The X-rays pass through a preliminary kapton plastic debris shield, which protects the detector, and are then imaged through a pinhole array and collimator. The pinhole array allows multiple images of the implosion to be recorded on the x-ray image plate in the same way that early pinhole cameras recorded images on photographic plates. The collimator is used to block as many of the high-energy neutrons from the implosion as possible from hitting the image plate and prematurely developing it.

The Ross Filter Pair diagnostic has three main features. It is spatially resolved, time integrated and absolutely calibrated. To be spatially resolved means that image information about the implosion is recorded in two dimensions. It is spatially resolved in the same way that a photograph is spatially resolved as opposed to a single line in that photograph. To be time integrated, means that the diagnostic records x-ray irradiance for the entirety of the implosion. It records radiation from the beginning of the implosion to the end. It is analogous to leaving the shutter open on a camera pointed at a light bulb as the bulb is momentarily turned on. The diagnostic records the x-ray flux as it increases like the light from a light bulb as it heats up and continues to record as it reaches peak brightness and then dims. To be absolutely calibrated means that an exact measurement can be made of the signal level received on the X-ray image plate[4]. The Ross Filter Pair diagnostic records information on the shape of the implosion as well as the absolute signal level. This latter part is used to infer the electron temperature of the implosion.

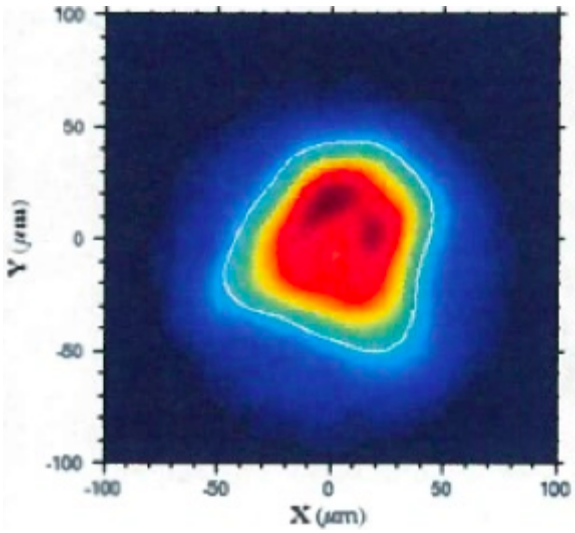


Figure 8: Self emission at peak compression is recorded on the image plate of the detector. Imaged through a pinhole, the X-ray flux from the implosion in recorded in two dimensions. Recording over the entire period of the implosion, it is seen that the center radiates most intensely. This is a product of its greater temperature. (needs citation)

This is possible because of the important link between the X-rays and the plasma electron temperature. In plasma, at the core of the implosion, all the deuterium and tritium gas atoms are fully ionized. This produces a “soup” of free electrons and ions that can move independent of each other. Due to conservation of momentum, the velocities of the electrons are much higher than those of the ions. Due to the electrostatic force of the ions on the moving electrons, a moving electron will be accelerated around a near ion. To conserve energy, the electron radiates energy in the form of X-rays. This radiation is known as Bremsstrahlung radiation. The radiation is emitted in a continuous spectrum with a frequency peak, which is dependent on the temperature of the plasma. When this radiation is emitted from the hot plasma at the center of the implosion and is incident on the x-ray image plate, the image plate can only record the total energy deposited on it, not the spectrum. With the use of filters, which attenuate preferentially different parts of the x-ray band, one can infer the initial Bremsstrahlung spectrum.

Previously using this method, an electron temperature was inferred. This temperature was however, inconsistent with the ion temperature, being too low. It was not known how sensitive the inferred electron temperature was to the choice of filters used in the diagnostic. It was not known if one filter could be throwing off the final inferred temperature.

This thesis discusses the work done to evaluate the sensitivity of the inferred electron temperature to filter choice. The chosen approach to solve this problem was to build a monte carlo method code to simulate the system, infer and electron temperature, and then evaluate how that temperature changes with choice of filters given random error in the recorded signal.

The analysis was completed using the programing language Mathematica. While it offered many of the advantages of a high level programing language, it was the cause of significant difficulty as the author needed to first learn the language to complete the summer research. It had the additional disadvantage, like other compiled languages, of running comparatively slowly. This necessitated a fair bit of cleverness to accelerate the run time of the program.

DATA ACQUISITION:

The data used in this research was both synthetic and experimental. After methods were developed with synthetic data, they were applied to experimental results to look for trends in implosion performance. To investigate the sensitivity of electron temperature measurements to filter configuration, synthetic data was required. To examine the errors in the calculation of the inferred electron temperature, the correct value needed to be known. Impossible under an experimental setting, the raw data from the detector was simulated, assuming an x-ray spectrum and knowing the attenuation characteristics and dimensions of the detector. By assuming a spectrum, an electron temperature was also assumed. This "true" electron temperature was then compared to the inferred values. Once the synthetic data was created and the error in the inferred electron temperature was analysed, data from NIF implosions was examined using the methods developed previously. Having built confidence in the electron temperature determining part of the code (by running error free synthetic values through it and inferring the original generating electron temperature), it was used to infer electron temperature from NIF implosion data.

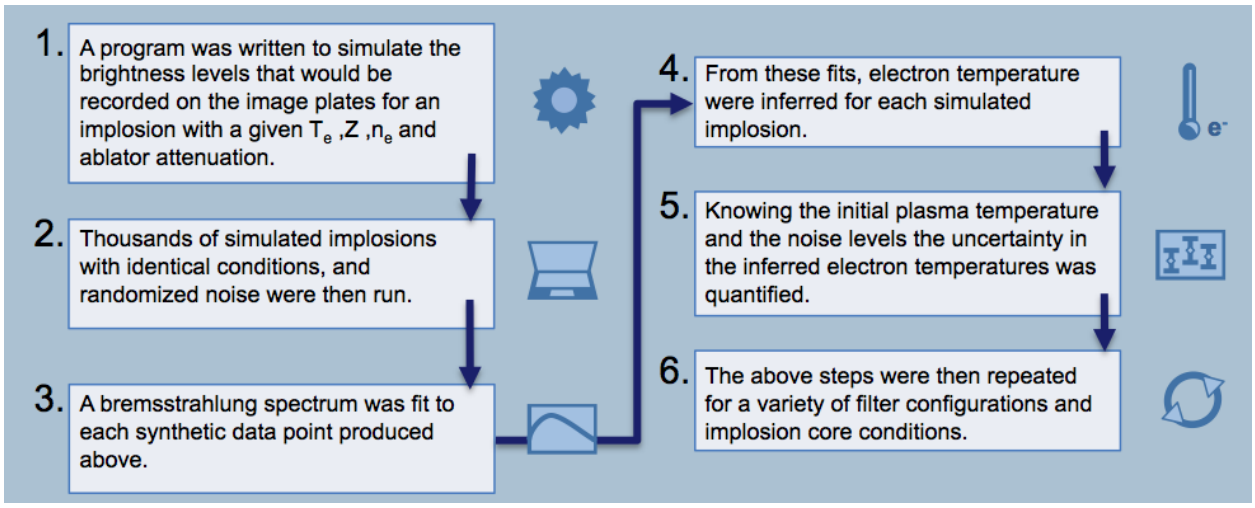


Figure 9: A visual description of the code used to calculate inferred electron temperature sensitivity to filter configuration.

Developing Synthetic Data:

To infer an electron temperature, one must fit a bremsstrahlung spectrum to the signal level data from the detector. The electron temperature can easily be inferred from the slope of the spectrum. To this end, the program must be capable of producing a synthetic spectrum. This is accomplished using the following equation for a Bremsstrahlung spectrum.

$$f_{spec}(T, E_{photon}) = \frac{0.0001*2.8*\rho^2*(5.04*10^{22})*e^{\frac{-E_{photon}}{T}}\left(\frac{4\pi}{3}\right)*(R*0.0001)^3(B_{wdth}*0.000000000001)*e^{-D_{Opt}}*\left(\frac{10.85}{E_{photon}}\right)^3}{(2.5)^2 E^{0.39}*T^{0.15}}$$

We then define the constants to describe the plasma conditions. We assumed a density of 75 grams/cc, a radius of 30 microns, a bandwidth (time it emits X-rays) of 225 picoseconds, and an optical depth factor of 1. The spectrum is initially in terms of mJ of energy and required a conversion to KeV. The spectrum was then scaled by the fraction of the solid angle that the detector could see. Next, the program imports the required data to simulate the implosion. The transmission data for vanadium, copper, germanium, molybdenum, aluminum, kapton, high-density carbon, and gold were imported. In addition, the response data for the x-ray image plate was also imported. This is required in a calculation the absolute signal on the plate, since it describes how sensitive the image plate is to different x-ray frequencies. Scaling the synthetic spectra by the transmission spectra, the image plate (x-ray image plate) response, the KeV conversion factor, and the solid angle factor, the implosion bremsstrahlung spectra on each image plate was calculated. This spectra was then integrated over the considered energy range (1 Kev to 50 KeV). This integration produces six PSL (photostimulated luminescence) values, one for each filter channel. The PSL is a measure of light given off by the x-ray image plate when read. Unlike

conventional x-ray film, the image plates used in the detector are reusable. Incoming radiation displaces electrons in the crystal lattice. When scanned with a laser, the electron can be stimulated to return to its position in the lattice. This transition emits a photon, which is then detected by the image plate scanner. The amount of stimulated luminescence is proportional to the prior irradiance of the image plate. Thus, given a set of plasma conditions, the program calculates the signal that would be seen behind any given filter.

Using this function, the program then creates a table of PSL values for a range of different plasma temperatures. (Because there is a different x-ray spectrum for each assumed plasma temperature). It repeats this process, creating a second table with greater temperature resolution (using the same range, but more values).

We then define a function which, given a filter configuration and a list of plasma temperatures, returns an array of PSL values at those temperatures and filter configuration. This function is then used to build a second function, which takes a temperature range, noise level, and filter configuration. It returns an array as before, but this time with random "error" added to each of the PSL values at levels set by the noise level input. Once the program was capable of creating synthetic PSL values for any filter configuration, plasma temperature, and noise level, a method to fit a bremsstrahlung spectrum to any PSL set was needed.

Developing Experimental Data:

As described above, the experimental data was generated from the Ross Filter Pair diagnostics recording x-ray flux from NIF implosions. All experimental data was obtained by collaborators of the author. After an experimental run, the image plate was removed from the diagnostic. The image plate was a Fujifilm BAS-SR type medical imaging imaging plate. This plate has 25μm resolution and was scanned on a GE FLA 7000. An image of similar image plates can be seen below (Figure 10).

Figure 10: Image plates like these where used in the Ross Filter Pair diagnostic to record the x-ray signal from NIF implosions.

The image plates were then scanned on a Tycoon FLA 7000 and the PLS values were recorded in a two dimensional plot. Figure eleven shows an image of a scanned image plate, note the multiple exposures behind the filter. Also present in this image is evidence of experimental mispointing. The diagnostic was not in perfect alignment with the target at the time of the implosion. This results in images being recorded on only part of the image plate.

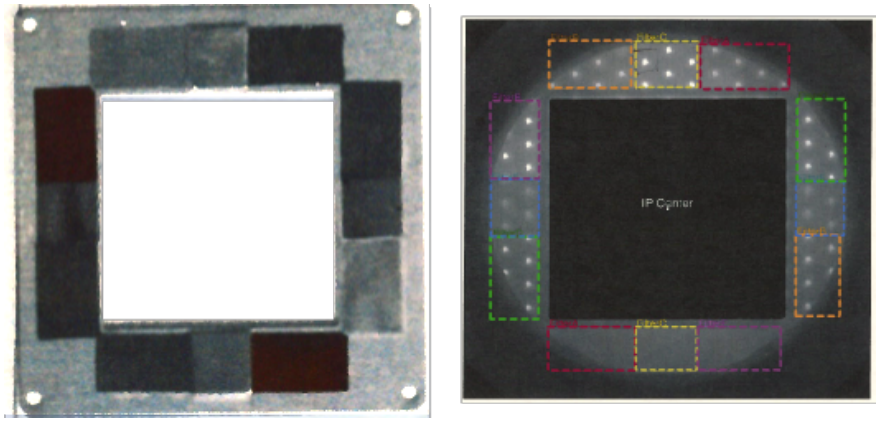
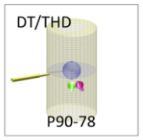


Figure 11. Left) An image of the the filter set used in the Ross Filter Pair Diagnostic. Right) The image recorded on the image plate after an implosion. Filters are repeated around the film to ensure an image is recorded in the event of diagnostic mispointing.

The images behind each filter are extracted and averaged together to produce one composite image for each filter. The shape characteristics and integral brightness was analyzed from each composite. This data was then compiled in a spreadsheet like the one seen in figure twelve.



	Vanadium	Copper	Molybdenum	Germanium	Al (85um)	Al (1000um)
# of Frames	5	5	3	5	3	5
Brightness (PSL)	$8323^{+789}/_{-557}$	$9242^{+289}/_{-636}$	$23020^{+3087}/_{-1515}$	$14351^{+1121}/_{-990}$	$26373^{+1042}/_{-1863}$	$4508^{+550}/_{-408}$
P0 (μm)	$27.10^{+1.54}/_{-0.62}$	$28.14^{+1.13}/_{-1.12}$	$27.64^{+1.89}/_{-1.25}$	$28.25^{+1.85}/_{-1.08}$	$27.57^{+1.38}/_{-0.56}$	$26.98^{+1.72}/_{-1.14}$
P2/P0 (%)	$-18.06^{+4.24}/_{-2.38}$	$-17.37^{+1.63}/_{-1.98}$	$-15.89^{+1.19}/_{-0.86}$	$-13.13^{+1.65}/_{-2.28}$	$-14.12^{+2.04}/_{-1.88}$	$-20.32^{+1.41}/_{-3.56}$
P3/P0 (%)	$-3.52^{+4.84}/_{-3.97}$	$-1.88^{+1.55}/_{-1.77}$	$-2.33^{+3.09}/_{-1.32}$	$-1.93^{+1.94}/_{-1.19}$	$-0.41^{+0.21}/_{-2.20}$	$-2.52^{+3.55}/_{-1.29}$
P4/P0 (%)	$3.03^{+4.48}/_{-2.71}$	$4.70^{+1.17}/_{-3.63}$	$1.42^{+4.84}/_{-1.31}$	$3.66^{+0.56}/_{-3.53}$	$3.08^{+3.08}/_{-3.76}$	$4.10^{+2.73}/_{-3.99}$

Figure 12. Date from the Ross Filter Pair diagnostic was compiled in a spreadsheet. The images vary slightly between filters. The integral PSL value of each composite image is calculated including error estimates. Note the number of images that each composite image is made of.

Additional data about each implosion was recorded with other detectors. Later analysis used the implosion's ion temperature, polar-equatorial average electron temperature, DT and DD neutron yields, maximum fuel velocity (speed of the implosion), X-ray burn width time

(approximately, the amount of time the core emitted X-rays), laser power, laser energy, polar and equatorial electron temperature . A collection of these data points were compiled in a different spreadsheet, recording data for each implosion.

There are two different electron temperatures. These two measurements differ by the orientation of the Ross Filter Pair diagnostic at the time of data acquisition. The polar electron temperature is the measurement of the electron temperature inferred from data from the detector when it is pointed at one of the poles of the target. The equatorial electron temperature is inferred from data from the detector when it is pointed at the equator of the target. These two differ in the material that the X-rays must pass through. From the polar view, X-rays pass through the ablative material of the target and continue unobstructed until they reach the detector. From the equatorial view, X-rays pass through an additional layer of gold and high density carbon. The difference in signal is simple to account for in calculations.

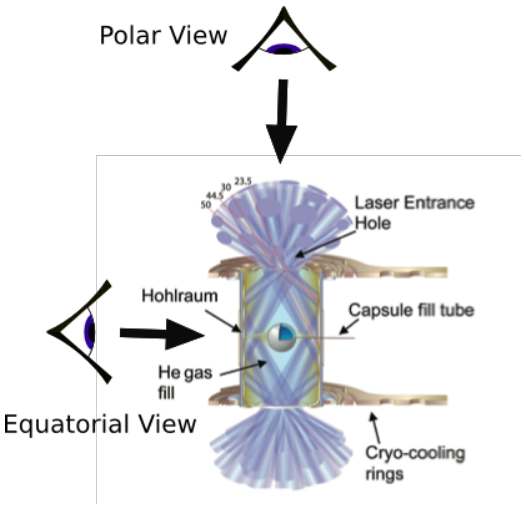


Figure 13: Equatorial electron temperatures are inferred from data produced when the Ross Filter Pair diagnostic has an equatorial view of the implosion. Polar electron temperatures are inferred from data produced when the Ross Filter Pair diagnostic has an polar view of the implosion.

DATA ANALYSIS:

Fitting A Bremsstrahlung Spectrum:

To do this, an ensemble of possible PSL values was calculated. An ensemble of at least 1000 was used. Each of the members has a random error. This ensemble of PSL values was then processed through the program. For each member, the set of PSL values was compared to a large list of synthetic error free PSL values produced from a wide range of temperatures. The program then calculated the root mean square error of the difference between the PSL values in the ensemble and the ideal list. The entry that produced the smallest root mean square error was selected and that entry's generating plasma temperature was taken as the inferred electron temperature of that member of the ensemble. This process was repeated for each member of the ensemble. The program then took minimum and maximum values of the inferred temperatures, and assuming they were mostly correct, used them for the second stage of the fitting process.

After an initial estimate of the temperature was made, the process was repeated, but at a higher resolution. The maximum and minimum values were used from the first stage to create a small window in which to search inside of for the high-resolution analysis. The second stage used a table of ideal PSL values using smaller temperature steps. This two-stage approach was employed to efficiently fit the bremsstrahlung spectra to the data.

Analysis of the Sensitivity to Filter Configuration:

Once the second fitting routine was completed, a list of inferred electron temperatures was returned. The standard deviation of these temperatures was then computed. This process was repeated for pairs of filters to compare the standard deviation of the inferred electron temperatures. It was found that there was no set of filters, which produced a significantly inferior inferred electron temperature.

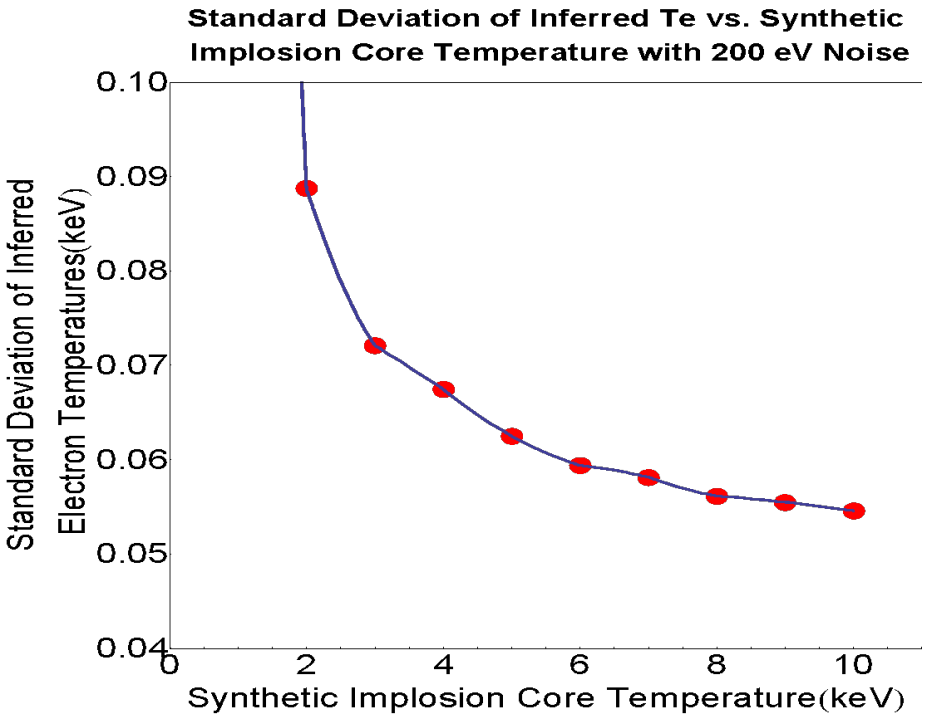


Figure 14: To verify the correct function of the program, synthetic implosion core temperatures were plotted against the standard deviation of the inferred electron temperature. As expected, it is seen that at higher temperatures, the standard deviation decrease. The standard deviation decreases because the signal to noise ratio increases with increased temperature. The hotter the plasma, the greater the number of X-rays that are incident on the image plate.

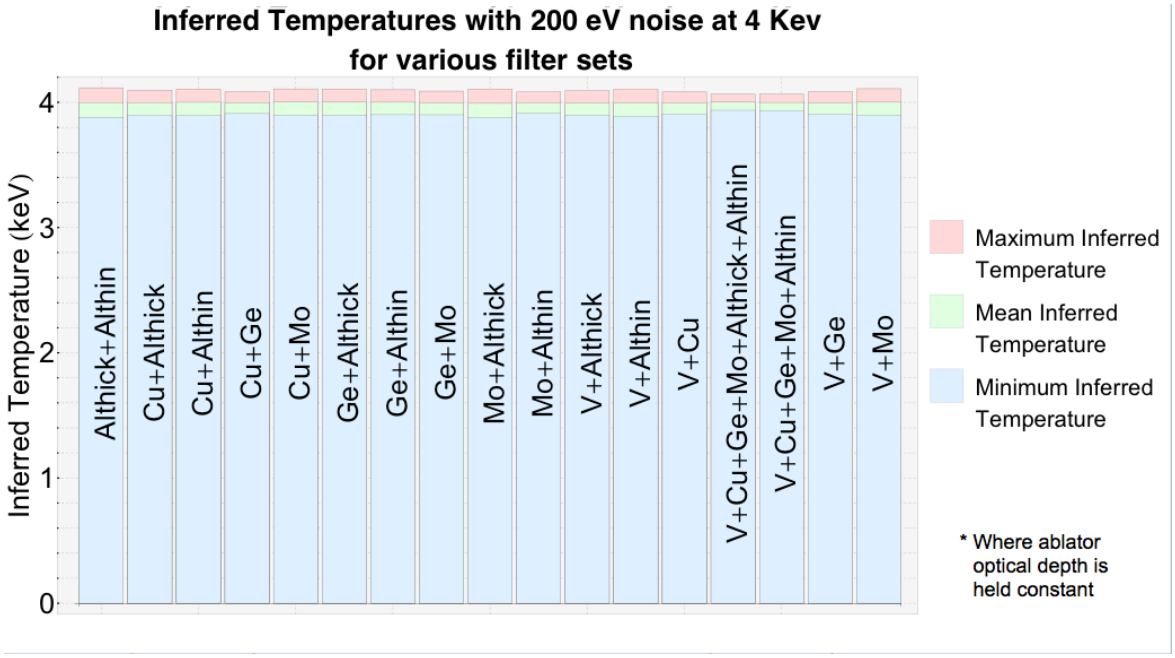


Figure 15: The known systematic error in the diagnostic is on the order of 200 eV. Applying this same value to the synthetic data, and then employing a monte-carlo method program to assess the subsequent error in the inferred temperature, it is demonstrated that all filter configurations yield nearly identical temperature values.

Application of the Code to Experimental Implosion Data:

After comparing the errors of the inferred electron temperature between the different filter configurations, the methods used to compute these values were then applied to experimental data. Implosions from April 2013 to August 2014 were analyzed, totaling fourteen in number. For each of these implosions, inferred polar and equatorial electron temperatures were calculated with associated errors. The errors were computed from the calculations with the synthetic data. These values were then added to an existing dataset which had values for each implosion's ion temperature, polar-equatorial average electron temperature, DT and DD neutron yields, maximum fuel velocity (speed of the implosion), X-ray burn width time (approximately, the amount of time

the core emitted X-rays), laser power, and laser energy. A Mathematica script was then written to harvest and plot the information from the dataset for any set of datapoints.

First the polar and equatorial inferred electron temperatures were plotted against each other. As can be seen in Fig. 16, both temperatures are very similar. While in principle they should be identical, they are not.

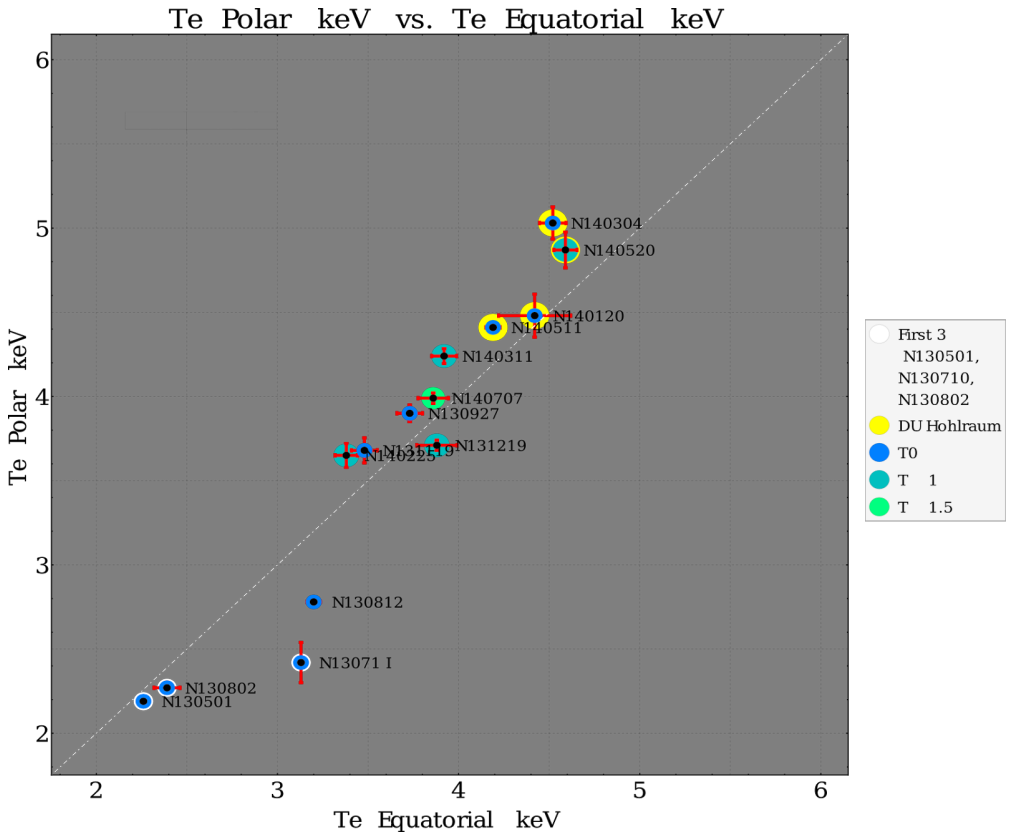


Figure 16: When plotting the polar inferred electron temperature versus the equatorial inferred electron temperature, it is evident that both measurements are not identical. This is likely due to asymmetries in the implosions, were more target material obscures the view of the core in one direction than the other.

Differences in the two inferred temperatures is likely due to the asymmetry of the target at the time of the implosion. The asymmetry could cause the core to be more shielded from view in one view than the other. In the case of implosion N130710 it is clear from the composite images from the

polar and equatorial views that the core of the implosion appeared quite differently to the detectors positioned with each of these views.

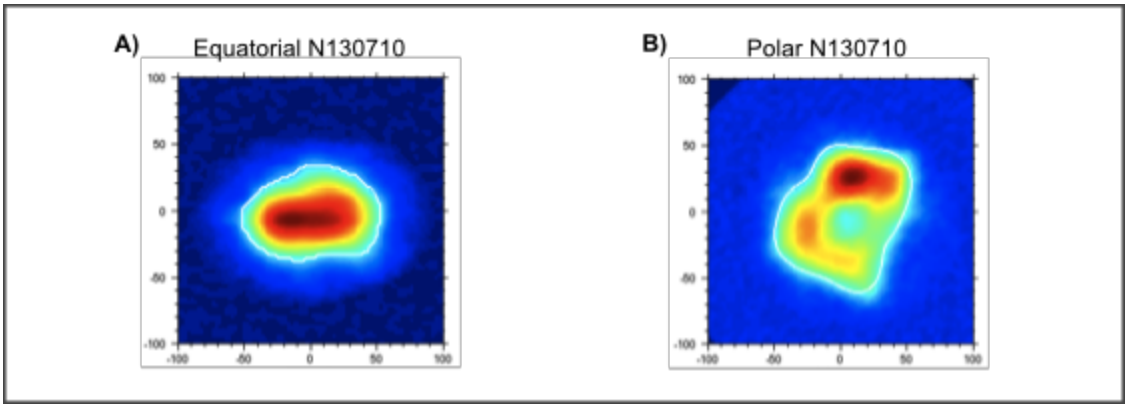


Figure 17: Polar and equatorial views show different images. For N120710 the polar electron temperature is significantly less than the equatorial temperature. It is thought that image B shows not a "donut" shaped temperature profile, but rather a uniform temperature distribution with the center obscured by residual cold fuel. By shielding the center from Ross Filter Pair diagnostic a "cooler" electron temperature will be inferred.

It is seen, except for a few exceptions, that polar inferred electron temperatures match equatorial temperatures well.

The next trend to be examined was the electron temperature versus the ion temperature. As discussed above, prior work had discovered that measurements of the electron and the ion temperatures yielded different results, with measurements of the ion temperature yielding higher values. This discrepancy was investigated using the inferred temperatures from this research as well as the errors. As seen in Fig. 18, this discrepancy still exists and cannot be accounted for by error in the inferred electron temperature.

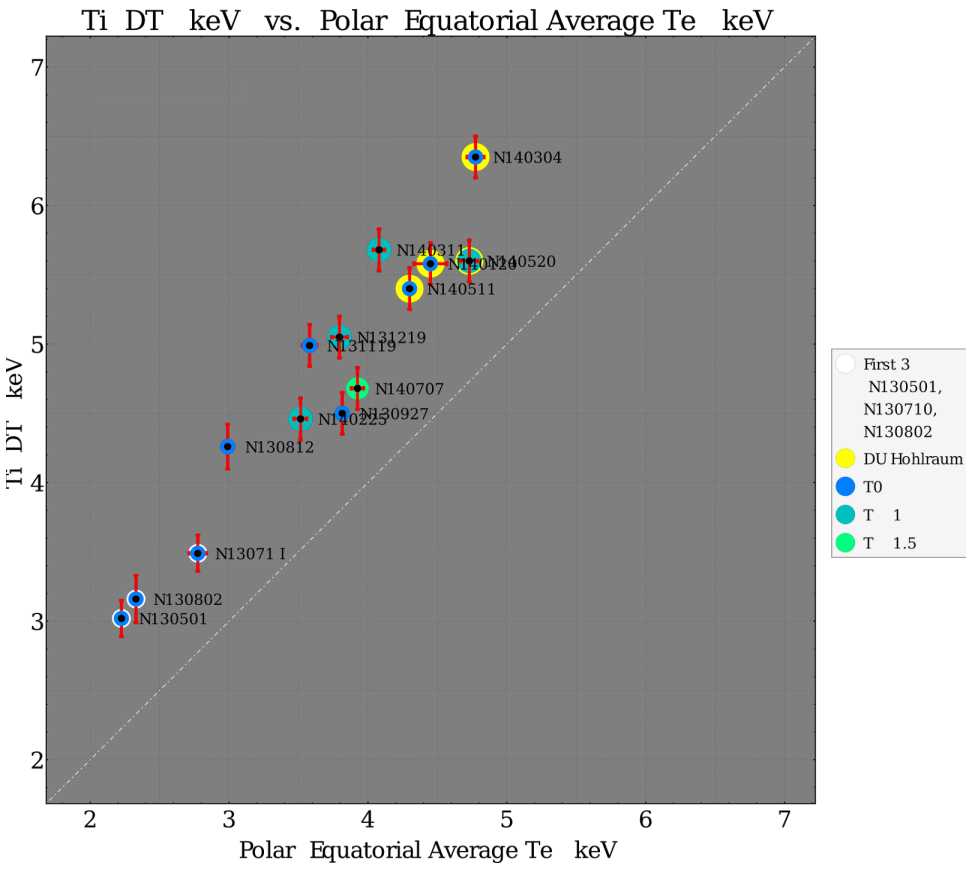


Figure 18: Plotting the implosion core ion temperature versus the polar equatorial average electron temperature, the ion temperature is shown to be characteristically higher. This may be due to residual motion of the core.

The difference in the electron and ion temperatures may be due to residual motion in the core. Under ideal conditions, the target is uniformly compressed on all sides in a spherical implosion, where all external forces balance out and the target stays stationary. If this does not occur, part of the laser energy can be lost into displacing the target within the target chamber. Consequently, this motion could skew the ion temperature.

The next trend to be examined was DT neutron yield versus inferred electron temperature. The relationship of neutron yield to temperature was one of the metrics examined in the planning of the

ICF program. Computer simulations predicted a power law scaling of $Y=T^{4.7}$ for the DT yield as a function of ion temperature. When the inferred electron temperature was plotted against the DT yield it was found that this relationship did not hold. Instead, as seen in figure 19, a power scaling of $Y=T^{2.64}$ was found. The exact cause for this discrepancy is not well known.

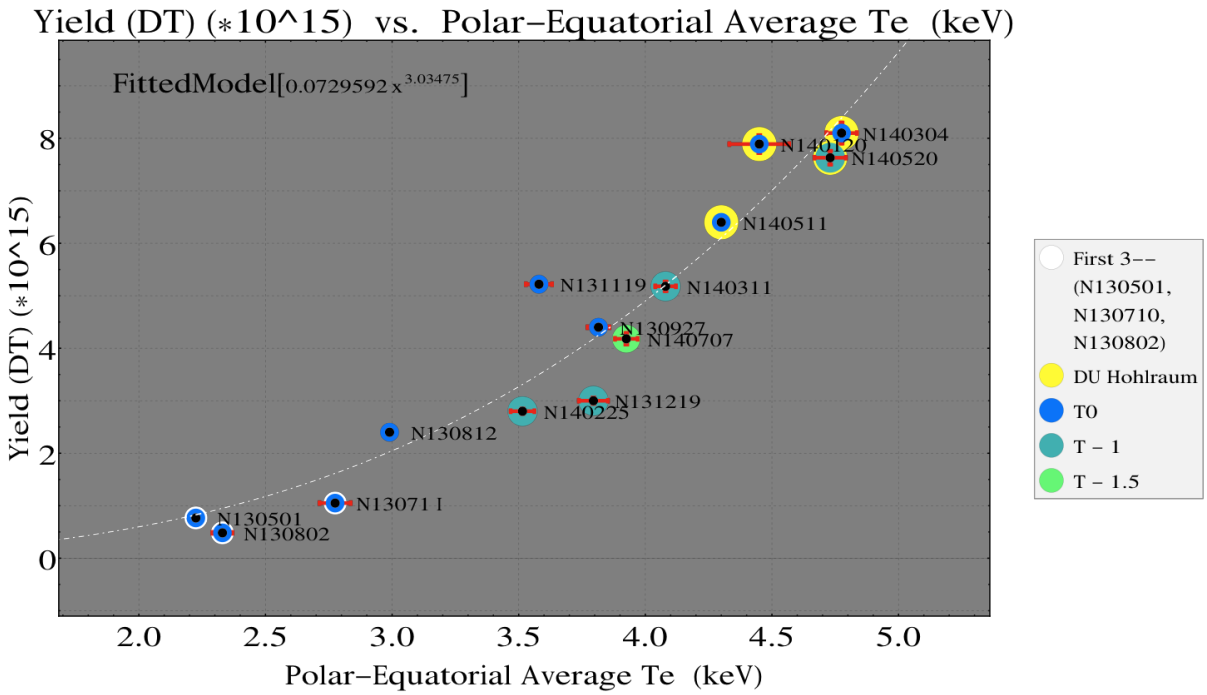


Figure 19: Plotting DT neutron yield versus the polar-equatorial average inferred electron temperature reveals that yield scales as $Y=T^{2.64}$. This is much more gradual that the predicted scaling of $Y=T^{4.7}$.

CONCLUSION:

The success of inertial confinement fusion would have a profound effect on american energy independence and the long term health and safety of the environment. Unfortunately, the problems to be solved before such technology can be developed are immense. The approach at the Lawrence Livermore National Laboratory has had much success in comparison to previous efforts in the field to reach ignition. There is still much to be learned about the particulars of inertial confinement fusion before ignition will take place. It an effort to understand how close the NIF is to achieving ignition, a robust measurement of the temperature of the core is needed. While the neutron time of flight method is currently used, data from these measurements seems to conflict with inferred electron temperatures. To understand the significance of this disagreement, the errors in the inferred electron temperature must be fully understood. The systematic uncertainty of the the measurements due to the construction of the detector had been previously investigated, but the significance of the choice of filters used in the diagnostic had never been probed. After using a monte carlo method to investigate the sensitivity of the inferred electron temperature on filter choice, it was found that no one filter impaired the accuracy of the inferred electron temperature. The method used to calculate error in the first simulation was then applied to experimental NIF data for fourteen implosions. Having done this, trends were then examined in the data. It was observed that for “well performing” implosions, the equatorial and polar inferred electron temperatures matched well. After plotting ion temperature versus electron temperature, the difference in the two was maintained and it could not be accounted for by the error in the inferred electron temperature. Lastly, plotting the yield of the implosions versus the electron temperature, the yield was found not to scale as strongly with temperature as predicted by simulations.

REFERENCES:

[1] A. Fitzpatrick, Igniting The Light Elements: The Los Alamos Thermonuclear Weapon Project, 1942–1952, Ph.D., Virginia Polytechnic Institute and State University, 1999.
[2] S. Hansen, Physics Of Plasmas 19, (2012).
[3] S. Atzeni and J. Meyer-ter-Vehn, *The Physics Of Inertial Fusion* (Oxford University Press, Oxford, U.K., 2009).
[4] B. Maddox, H. Park, B. Remington, N. Izumi, S. Chen, C. Chen, G. Kimminau, Z. Ali, M. Haugh and Q. Ma, Rev. Sci. Instrum. 82, (2011).
[5] J. Lindl, P. Amendt, R. Berger, S. Glendinning, S. Glenzer, S. Haan, R. Kauffman, O. Landen and L. Suter, Physics Of Plasmas 11, (2004).

APPENDICES:

Sample of Project Code Written in Mathematica:

Calculates the inferred polar electron temperatures assuming an uniform error level in PSL values.

Bremsstrahlung X-Ray Time Integrated PSL Value Calculator and Plasma Temperature Calculator

```
(*NIF Implosion Bremsstrahlung X-
 Ray Signal Through Ross Filter Pair Diagnostic Simulator

   ********USES STANDARD ERROR, NOT PERCENTAGE OF SIGNAL LEVEL************

  *********ONLY USE FOR Polar DATA**************
  This is an identical copy of the Polar version,
but Au and HDC have been added to the spectrum scalars

  *********DOES NOT INCLUDE KAPTON*****************
 (Note: swap names of kapton transmission file and a thick aluminum filer and
    exclude thin aluminium filter from inputed set to get around this problem)

 *This Program was written by Summer Student Michael Klem,
Summer (2014) as the backbone of his summer project to look at te sensitivity of
 infered electron temperatures of the core of NIF ICF implosions to choise of
 filters in the ross pair diagnostic. This program implements a Monte Carlo
 method to infer a error (standard deviation) in inferred electron temperatures.

 *Last Edit Date       Aug 21, 2014


*)

(*
*General Notes:

  *To indicate filter configuration this program
   used bionary indicators (1 or 0) to indicate the presence of a
   particular filter. The number 1 indicated that a filter is in a set,
the number 0 indicated that it has been excluded.
 *The filters are alway listed in the following order. Vanadium,
Copper, Germanium, Molybdenum, Althick (1000um),Althin (85um).

 *You must save this file to disk and
 create a file named /Users/klem3/Desktop/Simulation
 Data/ (obviously changing klem3 to whatever is appropriate) before the
 exportation of files will function properly. This is because all exported
 files expect to see the above path. You must same this notebook to disk,
because the program saves a copy of itself in each folder holding exported data. If
 the notebook is not saved to disk. The program will have no file to make a copy of.
*)
```

Part 1 : Bremsstrahlung Spectrum Equation

```
(* The X-Ray emmission spectra is a function of Temperture(y) and x-
 ray energy (x) *)


(*The implosion core plasma conditions are entered below*)
Density = 75; (*(g/cc)*)
Radius  = 30; (*(um)*)
Burnwidth = 120;(*(ps)*)
OpticalDepth = 1; (*at 10.85 keV *)
(*The bremsstrahlung spectrum equation from Prav's powerpoint*)
XRaySpectrum[Temp_, PhotonEnergy_] := 0.0001 * 2.8 * Density^2 / 2.5^2 *
   (5.04 * 10^22) * Exp[-PhotonEnergy / Temp] / PhotonEnergy^0.39 / Temp^0.15 *
   (4 / 3 * 3.14) * (Radius * 0.0001)^3 * (Burnwidth * 0.000000000001) *
   Exp[-OpticalDepth * (10.85 / PhotonEnergy)^3]
```

Part 2 : Initiate Variables and load Files

```
(*Initiate Variables*)

(*The spectrum is in units of mJ so Convert to KeV*)
ConverttoKeV = 6 241 500 000 000.0;
(*The Detector only sees a small portion of the total emitted X-rays,
thus scale the calculation by the detectors solid angle *)
SolidAngle = 0.00000000785398;
(*This is a list of x-ray energies for which the spectrum is defined*)
xvalImport = Import["/Users/klem3/Desktop/Files To Run Michael's Code/xval.csv"];

(*Import the filter transmission data for all the filters*)
vdata =
  Import["/Users/klem3/Desktop/Files To Run Michael's Code/VTransmission.csv"];
Cuboiddata = Import[
   "/Users/klem3/Desktop/Files To Run Michael's Code/CuTransmission.csv"];
Gedata = Import["/Users/klem3/Desktop/Files To Run
     Michael's Code/GeTransmission.csv"];
Modata = Import["/Users/klem3/Desktop/Files To Run Michael's
     Code/MoTransmission.csv"];
Althickdata = Import["/Users/klem3/Desktop/Files To Run
     Michael's Code/AlthickTransmission.csv"];
Althindata = Import["/Users/klem3/Desktop/Files To Run
     Michael's Code/AlthinTransmission.csv"];
Kapdata = Import["/Users/klem3/Desktop/Files To Run Michael's
     Code/KapTransmission.csv"];
HDCdata = Import["/Users/klem3/Desktop/Files To Run Michael's Code/HDC160um.dat"];
Golddata =
  Import["/Users/klem3/Desktop/Files To Run Michael's Code/filterAu2.25um.dat"];
(*Import the image plate response data*)
IPresponseData =
  Import["/Users/klem3/Desktop/Files To Run Michael's Code/IPdata.csv"];
```

Part 3 : Format Imported Data

```
(*Flatten the newly created lists to get rid of unwanted structure from import*)

xval = Flatten[xvalImport];
VTransmission = Flatten[vdata];
CuTransmission = Flatten[Cuboiddata];
GeTransmission = Flatten[Gedata];
MoTransmission = Flatten[Modata];
AlthickTransmission = Flatten[Althickdata];
AlthinTransmission = Flatten[Althindata];
KapTransmission = Flatten[Kapdata];
GoldTransmission = Golddata[[All, 2]];
KapTransmission = Flatten[Kapdata];
HDCTransmission = HDCdata[[All, 2]];
```

Part 4 : Calculate Spectrum Scalars

```
(*Create a unique spectrum scalar for each filter. It
 accounts for the transmittence of the filter metal,
transmittence of Kapton of 1225 um thickness, the IP response,
and also includes the unit conversion from mJ to KeV and the
 solid angle scaling to get expected values at the image plates*)
VSpectrumMult = VTransmission * KapTransmission *
    IPresponseData * ConverttoKeV * SolidAngle;
CuSpectrumMult = CuTransmission * KapTransmission *
    IPresponseData * ConverttoKeV * SolidAngle;
GeSpectrumMult = GeTransmission * KapTransmission *
    IPresponseData * ConverttoKeV * SolidAngle;
MoSpectrumMult = MoTransmission * KapTransmission *
    IPresponseData * ConverttoKeV * SolidAngle;
AlthickSpectrumMult = AlthickTransmission * KapTransmission *
    IPresponseData * ConverttoKeV * SolidAngle;
AlthinSpectrumMult = AlthinTransmission * KapTransmission *
    IPresponseData * ConverttoKeV * SolidAngle;
```

Part 4 : Define A Function to Solve For All PSL Values at a Certain Temperture

```
(*Now create a function to give expected time
 integrated PSL through each of the Ross Pair Filters*)
PSLvalues[y_] := (
  (*
  (*Create a list of the spectrum values at all the point in xval*)
  XrayEmissionValues=XrayEmissionfn[xval,y];
  *)


  XrayEmissionValues = XRaySpectrum[y, xval];


  (*Produce a spectrum of PSL values in terms
   of KeV. Multiply the spectrum by the Spectrum Scalar*)
  VSpectrum = VSpectrumMult * XrayEmissionValues;
  CuSpectrum = CuSpectrumMult * XrayEmissionValues;
  GeSpectrum = GeSpectrumMult * XrayEmissionValues;
  MoSpectrum = MoSpectrumMult * XrayEmissionValues;
  AlthickSpectrum = AlthickSpectrumMult * XrayEmissionValues;
  AlthinSpectrum = AlthinSpectrumMult * XrayEmissionValues;

  (*Create lists of ordered pairs of the xval values and spectrum values*)
  VIntegrateData = Transpose[{xval, Flatten[VSpectrum]}];
  CuIntegrateData = Transpose[{xval, Flatten[CuSpectrum]}];
  GeIntegrateData = Transpose[{xval, Flatten[GeSpectrum]}];
  MoIntegrateData = Transpose[{xval, Flatten[MoSpectrum]}];
  AlthickIntegrateData = Transpose[{xval, Flatten[AlthickSpectrum]}];
  AlthinIntegrateData = Transpose[{xval, Flatten[AlthinSpectrum]}];


  (*For each list,
  interpolate the data and then integrate it across all xval values*)
  VIntegrate = NIntegrate[Interpolation[VIntegrateData, InterpolationOrder → 1][x],
    {x, Min[xval], Max[xval]}];
  CuIntegrate = NIntegrate[Interpolation[CuIntegrateData, InterpolationOrder → 1][
     x], {x, Min[xval], Max[xval]}];
  GeIntegrate = NIntegrate[Interpolation[GeIntegrateData, InterpolationOrder → 1][
     x], {x, Min[xval], Max[xval]}];
  MoIntegrate = NIntegrate[Interpolation[MoIntegrateData, InterpolationOrder → 1][
     x], {x, Min[xval], Max[xval]}];
  AlthickIntegrate = NIntegrate[Interpolation[AlthickIntegrateData,
      InterpolationOrder → 1][x], {x, Min[xval], Max[xval]}];
  AlthinIntegrate = NIntegrate[Interpolation[AlthinIntegrateData,
      InterpolationOrder → 1][x], {x, Min[xval], Max[xval]}];

  (*Create a list to hold all the integrated PSL values through the
   different filters. This list is then returned by the function*)
  IntegratedPSLvalues = {VIntegrate, CuIntegrate, GeIntegrate,
    MoIntegrate, AlthickIntegrate, AlthinIntegrate};
  IntegratedPSLvalues
 )
```

Part 5 : Create A Table of All PSL Values of a Certain Range of Tempertures

```
(*Create a table of all PSL values of a certain range of
 tempertures. This list can then be writen to disk in a wdx file*)

(*Note:Make sure the iterator of this temperture range
  matches all interators of other temperture ranges. If they
  mismatch things that take transposes usually experience problems*)

(*The {x,0.0,0.0,0.0,0.0,0.0} term is to make the list a tensor for compiled
 function reasons. The last five elements of the first term is usually ignored*)



(*
mastertable=
 Import["/Users/klem3/Desktop/Files To Run Michael's Code/datawide0.01.wdx"];
mastertableOOne=
 Import["/Users/klem3/Desktop/Files To Run Michael's Code/datawide0.1.wdx"];
*)



(*Mastertable is a table of all PSL levels throught the six
 possible filters for synthetic plasma temperatures ranging from
 0.01 to 11 keV steping through temperature in 0.01 keV steps*)

mastertable = Table[{{x, 0.0, 0.0, 0.0, 0.0, 0.0}, PSLvalues[x]},
      {x, 0.01, 11, 0.01}]; // AbsoluteTiming;

(*mastertableOOne is a table of all PSL levels throught the six possible
 filters for synthetic plasma temperatures ranging from 0.01 to 11
 keV steping through temperature in 0.1 keV steps. This is a lower
 resolution table used in the first stage of the fitting routine*)

mastertableOOne =
  Table[{{x, 0.0, 0.0, 0.0, 0.0, 0.0}, PSLvalues[x]}, {x, 0.1, 11, 0.1}];

(*This indicates the resolution of mastertable. It
 is used when creating file and directory names*)

Resolution = 0.01;
```

Part 7 : Define A Function to Find PSL Values for a Given Temperture

```
(*This function returns a list of lists. The second level lists will have
 1 to 6 elements depending on the filter set configuration indicated in the
 input. This function takes a list of synthetic plasma temperatures (keV),
the mastertable table, and a filter set configuration. It returns the PSL
 vlaues recored through all of the filters in the inputed configuration,
at each temperature inputed in the temperature values list *)
```

```
compPSLvaluesfinder5only[Temp_, Master_, V_, Cu_, Ge_, Mo_, Althick_, Althin_] :=
 (
  currentlist = {};
  Vvalue = V;
  Cuvalue = Cu;
  Gevalue = Ge;
  Movalue = Mo;
  Althickvalue = Althick;
  Althinvalue = Althin;
  If[Vvalue == 1, AppendTo[currentlist, {1}]];
  If[Cuvalue == 1, AppendTo[currentlist, {2}]];
  If[Gevalue == 1, AppendTo[currentlist, {3}]];
  If[Movalue == 1, AppendTo[currentlist, {4}]];
  If[Althickvalue == 1, AppendTo[currentlist, {5}]];
  If[Althinvalue == 1, AppendTo[currentlist, {6}]];
  currentlist;

  (*StepSize calaculates the step size used to create the mastertable table*)
  StepSize = Rationalize[Differences[Extract[Master[[All, 1]][[All, 1]], {{1}, {2}}]]];

  (*Knowing the step size and that the temperature values begin at 1xstep size,
  This function calcuates the position of the each inputed
   synthetic plasma temperature in the table, and then uses that
   position to find the corresponding PSL values in that list*)

  GetPSLvalues[y_] :=
   Extract[Extract[Master, (Rationalize[y]) / (StepSize[[1]])][[2]], currentlist];


  Map[GetPSLvalues, Temp]

 )
```

Part 8 : Define A Function to Create a List of a Certain Length of Some PSL Values with a Certain Percentage of Noise at a Certain Temperture

```
(*Offby2 returns the difference in PSL values corresponding to two
 temperature. The first is the given synthetic plasma temperature and the
 second is the given synthetic plasma temperature plus a given step width
 in temperature. It returns these in a list whose entries are the change in
 PSL through each filter in the specified filter configuration. The numbers
 produced by this function are used to set the range by whic hPSL values
 are randomly varied. This produces uniform noise through each filter *)
```

```
Offby2[Temp_, OffbyAmount_, V_, Cu_, Ge_, Mo_, Althick_, Althin_] := (
  Tot = Total[{V, Cu, Ge, Mo, Althick, Althin}];
  Table[(

    ((compPSLvaluesfinder5only[{Temp + OffbyAmount},
          mastertable, V, Cu, Ge, Mo, Althick, Althin])[[All, i]][[1]])
     - ((compPSLvaluesfinder5only[{Temp}, mastertable, V, Cu, Ge,
          Mo, Althick, Althin])[[All, i]][[1]])), {i, 1, Tot}]
 )


(*monstertablefast is a function which returns a table of N filter sets
 with added noise. It takes as inputs, a synthetic plasma temperature,
a noise level (ie 0.2 keV) and the filter set configration*)

monstertablefast[Temp_, NoiseLevel_,
  Ntimes_, V_, Cu_, Ge_, Mo_, Althick_, Althin_] := (

  thismuch2 = Offby2[Temp, NoiseLevel, V, Cu, Ge, Mo, Althick, Althin];
  PSLvaluesforRandom = compPSLvaluesfinder5only[
     {Temp}, mastertable, V, Cu, Ge, Mo, Althick, Althin][[1]];

  Do[(
    Rowlength = Length[PSLvaluesforRandom];
    onetempNoisyPSL = {};
    monsterlist = {};

    monsterlist = Reap[Do[(

            Sow[onetempNoisyPSL = Reap[Do[(

                    PSLprenoise = PSLvaluesforRandom[[i]];


                    PSLwithNoise =
                     PSLprenoise +
                      RandomReal[{-thismuch2[[i]], thismuch2[[i]]}];

                    Sow[PSLwithNoise]
                   ), {i, Rowlength}];][[2]][[1]]

           ]
          ), {Ntimes}];][[2]][[1]]

   ), {1}];

  monsterlist)
```

Part 9 : Define A Function to Find a Best Fit Temperture for a List of a Certain Length of Some PSL Values with a Certain Percentage of Noise at a Certain Temperture

```
(*FindTheTemp5bonly is the function which executes the fitting
     routine. It returns an inferred electron temperature given a set
     of PSL values. The function opperates in two parts. The first
     (FindTheTemp5bonly) is a low resolution search for a best fit
     spectrum. The results from this search are then used to set the area
```

```
 of search for the second high resolution part (FindTheTemp5bonly2).*)


(*As mentioned above,
FindTheTemp5bonly2 is called by FindTheTemp5bonly to search for the
 best fit temperature in a narrowly diffined high resolution area*)
FindTheTemp5bonly2[TempRangeToLookOver_, ValuesToFindTempFor_,
  Va_, Cua_, Gea_, Moa_, Althicka_, Althina_] := (

  (*We first define a function which finds the positions
  of minimume RMS values in a list of lists of RMS values.*)
  PositionFinder5[RMSmins_, AllRMSs_] := (
    PositionOfMinRMSsInRMSlistArray[a_] := Position[a[[2]], a[[1]][[1]][[1]];
    Map[PositionOfMinRMSsInRMSlistArray, Transpose[{RMSmins, AllRMSs}]]
   );

  (* "This function creates a list of lists of RMS values. (RMS = Root Mean
       Square). Given a temperature range to look over, a list
       of lists of PSL values, and the the filter configuaration
       of the sets, the function creates a lsit of fit values
       (RMS) for each set of PSL values. This is done by
       comparing one set of PSL values to a table of PSL
       vlaues generated from a inputed range of temperature.
       For each entry in the table, the individual differences
       in PSL behind each filter are found and squared. The
       square root of the avarage of these differneces is
       then recored as an element in the list " RMS "" *)

compRMSvalue5 = Compile[{{Temp, _Real, 1},
    {list2, _Real, 2}, {V}, {Cu}, {Ge}, {Mo}, {Althick}, {Althin}}, (

    PSLsWithTemP = Table[
      Partition[
       Flatten[Transpose[{Partition[Temp, 1], compPSLvaluesfinder5only[
            Temp, mastertable, V, Cu, Ge, Mo, Althick, Althin]}]],
       (Dimensions[compPSLvaluesfinder5only[Temp, mastertable,
            V, Cu, Ge, Mo, Althick, Althin]][[2]] + 1)]
      , {i, 1, Length[list2]}];
    list3 = Transpose[Table[list2, {i, 1, Length[Temp]}]];

    DifferenceInValues =
     list3 - Table[compPSLvaluesfinder5only[Temp, mastertable, V, Cu,
        Ge, Mo, Althick, Althin], {i, 1, Dimensions[list2][[1]]}];

    SquaredDifferenceInValues = (DifferenceInValues) ^2.;
    TotalSquaredDifferenceInValues = Total[SquaredDifferenceInValues, {3, 3}];
    MeanTotalSquaredDifferenceInValues = Total[SquaredDifferenceInValues,
       {3, 3}] / Dimensions[SquaredDifferenceInValues][[1]];
    RMS = Sqrt[MeanTotalSquaredDifferenceInValues];
    RMS
   ), {{RMS, _Real, 2}}];


( * The step size used to create mastertable is measured *)
MastertableStepSize =
  Rationalize[Differences[Extract[mastertable[[All, 1]][[All, 1]], {{1}, {2}}]]];
```

```
(* " This function takes a list of positions in mastertable and
   returns the corresponding temperature values at those positions " *)

theTemp[posision_] := (
   TempFromPositionInMastertable[pos_] :=
    Extract[mastertable, Round[Round[(Round[Min[TempRangeToLookOver]]) *
            (1 / MastertableStepSize[[1]]) + pos]]][[1]][[1]];
   Map[TempFromPositionInMastertable, posision]
  );

(* " The above functions are executed here. Atable of RMS
   values are generte. Then a list of minimum RMS values is found from
   that table. The Position of these minimusm are then found. Lastly the
   corresponding temperatures are found from the positios of these RMS
   values. These temperatures are the inferred electron temperatures " *)

RMSlist = compRMSvalue5[TempRangeToLookOver,
   ValuesToFindTempFor, Va, Cua, Gea, Moa, Althicka, Althina];
MinOfRMSlist = Map[Min, RMSlist];
PositionsOfMinimumRMSvalues = PositionFinder5[MinOfRMSlist, RMSlist];
theTemp[PositionsOfMinimumRMSvalues]
)
```

```
(*This second part is almost a complete duplicate of the above function,
except that it is executed at lower temperature resolution and instead of
 returning a list of inferred temperatures, it calles FindTheTemp5bonly,
and defines a narrow region of tempertures for FindTheTemp5bonly to look
 in. It uses mastertableOOne in place of mastertable. mastertableOOne
 is a lower temeprature resolution version of mastertable*)

FindTheTemp5bonly[TempRangeToLookOver_,
  ValuesToFindTempFor_, Va_, Cua_, Gea_, Moa_, Althicka_, Althina_] := (

   (*We first define a function which finds the positions
   of minimume RMS values in a list of lists of RMS values.*)
  PositionFinder5b[RMSmins_, AllRMSs_] := (
    PositionOfMinRMSsInRMSlistArray[a_] := Position[a[[2]], a[[1]][[1]][[1]];
    Map[PositionOfMinRMSsInRMSlistArray, Transpose[{RMSmins, AllRMSs}]]
   );

  (* "This function creates a list of lists of RMS values. (RMS = Root Mean
       Square). Given a temperature range to look over, a list of
       lists of PSL values, and the the filter configuaration of
       the sets, the function creates a lsit of fit values (RMS)
       for each set of PSL values. This is done by comparing one
       set of PSL values to a table of PSL vlaues generated from a
       inputed range of temperature. For each entry in the table, the
```

```
          individual differences in PSL behind each filter are found and
          squared. The square root of the avarage of these differneces
          is then recored as an element in the list " RMS " " *)

compRMSvalue5bonly = Compile[{{Temp, _Real, 1},
    {list2, _Real, 2}, {V}, {Cu}, {Ge}, {Mo}, {Althick}, {Althin}}, (

    PSLsWithTemP = Table[
      Partition[
       Flatten[Transpose[{Partition[Temp, 1], compPSLvaluesfinder5only[
            Temp, mastertableOOne, V, Cu, Ge, Mo, Althick, Althin]}]],

       (Dimensions[compPSLvaluesfinder5only[Temp,
            mastertableOOne, V, Cu, Ge, Mo, Althick, Althin]][[2]] + 1)]
      , {i, 1, Length[list2]}];

    list3b = Transpose[Table[list2, {i, 1, Length[Temp]}]];
    DifferenceInValues =
     list3b - Table[compPSLvaluesfinder5only[Temp, mastertableOOne, V,
        Cu, Ge, Mo, Althick, Althin], {i, 1, Dimensions[list2][[1]]}];
    SquaredDifferenceInValues = (DifferenceInValues) ^2.;
    TotalSquaredDifferenceInValues = Total[SquaredDifferenceInValues, {3, 3}];
    MeanTotalSquaredDifferenceInValues = Total[SquaredDifferenceInValues, {3, 3}] /
      Dimensions[SquaredDifferenceInValues][[1]];
    RMSb = Sqrt[MeanTotalSquaredDifferenceInValues];
    RMSb
   ), {{RMSb, _Real, 2}}];

TempFromPositionInMastertableOOne[y_] := Extract[mastertableOOne, y];

theTempb[AllPositions_] := (

   Map[TempFromPositionInMastertableOOne, AllPositions]
  );

RMSlistb = compRMSvalue5bonly[TempRangeToLookOver,
   ValuesToFindTempFor, Va, Cua, Gea, Moa, Althicka, Althina];


MinOfRMSlistb = Map[Min, RMSlistb];



PositionsOfMinimumRMSvaluesb = PositionFinder5b[MinOfRMSlistb, RMSlistb];



LowResInferredTemps = theTempb[PositionsOfMinimumRMSvaluesb];



TempSmallEnd = Round[Min[LowResInferredTemps]] - 1;
TempLargeEnd = Round[Max[LowResInferredTemps]] + 1;



rangeFinder[Cone_, Ctwo_] :=
```

```
    If[Cuvaluname == 1, AppendTo[currentlistname, "Cu"]];
    If[Gevaluename == 1, AppendTo[currentlistname, "Ge"]];
    If[Movaluname == 1, AppendTo[currentlistname, "Mo"]];
    If[Althickvaluename == 1, AppendTo[currentlistname, "Althick"]];
    If[Althinvaluename == 1, AppendTo[currentlistname, "Althin"]];
   ), {1}]; currentlistname);

(*This takes the list of strings and joins them all to one string with "+" signs*)
FilterName = StringDrop[StringJoin[Table[namer[V , Cu, Ge, Mo, Althick, Althin][[i]] <> "+",
    {i, 1, Length[namer[V, Cu , Ge, Mo, Althick, Althin]]}]], -1];
(*Creates the folder name where all data is stored from a run*)
DirectoryFileName =
 ("/Users/klem3/Desktop/Simulation Data/" <> DateString[{ "Hour12Short", " ", "Minute",
     "AMPM", " ", "DayName", " ", "Month", ":", "Day", ":", "YearShort"}] <> "__" <>
   ToString[Resolution] <> "Res" <> ToString[Trials] <> "Trials" <> "__" <> FilterName);
(*Creates the folder name where all the numerical data is
 stored. It is placed inside the first folder*)
DirectoryFileNameDataFiles = DirectoryFileName <> "/" <> "/DataFiles";
(*Creates a string with all the
 information about a run for use in file names later on*)
FileInfo = ToString[Resolution] <> "Res" <> ToString[Trials] <>
   "Trials" <> " " <> DateString[
    { "Hour12Short", " ", "Minute", "AMPM", " ", "Month", ":", "Day", ":", "YearShort"}];
(*Creates the above stated directories*)
CreateDirectory[DirectoryFileName];
CreateDirectory[DirectoryFileNameDataFiles];
(*Finds path to current notebook file*)
DirectroyOfThisNoteBook = NotebookFileName[];
(*Creates name of copy of notebook*)
FileNameOfThisNoteBook =
 DateString[{ "Hour12Short", " ", "Minute", "AMPM", " ", "Month", ":", "Day", ":",
    "YearShort", " "}] <> StringReplace[NotebookFileName[], NotebookDirectory[] → ""];
(*Adds a copy of current notebook to data file*)
CopyFile[DirectroyOfThisNoteBook,
 DirectoryFileNameDataFiles <> "/" <> FileNameOfThisNoteBook];


(*Create empty list to apend
 standard deviations of inferred Te at one temperature to*)
STDperTemp = {};

(*Repeats the process below for each
 synthetic plasma temperature (1,2,3,4,5,6,7,8,9,10 keV)*)
Do[(
  (*Create empty lists*)
  numbers = {};
  numbersStandard = {};
  AllTemps = {};
  STDFourError = {};
  ListofSTDBars = {};

  (*Repeats calculations for 0.1 keV, 0.2 keV, 0.3 keV, 0.4 keV error *)
  Do[(
    (*Generates all inferred temperatures*)
    FoundTemps =
     FindTheTemp5bonly[Range[0.1, 11, 0.1], monstertablefast[Temp, NoiseLevel,
       Trials, V, Cu, Ge, Mo, Althick, Althin], V, Cu, Ge, Mo, Althick, Althin];


    MaxTemp = Max[FoundTemps];
```

```
  MaxTemp = Min[FoundTemps];
  MaxTempDiff = Max[FoundTemps] - Mean[FoundTemps];
  MinTempDiff = Min[FoundTemps] - Mean[FoundTemps];
  bars = {{NoiseLevel, Mean[FoundTemps]}, ErrorBar[0, {MaxTempDiff, MinTempDiff}]};
  (*Create error bar at each noise level*)
  STDbars =
   {{NoiseLevel, Mean[FoundTemps]}, ErrorBar[0, StandardDeviation[FoundTemps]]};

  (*Add the above variable values to pre-made lists*)
  AppendTo[ListofSTDBars, STDbars];
  AppendTo[numbers, {Max[FoundTemps], Min[FoundTemps]}];
  AppendTo[numbersStandard, {Mean[FoundTemps] - StandardDeviation[FoundTemps],
    Mean[FoundTemps] + StandardDeviation[FoundTemps]}];
  AppendTo[AllTemps, FoundTemps];
  AppendTo[STDFourError, StandardDeviation[FoundTemps]];
 ), {NoiseLevel, 0.1, 0.4, 0.1}];


(*Here, after the numbers have been calculated,
the images are made and the files are exported*)
Scalerr = Max[Flatten[numbers]];
Scalerr2 = Min[Flatten[numbers]];
Differ = Scalerr - Scalerr2;

(*Plot the purple max and min bars*)
BlueLineMinMax = Plot[
  numbers, {x, -1, 21},
  PlotRange → {{-1, 21},
    {Round[Temp - (Differ + 0.15 * Differ)], Round[Temp + (Differ + 0.15 * Differ)]}},
  PlotStyle → {RGBColor[0, 0, 1, 0.001]},
  Filling → {1 → {2}, 3 → {4}, 5 → {6}, 7 → {8}},
  FillingStyle → RGBColor[0, 0, 1, 0.09],
  PlotStyle → Opacity[0]];

(*Create tick marks on large plot*)
ticks[min_, max_] := Table[If[FractionalPart[i] == 0., {i, i, {.01, -.001}, Black},
   {i, "", {.005, -.001}, Black}], {i, Floor[min], Ceiling[max], 0.05}];

ticks2[min_, max_] :=
 Table[If[QuotientRemainder[Round[(i / 2) * 10^((Ceiling[1 / Log[Rationalize[1 / (Round[
                    (2 * Differ / 50), (10.^-Ceiling[1 / Log[Rationalize[1 / (2 * Differ /
                        50)], 10.]])])], 10.]]) - 0), 0.000001], 10][[2]] == 0.,
   {i, i, {.010 , -.001}, Black}, {i, "", {.005, -.001}, Red}],
  {i, Floor[min], Ceiling[max],
   Round[(2 * If[N[Rationalize[Differ]] == 0.25`, 0.2500000000000001, Differ] / 100),
    (10.^-Ceiling[1 / Log[Rationalize[1 / (2 * If[N[Rationalize[Differ]] == 0.25`,
                0.2500000000000001, Differ] / 50)], 10.]])]}];

(*Plot error bars and format large plot*)
StandardBars = ErrorListPlot[
  {ListofSTDBars},
  ErrorBarFunction → Automatic,

  PlotRange → {{0.0, 1.0}, {Round[Temp - (Differ + 0.15 * Differ),
      (10.^-Ceiling[1 / Log[Rationalize[1 / (2 * Differ / 50)], 10.]])],
    Round[Temp + (Differ + 0.15 * Differ), (10.^
       -Ceiling[1 / Log[Rationalize[1 / (2 * Differ / 50)], 10.]])]}},
  PlotStyle → {Thickness[0.004], Red},
```

```
  PlotLegends → SwatchLegend[{ToString[Temp] <> "  KeV"}, LegendMarkerSize → {50, 7}];
  PlotLabel → Style["Calculated Temperture of " <> ToString[Temp] <>
      " KeV Initial Temp for " <> ToString[Trials] <> " Trial", FontSize → 20],
  Frame → True,
  FrameLabel → {"Noise Level (keV)", "Calculated Temperture"},
  FrameStyle → Thickness[0.001],
  BaseStyle → {FontSize → 18}, ImageSize → Scaled[1], PlotMarkers → None,
  GridLines → Automatic,
  GridLinesStyle → Directive[AbsoluteThickness[1], RGBColor[0, 0, 0, 0.1], Dashed],
  FrameTicks → {ticks, ticks2, False, ticks2},
  FrameTicksStyle → Directive → FontSize → 10,
  Prolog → {{RGBColor[0, 0, 0, 0.04], Rectangle[Scaled[{0, 0}], Scaled[{1, 1}]]}},
  Epilog → ({Red, PointSize@.01, Point@ListofSTDBars[[All, 1]]})];

(*Create black lines that connect to error bars*)
ssws = Transpose[{Flatten[{{0.1, 0.1}, {0.2, 0.2}, {0.3, 0.3}, {0.4, 0.4}}],
   Flatten[numbersStandard]}];
sses = Transpose[{Flatten[{{21, 21}, {21, 21}, {21, 21}, {21, 21}}],
   Flatten[numbersStandard]}];
swld = Transpose[{ssws, sses}];
doxs = ListLinePlot[
  swld,
  PlotRange → {{-1, 21},
    {Round[Temp - (Differ + 0.15 * Differ)], Round[Temp + (Differ + 0.15 * Differ)]}},
  PlotStyle → {RGBColor[0, 0, 0, 1]},
  PlotStyle → Opacity[0]
 ];
(*Combine error bars, max/min bars, and black connecting lines into one plot*)
Graphx = Show[StandardBars, BlueLineMinMax, doxs, ImageSize → Scaled[1.1]];


(*Create Histograms of Distributions if inferred Te*)
hist1 = Histogram[AllTemps[[1]], Length[AllTemps[[1]]],
  ImageSize → Scaled[.22],
  PlotLabel → Style["Temperture Distrubution for 0.1 keV \n noise and " <>
     ToString[Trials] <> " Trial", FontSize → 14],
  Frame → True,
  BaseStyle → {FontSize → 14},
  FrameLabel → {"Temperture", "Counts"}];

hist2 = Histogram[AllTemps[[1]], Length[AllTemps[[1]]],
  ImageSize → Scaled[.22],
  PlotLabel → Style["Temperture Distrubution for 0.2 keV \n noise and " <>
     ToString[Trials] <> " Trial", FontSize → 14],
  Frame → True,
  BaseStyle → {FontSize → 14},
  FrameLabel → {"Temperture", "Counts"}];

hist3 = Histogram[AllTemps[[1]], Length[AllTemps[[1]]],
  ImageSize → Scaled[.22],
  PlotLabel → Style["Temperture Distrubution for 0.3 keV \n noise and " <>
     ToString[Trials] <> " Trial", FontSize → 14],
  Frame → True,
  BaseStyle → {FontSize → 14},
  FrameLabel → {"Temperture", "Counts"}];

hist4 = Histogram[AllTemps[[1]], Length[AllTemps[[1]]],
  ImageSize → Scaled[.22],
  PlotLabel → Style["Temperture Distrubution for 0.4 keV \n noise and " <>
     ToString[Trials] <> " Trial", FontSize → 14],
```

```
    Frame → True,
    BaseStyle → {FontSize → 14},
    FrameLabel → {"Temperture", "Counts"}];

  (*Combine histograms and large plot to one image*)
  dots = Graphics[{
     Inset[Graphx, {1.305, 3.45}, Center, {2.6, 2.6}],
     Inset[hist1, {0.435, 2.25}, Center, {0.6, 0.6}],
     Inset[hist2, {1.08, 2.25}, Center, {0.6, 0.6}],
     Inset[hist3, {1.685, 2.25}, Center, {0.6, 0.6}],
     Inset[hist4, {2.315, 2.25}, Center, {0.6, 0.6}]

    }, PlotRange → {{0, 2.75}, {2, 4.35}}, ImageSize → 1200];



  (*Export final image in 4 file
   formats and all the date usedto make images to DataFiles*)
  Export[DirectoryFileName <> "/" <> FilterName <> "__" <> FileInfo <>
    "__" <> "Temp-" <> ToString[Temp] <> ".pdf", dots, "PDF"];
  Export[DirectoryFileName <> "/" <> FilterName <> "__" <> FileInfo <>
    "__" <> "Temp-" <> ToString[Temp] <> ".gif", dots, "GIF"];
  Export[DirectoryFileName <> "/" <> FilterName <> "__" <> FileInfo <>
    "__" <> "Temp-" <> ToString[Temp] <> ".png", dots, "PNG"];
  Export[DirectoryFileName <> "/" <> FilterName <> "__" <> FileInfo <>
    "__" <> "Temp-" <> ToString[Temp] <> ".jpeg", dots, "JPEG"];
  Export[DirectoryFileNameDataFiles <> "/" <> FilterName <> "__" <> FileInfo <>
    "__" <> "Temp-" <> ToString[Temp] <> ".wdx", AllTemps, "WDX"];

  AppendTo[STDperTemp, STDFourError]

 ), {Temp, 1, 10}];



(*Create Plots of standard deviation
 versus temperatue for each error percent level*)
five = ListPlot[Transpose[{Table[x, {x, 1, 10}], STDperTemp[[All, 1]]}],
  Joined → True, PlotStyle → Red,
  PlotLegends → SwatchLegend[{"5%"}, LegendMarkerSize → {50, 7}]];

ten = ListPlot[Transpose[{Table[x, {x, 1, 10}], STDperTemp[[All, 2]]}], Joined → True,
  PlotStyle → Blue, PlotLegends → SwatchLegend[{"10%"}, LegendMarkerSize → {50, 7}]];

fifteen = ListPlot[Transpose[{Table[x, {x, 1, 10}], STDperTemp[[All, 3]]}], Joined → True,
  PlotStyle → Green, PlotLegends → SwatchLegend[{"15%"}, LegendMarkerSize → {50, 7}]];

twenty = ListPlot[Transpose[{Table[x, {x, 1, 10}], STDperTemp[[All, 4]]}], Joined → True,
  PlotStyle → Purple, PlotLegends → SwatchLegend[{"20%"}, LegendMarkerSize → {50, 7}],
  PlotLabel → Style["Standard deviation vs Temperture", FontSize → 16],
  Frame → True, FrameLabel → {"Temperture", "Standard Deviation"},
  FrameStyle → Thickness[0.002], BaseStyle → {FontSize → 16}];
(*Combine all plots*)
ErrorVStemp = Show[twenty, fifteen, ten, five, ImageSize → 400];

(*Export Standard deviation vs. Temerature plot as well as data*)
Export[DirectoryFileName <> "/" <> FilterName <> "__" <>
  FileInfo <> "__(STD-mean)VS Temp__" <> ".pdf", ErrorVStemp, "PDF"];
Export[DirectoryFileName <> "/" <> FilterName <> "__" <> FileInfo <>
  "__(STD-mean)VS Temp__" <> ".gif", ErrorVStemp, "GIF"];
```

```
  Export[DirectoryFileName <> "/" <> FilterName <> "__" <> FileInfo <>
    "__(STD-mean)VS Temp__" <> ".png", ErrorVStemp, "PNG"];
  Export[DirectoryFileName <> "/" <> FilterName <> "__" <> FileInfo <>
    "__(STD-mean)VS Temp__" <> ".jpeg", ErrorVStemp, "JPEG"];
  Export[DirectoryFileNameDataFiles <> "/" <> FilterName <> "__" <>
    FileInfo <> "__(STD-mean)VS Temp__" <> ".wdx", STDperTemp, "WDX"];


 )
```

Find Inferred Temperature

```
(*Use this function if you already have PSL values
     you want to inferrer a electron temperature from. ***Made sure to
   set the BWXray correctly at the vary beginning of the code.******DOES
 NOT WORK FOR Equatorial DATA.********DOES NOT WORK FOR
 FILTER SET WITH KAPTON,
UNLESS THE ABOVE WORKAROUND IS IMPLEMENTED****)

(*To input data--Enter the PSL values as a ***matrix***
 (in the form {{},{}...}, it can be a one dimentional matrix ie ({{}}))
 for the PSLset_ argument. They must be in the order (Vanadium, Copper,
  Germanium, Molybdenum, Althick (1000um),Althin (85um). Replace the V_,
  Cu_,Ge_,Mo_,Althick_,Althin_ arguemts individualy with 1's or 0's to to
   tell the program if each filter is presentin the set. Enter 1 if it is,
  enter 0 if it is not. The numner of 1's must equal thenumer of PSL
   values in the list entered at the PSLset_ argument. This program is
   designed to allow the user to select anywhere from 2 to 6 filters to
   inferrer and electron temperature form. By entering the 1's and 0's,
  you tell the program what filers the entered PSL values are recorded after )*)

ErrorPlotteronlyFind[PSLset_, V_, Cu_, Ge_, Mo_, Althick_, Althin_] := (
  FindTheTemp5bonly[Range[0.1, 11, 0.1], PSLset, V, Cu, Ge, Mo, Althick, Althin]
 )
```

Run Program

```
ErrorPlotteronlyGenerate[1000, 1, 1, 1, 1, 0, 1];
```

```
$Aborted
```

```
ErrorPlotteronlyFind[{{1, 1, 1, 1, 1, 1}}, 1, 1, 1, 1, 1, 1]
```

```
{0.84}
```