4-15-2011

Cross Calibration of Hinode's X-ray Telescope and Extreme Unltraviolet Imaging Spectrometer Using X-ray Bright Points

Jason Andrew Kimble

To the University Council:

The Thesis Committee for Jason Andrew Kimble certifies that this is the final approved version of the following electronic thesis: “Cross Calibration of Hinode's X-Ray Telescope and Extreme Ultraviolet Imaging Spectrometer Using X-Ray Bright Points.”

_____
Joan T. Schmelz, Ph.D.
Major Professor

We have read this thesis and recommend its acceptance:

_____
Donald R. Franceschetti, Ph.D.

_____
Firouzeh Sabri, Ph.D.

Accepted for the Graduate Council:

_____
Karen D. Weddle-West, Ph.D.
Vice Provost for Graduate Programs

CROSS CALIBRATION OF HINODE'S X-RAY TELESCOPE AND EXTREME ULTRAVIOLET IMAGING SPECTROMETER USING X-RAY BRIGHT POINTS

by

Jason A. Kimble

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Major: Physics

The University of Memphis

May, 2011

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr Joan Schmelz for all her guidance, and also my thesis committee, Doctors Firouzeh Sabri and Donald R Franceschetti for their support and understanding. I would also like to express my appreciation to fellow student Jennifer Garst and to my roommate Marc Brown for their efforts in proofreading this paper, as I write too much like I talk.

I should also mention the hard work of the many researchers who created the instruments and masses of both data and software without which a study like this would be completely impossible. Hinode is a Japanese mission developed and launched by ISAS/JAXA, with NAOJ as domestic partner and NASA and STFC (UK) as international partners. It is operated by these agencies in co-operation with ESA and the NSC (Norway). CHIANTI is a collaborative project involving researchers at NRL (USA) RAL (UK), and the Universities of: Cambridge (UK), George Mason (USA), and Florence (Italy). Solar physics research at the University of Memphis is supported by NSF ATM-0402729 as well as a Hinode subcontract from NASA/SAO.

ABSTRACT

Kimble, Jason Andrew. M.S. The University of Memphis. May 2011. Cross Calibration of Hinode's X-Ray Telescope and Extreme Ultraviolet Imaging Spectrometer Using X-Ray Bright Points. Major Professor: Dr. Joan T. Schmelz.

The focus of the present study is to produce a cross calibration factor for Hinode's X-Ray Telescope (XRT) and Extreme Ultraviolet Imaging Spectrometer (EIS) instruments by using observations of X-ray bright points. Material in the center of several bright points will be analyzed using data from EIS, and these results will be applied to concurrent XRT data to generate a relative response ratio between the two instruments. A secondary purpose is to investigate the thermal characteristics of bright points themselves. The differential emission measure curves produced from the EIS observations will themselves be valuable information about these coronal phenomena.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction to the Sun

1.1 Formation of the Sun

In any discussion of a part of the sun, it becomes necessary to put the information in context by developing a basic working knowledge of the sun as a whole. The sun can be described in a number of ways, depending on one's perspective; one can talk about the sun as a celestial body and the center of our solar system,a star, a reactor producing energy, or an object composed of different structures of certain materials undergoing certain processes.

From the perspective of a stellar astronomer, the sun is a main sequence star of spectral class G2. It is considered a typical example of its kind. The main sequence is a classification of a star based on a positive correlation between the luminosity of the star and its temperature as shown on a Hertzprung-Russel diagram(Golub and Pasachoff, *Nearest Star*, 29). This is the largest group of stars. The sun's place on the main sequence, along with its mass are determining factors in current theories on the development and ultimate fate of the sun. The spectral class is determined by a star's spectroscopic characteristics, and indicates an approximate temperature of the star's radiating surface. In this case it indicates the sun's surface is about 5800 K (Seeds, 35).

The sun is not the same as all other stars, however. Many stars are binary or multiples, but the sun is a single star. Further, some stars are at other places on the main sequence, or not on the main sequence at all. Stars that differ much from the sun may have different nuclear processes as their source of energy and may be of very different

temperatures. As such, their internal structures may differ radically, inverting some of the regions existing in the sun, or lacking them altogether.

The sun is a roughly spherical structure of energetic gas. Its mass is 1.989×10^{30} kg and its radius is about 6.960×10^{5} km. Its average luminosity is 3.85×10^{23} kW (Phillips, 277), which means the amount of energy reaching the earth, the solar constant, is 1369 W/ m^2, though this "constant" is only an average and can vary. Its density is about 1400 kg/m^3 and it rotates differentially, about once per 26 days at the equator and about once per 30 days at high latitudes. The sun contains many of the elements that can be found on the earth in measurable quantities. However, from observations, the parts of the sun that can be seen are predominantly hydrogen, which is about 91% of the sun's composition. Helium is the next most common, with a somewhat uncertain measurement of about 9%. Oxygen, carbon and many other heavier elements are found in much smaller amounts, totaling about .1% (Phillips, 101).

Like other stars, the sun contracted from a cloud of gas and interstellar material, heating up from the release of gravitational potential energy to become a protostar. When it began radiating, it became a T Tauri star, the last stage of formation before fusion ignition. In this T Tauri phase, the sun developed a strong wind that swept away the debris left over from the star formation process. After fusion ignition occurred, it probably took about 10 million years for the sun to settle into the main sequence. Stars like the sun tend to remain near the same point on the main sequence of the H-R diagram that they entered, drifting little, until the ending stages of their life-cycle, which are

determined largely by the star's mass. This formation took place about 5 billion years ago, and the sun is about in the middle of its life.

The sun has enough nuclear fuel to continue for about another 5 billion years,when it will become a red giant. When this happens, only the hydrogen in the core will be exhausted; much will still exist outside of this region. The radiating zone will contract and the hydrogen just outside the core will be compressed and heated to the point where it can fuse into more helium. This will in turn heat the core and provide it with more helium. Some of the helium will begin to fuse, but at first only one reaction can take place. Two helium atoms will fuse to become an unstable ^{8}Be atom. When the core temperature rises to about 100 million degrees, a third helium atom will be able to fuse with the ^{8}Be atom before it can decay to create stable carbon (Kutner, 178).

When the sun enters the red giant phase, it will move off of the main sequence onto a branch of the H-R diagram. The sun as a red giant will have a surface temperature of about half that at present, and its outer boundaries will have expanded greatly. The increasing solar wind during the red giant phase will have increased the density of matter in the space surrounding the sun. The sun will expend the new helium fuel in its core fairly quickly, and this will end the red giant phase. It may then begin to pulsate as it collapses under the force of gravity to the point that a burst of helium fusion can propel it back out. This will expel more matter, and this, combined with the increased solar wind from the red giant phase, will create a planetary nebula carrying carbon and heavier elements into interstellar space. When the star has lost enough matter to this process, the

remaining material in the star will become compressed to electron-degenerate matter and will slowly cool as a white dwarf.

1.2 Structure of the Sun

As to the structure of the sun, astronomers can directly observe the solar atmosphere and surface, but below that, the traditional methods of analyzing light directly emitted by the object of observation fails. Since the solar surface is opaque, it is not possible to detect light emitted by the interior. However, using hypothetical models of the structure of the sun, it is possible to deduce something of the sun's interior. Using some indirect methods of measurement, data can be obtained on some of the properties of the regions within the sun which can either support or invalidate these models. The sun's interior, in fact most of its structure, can be characterized into regions that are approximately spherical shells, each differentiated by the properties and processes of that region, as illustrated in figure 1.

The energy source for the sun is nuclear fusion, and this takes place in the core where the pressure and temperature, about 15 million degrees (Golub and Pasachoff, *Nearest Star*, 49), are high enough to maintain such a reaction. This core extends about a tenth of the way to the surface. The main process at this point in the sun's development, according to current understanding, requires four hydrogen atoms and results ultimately in a helium atom and a great deal of energy. Specifically there are three processes that combine to produce this net reaction. First two protons fuse to form a deuterium nucleus (^{2}H), a positron (an anti-electron) and a neutrino. After this reaction, a second reaction

can take place: a deuterium and a hydrogen fuse to form helium-3 (^{3}He) and a gamma ray. After two of this second reaction, two helium-3 nuclei fuse to form a helium-4 (^{4}He) and two protons. The excess positron from the initial reaction will annihilate an electron to produce another gamma ray, and thus contribute to energy output. This process converts hydrogen to helium, and requires hydrogen for fuel. However, it is estimated there is enough hydrogen remaining for this process to continue for about 5 billion years (Phillips, 53).

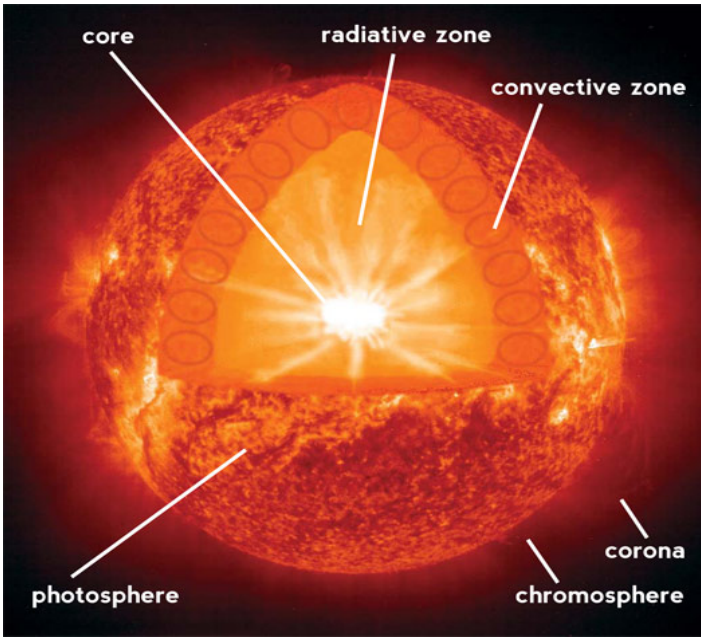

Figure 1: The Layers of the Sun. A simplified diagram of the sun's various layers. Courtesy of NASA.

Outside of the core region where fusion occurs is a layer of the sun where the pressure and temperature are not high enough to maintain fusion. The energy density of

this layer is high enough to keep most of the atoms completely ionized most of the time. This being the case, there are no orbiting electrons to absorb the photons produced in the core. These photons can only be scattered, and the majority of the energy passing through this region from the core is in the form of these photons.

Farther out from the radiation zone is a region where the energy traveling out from the core has diffused enough for the nuclei to hold electrons. Thus, in this region photons may be absorbed. As such, this region is opaque. Further, the energy arriving into this region as photons can result in the motion of these atoms, causing currents where high energy atoms may move downward to become ionized or, more often, upward to bring this energy to higher levels. Since this energy is transferred in this region by convection, this is the convection zone. This zone is only about half as thick as the radiation zone.

Above the convection zone is a thin layer of less opaque gas, only about 500 km thick, called the photosphere. Most of the radiant energy received from the sun originates here. This is usually considered the sun's surface, and is the deepest layer of the sun that can be directly observed with optical instruments. The temperature in this region varies greatly by depth, but is about 5800 K (Seeds, 147).

In the photosphere, free electrons moving within a range of speeds may become bound to atoms, either ionized or neutral, which would result in negatively ionized atoms. This process would result in the release of photons with a continuous range of energies. These negative ions may lose their weakly bound electrons by absorbing photons of a range of energies, the excess energy in the photon imparting kinetic energy to the freed

electrons. For this reason, the photosphere is opaque and radiates a continuous spectrum. This is why the sun appears to approximate a black body at about 5800 K. The photosphere is the region where such features such as sunspots, granulation, and supergranulation are observed.

Above the photosphere is a fairly shallow region of thinner, more transparent gas called the chromosphere. This region gets its name from its red appearance around the photosphere during an eclipse. Still greater in extent than the photosphere, the chromosphere averages about 5000 km through, and extends irregularly to about 9000 km. The chromosphere has a less regular shape, the boundary including spicules, short lived columns of rising gas that reach up into the corona. Another feature of this region is filaments, elongated dark areas. In this less dense region, the temperature rises farther out from the surface. Due to its increasing temperature and rarity, the chromosphere has a characteristic emission spectrum consisting mainly of Balmer hydrogen lines.

Between the chromosphere and the much hotter corona there is a section, sometimes considered part of the chromosphere, called the transition region (Phillips, 118). In this extremely non-uniform zone, the temperature rises from the chromosphere's temperature range up to temperatures like those found in the corona.

Outward from the transition region is the solar corona. If the chromosphere's tendency to rise in temperature at higher altitudes is surprising, then the corona is downright shocking, because it is at about 2 million Kelvin, though it is by no means uniform. This very hot outer atmosphere is also extremely rarefied. Like the chromosphere, the corona is not easy to observe with ordinary instruments except during

a total eclipse, where it is visible to the naked eye. Because of its extremely high temperature, the corona has a very broad range of emission lines, but because of its tenuousness, its luminosity is quite low compared to the rest of the sun. The corona's boundaries and extent can be difficult to define. Often its beginning is arbitrarily designated as anything outward of the transition region over a certain temperature, about 100,000 K (Golub and Pasachoff, *The Solar Corona,* 2). As this region is very dynamic, its shape is also highly irregular, largely dominated by streamers, plumes and other large structures. These regularly reach hundreds of thousands of kilometers outward. The mechanism behind the extreme temperature of this outer atmosphere, which by all rights should be of diminishing temperature, is a major mystery. This shall be explored in further detail later.

The corona has a higher density than interstellar or even interplanetary material, and because it can conduct heat well and is so hot, the gravitational pull of the sun is unable to keep it perfectly trapped. This allows the outer parts of the corona to expand continually, all the time accelerating. The outer corona is said to become the solar wind at a point where the accelerating particles reach the acoustic speed of the outer coronal plasma, at a distance of about 1.4 solar radii. This stream of charged particles spirals out from the sun into interplanetary space along the sweeping magnetic field lines generated by the sun as it spins. It is this thin and very fast ionic gas that causes cometary tails and Earth's aurorae.

Somewhere near this point it becomes difficult to describe features as belonging to the sun, but rather related to it, or affected by it. The region filled by the solar wind is

called the heliosphere, though this may be misleading as it is suspected that this may not be spherical or even symmetrically shaped, but probably extends out 40 to 100 AU (Phillips, 239). This boundary is called the heliopause, and at this point the solar wind drops back to subsonic velocities

Of course the sun will have effects and send particles and light out far beyond this point, but the regions inside this can be said to be the sun's demesne, and in a manner of speaking they could be said to be parts of the sun. Anything beyond this is unlikely to be relevant to the current discussion.

1.3 Solar Phenomena

To gain a better understanding of the sun's processes, it is desirable to examine a few phenomena that occur observationally in the sun, as well as the techniques by which they can be observed. A fairly easy phenomenon on the sun both to observe and to understand is that known as limb darkening. When observing images of the sun optically, the area right around the edge, or limb, of the solar disk does not appear as bright as the rest of the image. This is a simple optical depth effect. Since the photosphere is not perfectly opaque, the line of sight of the image can penetrate it to a certain depth. When looking straight on to the photosphere, this depth is greater than when looking at an extreme angle, such as at the limb. This greater depth also has a corresponding greater temperature, and is therefore brighter. To put it another way, at the steeper angle at the limb, the same depth will be viewed as in the center, but through more photospheric material, and thus it will appear darker.

It is possible to obtain spectral information and find the spectral lines shifted slightly one way or the other, or to find the lines crooked or 'wiggly' rather than straight. If the source of the signal being read is moving with respect to the observer, the wavelength of the observed radiation will be Doppler shifted. The effect will shift the lines toward the blue end of the spectrum (shorter wavelengths) for motion toward the observer, and toward the red end (longer wavelengths) for motion away. If there is oscillatory or combined motion, the observed lines may wobble or be recorded as wiggly between the two positions.

Spectra of the photosphere, when examined closely, show movement both toward and away at about 7200 km/h. Visual observations of the photosphere show a phenomenon known as granulation, a constantly shifting pattern on the surface of small bright areas about 1100 km across (Phillips, 74) separated by narrow darker channels. The bright granules appear and disappear constantly. The Doppler information shows that the bright granules are moving upward, while the dark channels are sinking into the sun. These bright granules are convection cells, much smaller than the ones in the convection zone. Hot plasma, about 400 K hotter than in the darker lanes, boils to the surface, and after it has cooled sinks back down into the photosphere, the hotter and cooler plasma moving at about 2 km/s relative to each other.

If examined carefully, the granules may seem to oscillate slightly in brightness with a period of about 5 minutes (Golub and Pasachoff, *Nearest* Star, 67). Also observed have been Doppler shifts of spectra of the photosphere of about half a km per second, first one direction and then the other, with periods of about five minutes. These persist in

an area for about half an hour before fading and being detected somewhere else. These are the surface effects of waves traveling within the sun. These sonic helioseismic waves travel down into the sun, are redirected by the varying density back toward the surface, and are reflected back inward when they hit the surface. A mathematical model of the sun, including maps of the internal structure as related earlier along with temperature and density assumptions about internal regions, can be used to predict the behavior of these waves. By comparing this prediction with the actual recorded waves, the model can be tested.

Sunspots are areas on the photosphere over 1000 K cooler than the surrounding surface. They typically appear at lower latitudes, and can last anywhere from under an hour to over six months. At any given time less than 1% of the sun's surface is covered by sunspots (Phillips, 168). They develop from pores, tiny dark places that generally last under an hour. The Zeeman effect, the splitting of some spectral lines under magnetic influence, indicates that magnetic fields are coincident with sunspots. Sunspots always appear in pairs or groups, and the field strength has been measured at about 3000 gauss (Golub and Pasachoff, *Nearest Star*, 40-41).

There are a few other structures associated with sunspots. Plages are bright areas around sunspots that seem to be a brightening in the chromosphere caused by magnetic field activity and that remain for a time after the sunspot is gone. Groups of sunspots are surrounded by an area of the photosphere that is a little hotter than the surrounding surface, and which is visibly slightly brighter, called a facula. Probably more interestingly, above sunspots are vertical structures made up of individual filaments that

extend up from the spot about 10,000 km. Within this structure, in the chromosphere, radio emissions are detected, and in the corona the temperature is lowered.

The number of these sunspots increases and decreases with a cycle of about 11 years. Throughout this period, new sunspots appear at lower latitudes, and there is a general drift of spots toward the equatorial regions. The leading spot of any pair in each hemisphere is of the same polarity. However, after two cycles, or what could be called a full cycle of about 22 years, this polarity reverses (Kutner, 115). Each cycle of sunspots is not necessarily equal in strength, as viewed by the number of spots, and any pattern in the strength of cycles is not yet predictable.

There may be even more variation than this. There was a period of about seventy years in the late seventeenth and early eighteenth century when sunspots were apparently largely absent. This period is named the Maunder Minimum after the twentieth century scientist who discovered this in historical records. During this period, few aurorae were reported, indicating a possible period of generally less solar activity. To further support this link to other solar activity, this period corresponds to a period in Europe known as the Little Ice Age (Phillips, 44). In old trees alive at that time, an odd pattern of growth rings has been shown for this period. Other sets of similar growth rings have been found, possibly indicating other similar periods of low activity. This would tend to indicate that the 22-year cycle is not only the cycle of the number and behavior of sunspots, but that this might be only an indication of other processes at work.

Since sunspots are always an area of intense magnetic fields, this cycle of sunspot activity may be indicative of a cycle of magnetic activity. The source of the sun's

magnetic field is of critical interest. While there is a hypothesis that the sun's magnetic field is primordial, trapped by the collapsed plasma, it is more likely that some kind of dynamo mechanism within the sun generates it. The interior of the sun is composed of conductive ions. This plasma, an efficient conductor, will lock a magnetic field passing through it into place, either dragging that field with it, or being dragged by it.

In the Babcock model, the field starts out as toroidal, with poles more or less at the sun's rotational poles. The differential rotation causes the equatorial region of this field to be pulled around the sun while the polar regions remain the endpoints. Kinks in the field lines emerging through the photosphere would account for sunspots, their pairing, the leading polarity, and their general equatorial movement through the cycle. Toward the end of the cycle, the trailing field components move toward the poles, closing the circumsolar loops in the field lines, neutralizing polar components of the field, and replacing it at reduced strength. The remaining field is similar to the polar field before this, but of opposite polarity. This new configuration is wrapped around the sun, but in the opposite direction, and the differential rotation eventually brings it back to toroidal. Of course other models exist. Even the dynamo theory itself, while it seems likely, is not a certainty.

The previously mentioned solar oscillations can be used to determine some of the specifics about the structure of the sun's interior, such as interior plasma flows, which would generate the sun's magnetic field. Any solar model tested by observing these waves must include any subsurface differential rotation that the observations indicate, and in doing so should be able to account for the magnetic field characteristics.

One major mystery in observational science concerns the sun's energy source. The nuclear reactions assumed to take place at the core of the sun produce neutrinos at such a rate that the flux of these extremely non-interacting particles at the distance of the earth should be an amazing 6×10^{14} per square meter every second. Neutrinos have almost no interaction with matter, and are therefore hard to detect. However, some of these neutrinos, the ones from a reaction creating boron (^{8}B) have enough energy to interact with a chlorine isotope (^{37}Cl) to produce an electron. An American experiment called Homestake used this interaction with chlorine to attempt to detect these higher energy solar neutrinos. This experiment consisted of an enormous tank located far underground to shield it from any other energy sources, as neutrinos pass through most matter without interaction. Surprisingly this experiment only detected about a third of the expected number of neutrinos (Phillips, 55).

A series of other experiments, such as the Japanese Kamiokande and its descendents, use other detectors for the same kinds of neutrinos. The Russian and Italian experiments use gallium to detect

lower energy neutrinos. The Canadian SNO used a tank of heavy water to detect all kinds of them. In general, these experiments detect about half the expected neutrino flux (Golub and Pasachoff, *Nearest Star*, 109).

There are a number of possible explanations for the discrepancy between detected and expected neutrino fluxes. A hypothetical particle, the Weakly Interacting Massive Particle, or WIMP, created early in the universe, may be responsible for some of the energy transfer within the sun. However, this is not seen as the most likely explanation.

Another possibility, involving a lack of understanding of neutrinos themselves, is called the MSW, after Mikheev, Smirnov and Wolfenstein. This is a quantum mechanical effect that allows the neutrinos to change to another type before arriving at the detector. This would necessitate an oscillation of neutrinos between their different forms (Phillips, 57). It is also possible that some process within the sun prevents detection of the neutrinos, or that a process at the core of which we are not yet aware reduces the neutrino output, or that the exact ratio of types of nuclear reactions in the core is not what is expected from models. As more information is gathered on the neutrino emissions, as well as on the interior of the sun, it can be hoped that this enigma will be solved.

As the focus of the current work is on the corona, it would be useful to describe the corona in somewhat more detail. As has been seen earlier, the solar corona is an irregular envelope of extremely hot gas, highly ionized, that forms the upper atmosphere of the sun. It is more circular at sunspot maximum, and more flattened at minimum. Because it is so faint by comparison with the solar disc, the corona is not visible from the ground except during a total solar eclipse. As it can be seen during such an event, it seems obvious that humans would have seen the corona on a number of occasions throughout history. However, though there are reports that seem to indicate a sighting of the corona, the first unambiguous mention of it is by Kepler in the seventeenth century, though he thought he was viewing the lunar atmosphere.

In 1860 observations by De la Rue and Secchi at different locations showed that prominences (structures extending up through the corona) were solar in nature and not effects of the earth's atmosphere, as generally believed at this time (Golub and Pasachoff,

The Solar Corona, 27). They showed in photographs the gradual covering of these prominences by the moon. In 1868 Lockyer and Janssen found they were able to observe and measure these prominences even after the occurrence of an eclipse.

Frankland and Lockyer determined that a yellow emission line observed in the preceding eclipse did not belong to any known element. Thus they believed it to be a new element, and named it helium. The subsequent discovery by Young and Harkness of a green line, also not known in any element's spectrum, was initially called coronium. It was eventually discovered that this line came from iron ionized 13 times, Fe^{+13}. This was instrumental in the discovery of the spectra of highly ionized states of elements.

Observation of the corona has changed much since those times. Coronagraphs, generally built at mountain top observatories to reduce atmospheric interference, use an occulting disc to block the photosphere and get an image of the corona. Some of these ground-based observatories are still in use, such as the High Altitude Observatory at the National Center for Atmospheric Research in Boulder, CO. Sounding rockets have also been used for coronal observations. Following a ballistic trajectory, they allow about five to fifteen minutes of observation before the instrument reenters. More recently, these instruments have been mounted on satellites. Since the focus of the current work involves observations made entirely by these spacecraft, a brief overview of some of the more significant recent space-based missions is prudent.

Skylab contained the Apollo Telescope Mount, or ARM, which had eight instruments for observing the sun between 2 and 7000 angstroms (Golub and Pasachoff, *The Solar Corona*, 179). The Solar Maximum Mission ran from 1980 to 1989 and

carried a suite of instruments to study flares. Its instruments were sensitive over a broad range of wavelengths. The Yohkoh satellite, launched in 1991, carried the Soft X-ray Telescope and the Hard X-ray Telescope (SXT and HXT), as well as the Bragg Crystal Spectrometer and the Wide Band Spectrometer. The Solar and Heliospheric Observatory (SOHO), launched in 1995, had what was then the largest compliment of instruments for studying the sun. It included both spectrometers as well as instruments to study the solar wind. Specifically for studying the corona were the Solar Ultraviolet Measurements of Emitted Radiation (SUMER), the Coronal Dynamics Spectrometer (CDS), and the Extreme-ultraviolet Imaging Telescope (EIT). The Transition Region and Coronal Explorer (TRACE) is a telescope with ultraviolet and extreme ultraviolet channels, designed specifically to investigate the relationship between the surface magnetic fields and coronal structures.

The emission in the corona is not a continuous spectrum of radiation, but rather a set of discrete frequencies. Discrete spectral lines can be explained using the Bohr model of the atom. Electrons occupy discrete orbitals with angular momentum restricted to integral multiples of Planck's constant, ħ. Transitions from one orbit to another occur by emission or absorption of a photon, and only in these discrete amounts of energy. These energy levels correspond to the frequencies found by Fraunhofer. Balmer found the progression of visible hydrogen lines, and now these are called Balmer lines.

Among other things, the coronal emission lines allow for the determination of the temperature of the material from which the particular signal was emitted. For instance, Fe XIV exists in the coronal spectrum. Since it would normally de-excite or excite to higher

energy by collision, it means not only that the corona is very hot, but also that it is of very low density, around 10^{14} electrons per cubic meter (Phillips, 138). With a particular mixture of elements at a particular temperature, it is possible to calculate the number of atoms in any given volume that are in a specific ionization state and which emission lines are produced by these. By adding up all these states, as well as electron capture and escape contributions, it is possible to arrive at a total spectrum of the material.

Electron densities can be determined, within a certain region, by using the ratio between a likely radiative transition of an ion excited by collision, and a metastable state, where the decay's likelihood is a function of the density. Abundances are also necessary to analyze spectrographic observations of the corona. Since hydrogen, which is the usual baseline, is difficult to detect in the corona, these abundances are relative. Analyses of different spectra can provide data to which a model abundance can be fit. Further, measurements of the solar wind and other samples of particles originating in the corona can show aspects of the coronal abundances.

Probably the greatest mystery about the corona is why it is so hot, even though it is farther out from the core than other, cooler layers of the sun. For this to be the case the corona must be continuously heated, and in some other way than simple conduction or convection, since the material immediately below it is cooler. In fact, because the temperature actually rises through the chromosphere and transition region, the coronal heating mechanism must be non-radiative and must originate in the photosphere or below. There seems to be a general agreement that the heating mechanism of the corona is driven by the sun’s outer convection zone.

Different parts of the corona range from one million to ten million Kelvin. The corona's temperature relative to the immediate inner layers requires heating by some non-thermal source. Yet because of its low density, this requires only .01% of the sun's energy output. This is still a very great amount, and is only small compared to the output of the sun. It is thought that magnetic fields are basic to the heating of the corona. For example, the coronal temperature is higher in regions near sunspots, even though the corona's emissions are suppressed above the center of a sunspot. Further, by studying other stars, it is known that there is a correlation between the magnetic flux at the surface and coronal-type x-ray emissions. There is also a correlation between the rotational rates and the coronal emissions of these stars (Golub and Pasachoff, *The Solar Corona* , 18).

To better understand this mechanism, it is necessary to map and model the magnetic fields that influence the corona. However, if the magnetic fields originate below the visible surface of the sun, the mechanism that causes them cannot be directly observed. The coronal magnetic field strength is also difficult to measure directly. The Zeeman effect is too small to be measured in the corona's spectral lines. However, since the corona is affected - in fact, thought to be heated - by these fields, it should be possible to find features that can reveal the properties of them. Coronal structures are usually linked to magnetic fields at the solar surface, and active regions seem to be magnetically created and direct coronal plasma.

The activity on the sun is driven by the emergence of magnetic flux lines from the surface. This activity leads to coronal structures. These structures, being driven by the

magnetic cycles in the sun, must be taken into account in any solar models so that the magnetic driving mechanism is consistent.

In the photosphere and presumably lower, the ionic currents create the magnetic fields. In the corona, however, the density of the plasma is so low that the individual ions become trapped around a field line, moving in a helical path along the line (Phillips, 155). Footpoints of coronal structures are anchored in the photosphere, and are generally cooler than the rest of the structure. Ions travelling along a field line are reflected by the denser gas at the footpoint of the structure, but lose energy to the cooler gas below. This is one way the corona loses energy. However these structures do not actively exchange much thermal energy with the surrounding coronal gas. A modern model of the corona understands it as a collection of these individual structures driven by magnetic fields generated in lower levels of the sun. These are generally called loop models.

Structures called loops (see fig. 2) appear to emanate from the edges of sunspots and arch up into the corona and back down again. Loops are connected to sunspot groups, and are generally about 10,000 km long. Coronal arches (or interconnecting loops) are fainter, more stable, and generally much larger, often hundreds of thousands of kilometers long. Arches connect to active regions. Comparisons between magnetograms and x-ray images show that loops and arches connect regions of opposite magnetic polarity. Their height is generally indicative of the distance between the regions they connect.

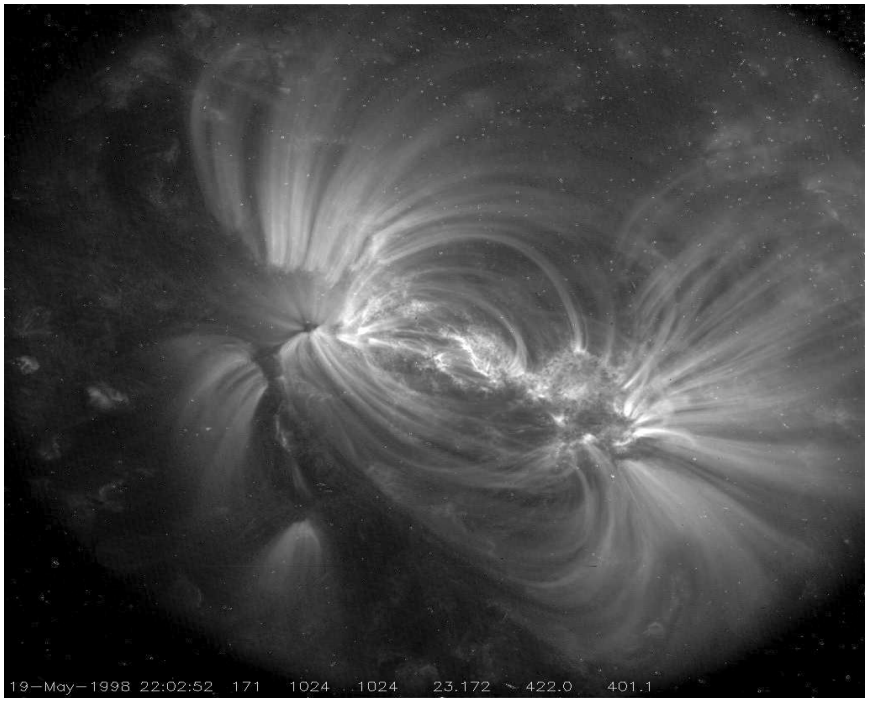

Figure 2: Coronal Loops. An X-ray image in 171 Angstroms of some coronal loops. Taken by the TRACE instrument on May 19, 1998 at 22:02:52 UT.

There are other features, structures and phenomena in the corona, besides loops and arches. X-ray bright points are distinct areas with intensified x-ray emissions. They do not appear to be associated with sunspot activity. They only last a few hours, and are up to about the size of a small group of sunspots. Bright points are always associated with otherwise unmarked opposite polarity magnetic features in magnetograms. They are located away from active regions, and are often seen at higher latitudes than active regions, and are much shorter-lived (Golub and Pasachoff, *The Solar Corona*, 177). The analysis presented in this thesis relates extreme ultraviolet and X-ray observations to not

only study bright points, but also to cross-calibrate two instruments used to observe the sun: the Hinode EUV Imaging Spectrometer (EIS) and X-Ray Telescope (XRT).

2. Overview of Instruments

2.1 General Information about *Hinode*

The *Hinode* spacecraft is an earth-orbiting satellite intended for solar observation. Launched at 21:36 GMT on 22 September 2006, it is a follow-up to the Yohkoh satellite which was launched in 1991. Originally named Solar-B, after its successful launch the satellite was renamed *Hinode* which means “sunrise” in Japanese. While a number of institutions and individuals are behind different aspects of *Hinode*, the overall mission, the launch, and the satellite itself are mainly attributable to the Japan Aerospace Exploration Agency's Institute of Space and Astronautical Science (ISAS/JAXA) (Kosugi et al.).

Intended to help better understand the sun's variation and its mechanisms, the mission purpose of the spacecraft is to investigate the energy transfer between the photosphere and the solar corona and to determine the cause of phenomena such as solar flares and coronal mass ejections, undoubtedly associated in some way with such energy transfers (Culhane et al.). Another main focus of this mission is to measure the structure, features, and properties of the solar magnetic field. It does this by using its optical instruments to deduce the magnetic effects on photon emission in the solar plasma.

The vehicle itself measures 4 meters long and masses 900 kilograms. The satellite orbits in a sun-synchronous, circular pole-to-pole orbit. At its altitude of 680 kilometers, with a 98 minute orbital period, the vehicle is in continual solar view for about 9 months per year (Kosugi et al.).

Hinode is intended to take observations in the visible, X-ray and extreme ultraviolet ranges of wavelengths. To this end it carries three instruments observing individually within these three ranges, so that it can encompass different approaches as they are applicable to different types of measurements of solar phenomena. These instruments are the Solar Optical Telescope (SOT), the X-Ray Telescope (XRT), and the EUV Imaging Spectrometer (EIS) (see fig. 3). These instruments are designed to compliment each other, and thus the whole forms an orbiting solar observatory.

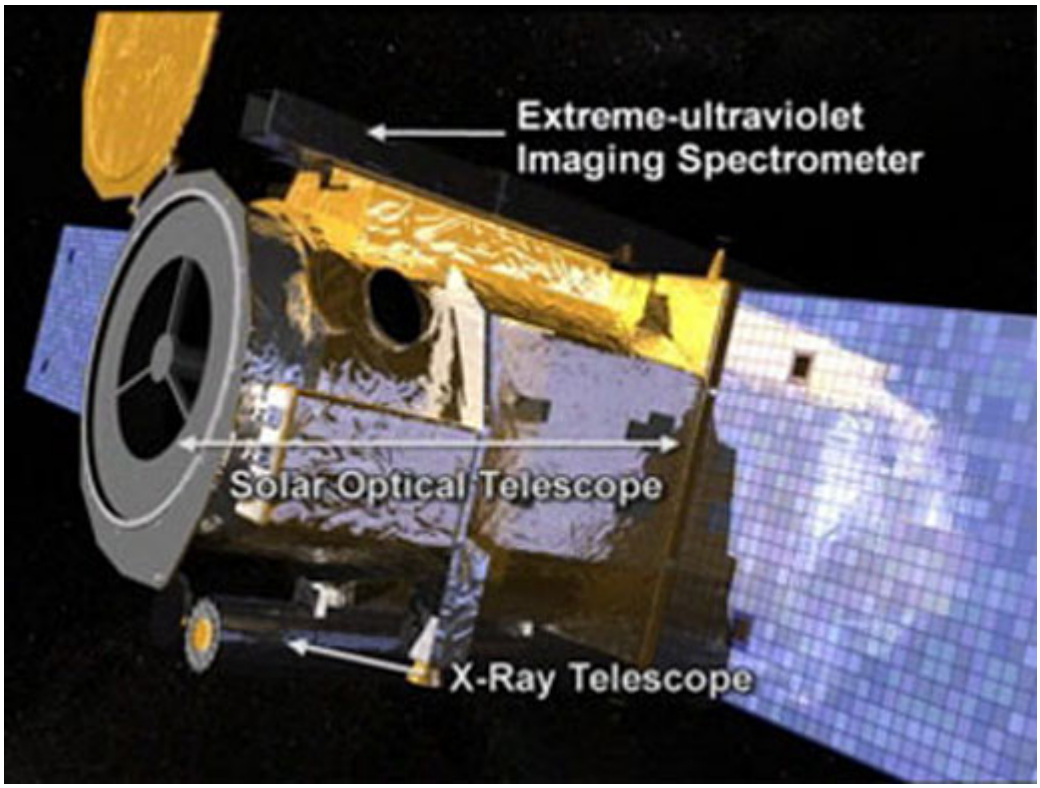

Figure 3 The Hinode Spacecraft. Conceptual image of the Hinode spacecraft showing relative locations of the carried instruments. Image courtesy of NASA's Hinode website. <http://www.nasa.gov/mission_pages/hinode/instruments.html#spectrometer>

2.2 General Comparison of *Hinode's* Instruments

The Solar Optical Telescope (SOT) is a 50 cm Gregorian telescope operating in the visible spectrum which is intended primarily for imaging the photosphere and for spectropolarimetry. At launch, SOT was the largest space-born solar optical instrument to date. The X-Ray Telescope (XRT) is a 30 cm grazing incidence reflecting imager operating in the X-Ray spectrum, at wavelengths from 6 to 60 Å. EIS is an imaging spectrometer using a normal incidence primary mirror and it operates in two wavelength ranges in the extreme ultraviolet between 170 and 290 Å. As the current topic of interest is activity and structures in the corona and upper transition region, the X-ray and ultraviolet instruments are of primary focus.

As a telescope XRT boasts high spatial resolution combined with a speed of image capture not possible with a spectrometer. An XRT image provides a true 'snapshot' of the emissions at the time of observation, rather than as a scan across the field of view. In contrast, a spectrometer will take data progressively and the entire image will be built up over this period, each part of the image reflecting both a spatial and a temporal displacement.

From a perspective of analysis as well as data transmission and storage, XRT allows for faster processing and interpretation than the much weightier information provided by the companion spectrometer. However, this obviously comes with a corresponding lack of spectral detail. Conversely while its spatial resolution is lower, EIS is capable of far greater spectral discrimination, which is useful for multi-thermal analysis. The two instruments when brought to bear on the same well chosen event are

complimentary. XRT can be co-aligned with EIS to 1 XRT pixel (Golub et al.). They can be used to analyze the same structure with different insights. This makes them mutually corroborative and increases the confidence in the result. Alternatively, there may be specific events of interest that are more suited for one or the other instrument, possibly observable only within one's spectral range. Their combination gives a wider opportunity to observe such phenomena meaningfully. To understand the data obtainable with this spacecraft, and any subsequent analysis, it is necessary to look at each instrument individually in more detail.

2.3 Solar Optical Telescope

The Solar Optical Telescope, or SOT, is a Gregorian reflecting telescope operating in the optical wavelength range. At time of launch, it was the largest space-born optical instrument for solar observations (Culhane et al.). SOT was constructed by Lockheed-Martin and the National Astronomical Observatory of Japan (NAOJ). In addition to visible wavelength observations of the photosphere, it is also able to take magnetograms.

SOT consists of two modules: the Optical Telescope Assembly and the Focal Plane Package. Being separate units, one is concerned with manipulation of photons, and the other with detection and recording of the received properties of those photons. However, they can be considered as the single instrument they comprise.

SOT has a 50 cm aperture, and the separation between the primary and secondary mirrors is 1.5m (NAOJ website). It is diffraction limited in the wavelength range of 2880 – 6700 Å. This means that the optics are of sufficient quality to produce the maximum

theoretical angular resolution for the telescope's area in this range, which is .2 - .3 arc sec. The CCD detector is capable of a resolution of .22 arc sec. Beyond the primary mirror there are multiple optical paths for narrow and broad band imaging, polarimetry, and correlation tracking for image stabilization (Kosugi) (see fig. 4). As the current objectives relate to the corona and upper transition region, and specifically to structures that do not radiate significantly in the visible spectrum, further detail on this instrument will not be necessary.

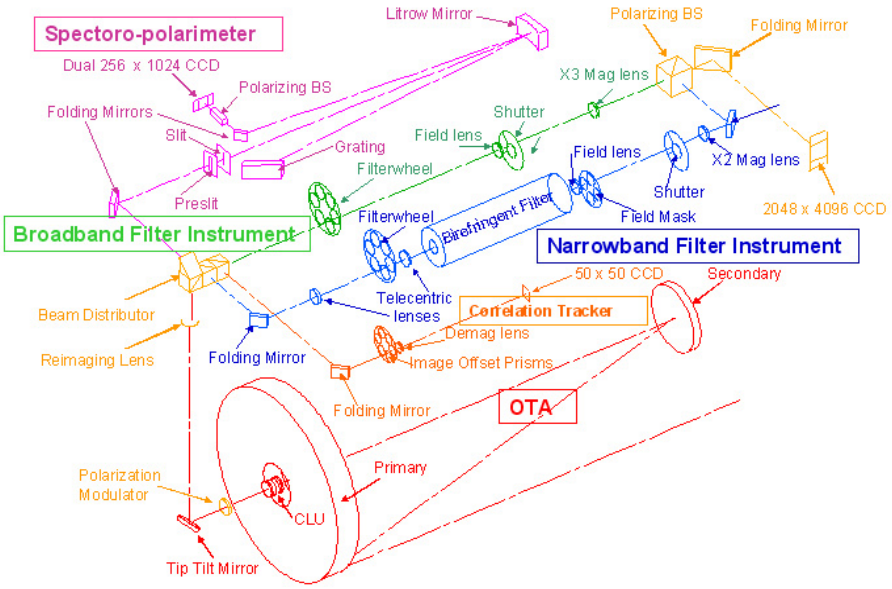

Figure 4 Diagram of the SOT's Optical Telescope Assembly. Image courtesy of NAOJ's SOT website. <http://solar-b.nao.ac.jp/sot_e/optics_e.shtml>

2.4 X-Ray Telescope

Emissions from the corona are primarily in soft X-rays, its temperature being in the millions of kelvins (Golub et al.). Consequently, an X-ray imager is needed to resolve structures in the plasma that makes up this region. Two complimentary methods are used aboard *Hinode* to observe these structures as they appear in different wavelength regimes. One is the use of an imaging spectrometer (EIS), which shall be discussed later, and the other is the X-Ray Telescope, or XRT.

The body of XRT is a tube of carbon-fiber-reinforced polymer with low thermal expansion characteristics. The entrance aperture is 341.7 +/- .1 mm and is a thin film filter consisting of 1200 Å thickness of aluminum on a 2500 Å backing of polyimide. Though care was taken to handle the aluminum filter in a low humidity nitrogen atmosphere to minimize oxidation, there is an estimated 100 Å layer of aluminum oxide. This prefilter serves to shield against the entry of visible wavelengths. It is also important to reduce the heating of the body cavity and the elements within (Golub et al.).

Since the instrument has a secondary, visible imaging system that uses a coaxial optical path to the X-ray optical system, there are in fact two primary elements. Treating the visible system as a whole, it is an achromat assembly with an independent focusing mechanism to allow alignment of the focal plane for the visible imager with that of the X-ray imager. The X-ray primary is a grazing incidence mirror of non-conic, high order polynomial surface (Golub et al.). This Wolter-Schwarzschild design allows for the use of a single optic with no secondary. Although this sacrifices image quality near the center of

the image, these imperfections are acceptable and nearly within the tolerances of the necessary imperfections of the manufacturing process.

The optical path passes through two filter wheels containing analysis filters, allowing a combination of two filters. These combinations define a number of X-ray passbands for the resultant image. Near the focal plane there is a removable glass filter that is used when operating the visible light imager and when the X-ray imager becomes saturated by higher than normal intensities. The shutter is a focal plane of the rotating blade type. It has both a narrow and large opening around its circumference, and it can either be swept across the field of view, or it can rotate the large opening into position for an extended exposure.

The single CCD detector itself is movable by 1 mm to bring it into the focal plane of the X-ray optics. This detector is shared by the X-ray and visible imaging systems that operate along the same axis. Ultimately, the focal length of the optical system is 2707.5 mm, though it varies somewhat by field angle (Golub et al.) (see fig. 5).

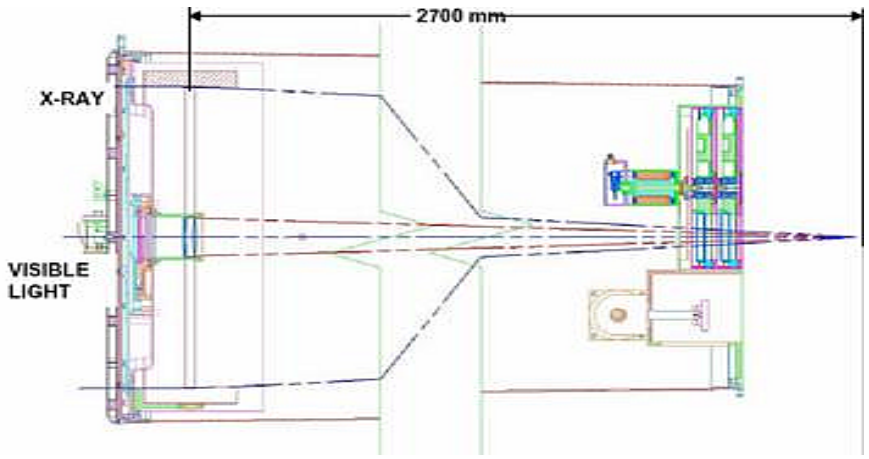

Figure 5 The XRT Optical Path for both X-ray and Visible Imaging. Image courtesy of NAOJ's XRT website. <http://solar-b.nao.ac.jp/xrt_e/optics_e.shtml>

The resulting machine has a field of view of 35 arc min with optimization for 15. It's observable wavelength range is 6 – 60 Å. Exposure times can be varied from 4 ms to 10 sec with exposures being taken as little as 2 sec apart. The transmission characteristics of the individual filters on the filter wheels can be found in table 1. These show the percentage transmission of the indicated spectral line for each filter, and are used with the detector characteristics to produce the instrument's response functions. The ones shown are the experimental values, without associated errors.

Table 1 Transmission Characteristics of the XRT Filters by Emission Line. Data from Golub et al.

Emission Line wavelength	C – K 54.7 Å	O – K 23.6 Å	Cu – L 13.3 Å	Al – K 8.33 Å	Mo – L 5.0 Å
Filter					
Al-Mesh	23.8	80.8	93.7	94.5	77.4
Al-Poly	24.0	51.5	77.4	94.5	89.6
C-Poly	64.1	7.9	60.6	79.2	94.1
Ti-Poly	41.8	5.5	33.9	68.8	91.0
Be-Thin	5.6	0.0	26.1	77.8	90.7
Be-Med	6.8	0.0	4.5	48.9	77.2
Al-Med	0.0	0.0	2.3	22.6	2.5
Al-Thick	0.0	0.0	0.0	5.1	0.0
Be-Thick	0.0	0.0	0.0	0.0	7.8

2.5 Extreme-Ultraviolet Imaging Spectrometer

EIS, or the Extreme Ultraviolet Imaging Spectrometer aboard the *Hinode* spacecraft, is an imaging spectrometer designed to observe the discrete spectral emissions of the sun's

corona and upper transition region at separate wavelength ranges between 170 and 290 Å. The instrument is precise enough to allow for observation of the flow velocity of plasma through Doppler effects.

Measurement of an observed spectral line's width compared to an emission profile of that line allows for the interpretation of the magnetic characteristics of the emitting plasma's environment, and accounting for a spectral line's shift gives the relative velocity of the plasma through the phenomenon of Doppler shift. Aside from mapping the flow of plasma on a structure, the velocity measurements of coronal plasma is important in the observation of oscillations and shock waves, which might be a means of energy transfer to the corona and are found associated with events such as flares. Being magnetic in nature, these events can be studied from their magnetic characteristics as well as from their kinetic effects on the surrounding plasma.

The optical and support components of EIS are mounted on a composite structure with low thermal expansion characteristics, which is necessary to keep the optical components in the correct spatial relation and orientation. Since this structure is not very electrically conductive, the harness for the assembly is screened and the structure is surrounded by the conductive Multi-Layer Insulation. These steps are necessary to protect the internal electronics (Culhane et al.).

The entrance aperture for the instrument is a 1500 Å aluminum filter. This is required to prevent the intrusion of visible and other wavelengths.

Toward the back of the optical assembly is the concave primary mirror. This mirror is oriented so that the incoming photons strike it with normal incidence. This is a

significant departure from similar instruments as most materials have low reflectivity at the operational wavelengths of this device, and so grazing incidence has generally been used. Increased reflectivity at normal incidence is achieved through the use of a multilayer coating on the mirror of molybdenum and silicon. Since a coating such as this can only optimize the reflectivity of the mirror for a narrow range of wavelengths, the mirror is coated with differently optimized coatings on each half side. This gives the instrument its operating ranges of 170 – 210 Å for one coating and 250 – 290 Å for the other.

Much in the manner of a telescope, the mirror focuses the incoming radiation on to one of four interchangeable slits, allowing 1, 2, 40 and 266 arc sec widths. These are mounted on a paddle wheel like device (see fig. 6) to allow the desired slit to be rotated into position. Attached to the slit selection assembly is a circular blade shutter.

Beyond the slit the radiation is diffracted by a concave, reflective, aspherical (toroidal) grating. The grating is coated on opposite halves with the same two multilayer coatings as the mirror, and its geometry allows for stigmatic image formation. Despite the dual coating optimized for different wavelengths, the entire grating shares the same 4200 lines per mm spacing.

The final two images, which correspond to the two wavelength ranges, are formed on two thinned back-illuminated CCD detectors which are separated in the direction of dispersion. These CCDs have 1024 x 2048 pixel resolution and are cooled by an external radiator to keep them below -50 degrees C during operation (Culhane et al.).

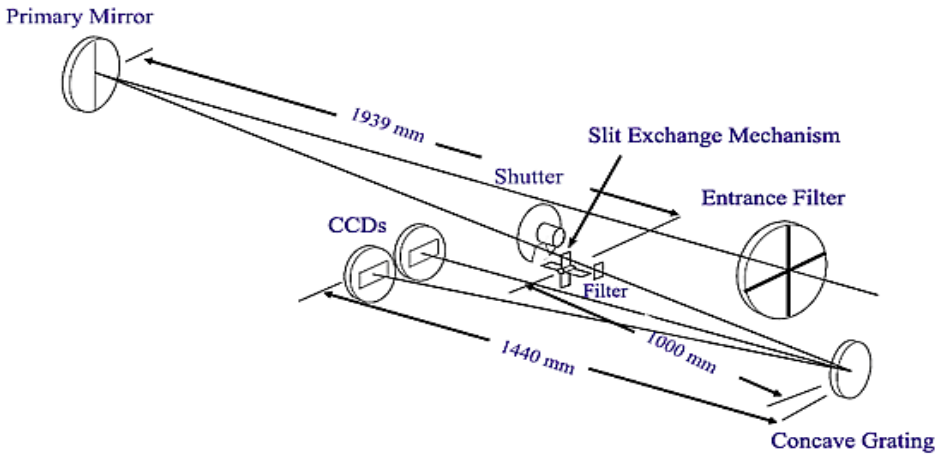

Figure 6 Simplified View of the EIS Optics. Image courtesy of NAOJ's EIS website. <http://solar-b.nao.ac.jp/eis_e/optics_e.shtml>

The primary mirror has a rotating mounting which allows it to be turned by one-eighth arc sec steps in solar north-south. Images are taken by sweeping the mirror across the region of interest and taking exposures at each increment. To mitigate the relatively limited field of view of the instrument as compared to its companion instruments, the primary mirror can be rotated in the solar east-west direction by 15 arc min to offset the instrument pointing from that of the spacecraft.

The instrument electronics packages are separated. Most power conversion is done outside the actual instrument in a separate housing removed about 2.5 meters distant which minimizes noise passed to the detector electronics. Interior electronics packages regulate the bake-out and maintenance heaters, control the CCDs, and pass their output back to the exterior electronic packages (Culhane et al.).

The resultant instrument has a field of view of 6 by 8.5 arc min. It can offset this field of view 15 arc min from the pointing of the spacecraft in solar east-west directions. It has a spectral resolution of 47 mÅ at 185 Å and a 2 arc sec spacial resolution. It can observe as many as 25 spectral lines simultaneously and is capable of rastering a region 4 by 8 arc min in less than 2 minutes (Culhane et al.). This is a short acquisition time for a spectrometer and is important for both the observation of dynamic phenomena and the evolution of coronal loops and other large-scale structures.

2.6 Summary of Hinode

Operating in different wavelength ranges and through different methodologies, these three instruments, which are summarized in table 2, can be used to observe a wide range of phenomena. However, when the capabilities of the instruments overlap, they can be used to obtain a wider range of data about the same observable.

Table 2 Summary of the Instruments Aboard Hinode. Including their primary operational characteristics.

	SOT	XRT	EIS
Instrument Type	Reflecting Telescope	Reflecting Telescope	Imaging Spectrometer
Optics Type	Gregorian	Grazing Incidence	Normal Incidence
Size of Primary	50 cm	30 cm	
Wavelength Range	2800 – 6700 Å	6 – 60 Å	170 – 210, 250 – 290 Å
Angular Resolution	.3 arc sec	1 arc sec	2 arc sec
Acquisition Time		4 ms – 10 sec	~ 2 min

As a complete orbiting observatory, *Hinode* is capable of uploading 2 Mbps of data and can store 8 Gbits awaiting download from its three instruments. In many ways

these observational tools are orders of magnitude superior to those which previous orbital solar observatories had to offer and represent several major innovations in the field.

3. X-ray Bright Points Described

X-ray bright points are small, short lived features in the solar corona visible in the X-Ray and EUV spectral range. They are around 15 to 30 arc seconds across and last anywhere from about 2 hours to 2 days. Being regions of enhanced intensity in this wavelength band, they appear brighter than the surrounding area. Like active regions they are emission features, but tend to be located apart from them and are often found at higher latitudes (Golub and Pasachoff, *The Solar Corona*, 177). They are always present on the sun, but like the visible phenomenon of sunspots, their number varies over the solar cycle. In contrast, their population varies in inverse phase with the solar cycle with there being more of them during solar minimum with fewer during the more active period (Harvey, 1985).

The classification of an emission feature as a bright point is not clearly prescribed, as there is no definitive characteristic that differentiates these smaller phenomena from larger or more long-lived emission regions. Therefor an arbitrary distinction is typically made. Those called bright points are coronal regions of enhanced X-Ray and EUV emission that are less than 1 arc minute across and that last less than 48 hours. It is also important to point out that these features are not necessarily stable. They frequently fluctuate in shape and intensity fairly rapidly, on the order of tens of minutes, and can even produce flares (Strong et al. 1992). For this reason it is important in any analysis of static characteristics of a bright point, or for analysis using an instrument or instruments that take data over a significant amount of time, to ensure that the region in question is sufficiently stable over the course of the observation. Fortunately, variations in the

temperature of a bright point are often indicated by fluctuations in the total X-ray emission of the structure (Kariyappa et al. 2011). This makes it relatively simple to judge the stability of a particular bright point by simple inspection.

Bright points were first described in 1970 from observations by X-ray telescopes carried by rockets (Strong et al. 1992). Since X-Rays and EUV do not penetrate the Earth's atmosphere well, direct observation of the spectrum in which bright points are visible is not possible from the ground. So apart from sporadic rocket flights, only Skylab was able to observe them regularly until the launch of the Soft X-ray Telescope (SXT) aboard Yohkoh.

During these early observations a connection was noticed with another phenomenon observed on the sun – dark points. Located in the chromosphere, these regions of decreased emission appear in the infrared. They are typically observed using the spectrographic line of He I at 10830 Å. Comparison of ground observations in the infrared and these early X-Ray data seemed to show a correspondence between the X-ray bright points and these lower altitude infrared dark points.

The dark points themselves are roughly the same size and duration as bright points and have a similar distribution. They are absorption features in the chromosphere. It is believed that they result from a saturation of the He I population in a triplet state, resulting in enhanced absorption of the He I 10830 Å spectral line. Since the increase in ionization of Helium to this state can be produced by recombination following photoionization by UV photons, this suggests a mechanism for the relation between this type of feature and the UV radiation producing bright points in the corona above. This

absorption phenomenon can be seen in relation to other emissions originating in the corona as well.

Since the bright points could not be observed from the ground directly and orbital observations were difficult and infrequent, dark points had been used to study bright points. However it is now known that even though absorption phenomena always underlie bright points, these regions are not always prominent enough to be recognized as dark points (Golub et al. 1989). Fortunately it is no longer necessary to use dark points as a proxy for bright points, because orbiting observatories like Yohkoh and Hinode can now monitor them directly. Still the relationship between the two observables is notable.

Bright points are also associated with small regions of bipolar magnetic flux. A bright point will always have one of these magnetic regions, but not every one of these magnetic regions will also have an associated bright point. An example of the correlation between bright points and these magnetic regions is shown in figures 7 and 8. These bipolar flux regions can be of two natures. They can be either small ephemeral emergent regions of bipolar flux, or a random encounter of two unipolar regions. The bright points found with emergent regions would be like tiny active regions. While these emergent regions are more common during the more active phases of the solar cycle, there are more areas containing regions of discrete opposite polarity flux and more encounters of the networks during the quiet sun. It would appear from studies of magnetograms that about 2/3 of bright points overlie these meetings of unipolar magnetic flux, which perhaps explains the bright point's greater population during the low points of the solar activity cycle. There also has been observed a correlation between areas with large

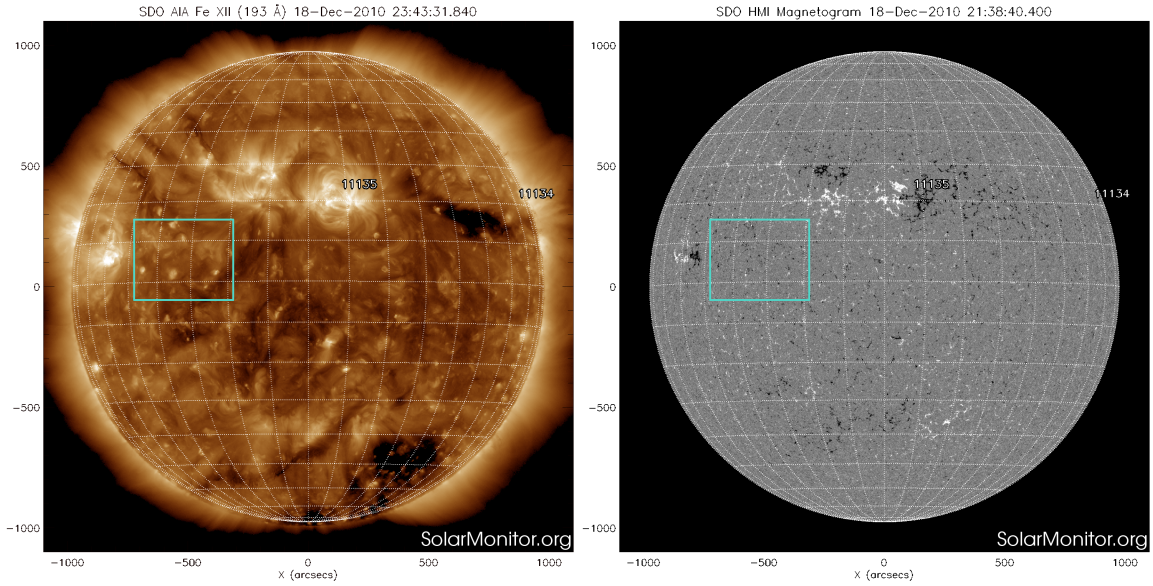

Figure 7: Full Disk EUV Image and Magnetrogram Showing Bright Points. From 18 December, 2010. Left: AIA image in Fe XII 193 A. Right: HMI magnetogram. Boxes show region of interest for Figure 2. Image courtesy of NASA and SolarMonitor.org.

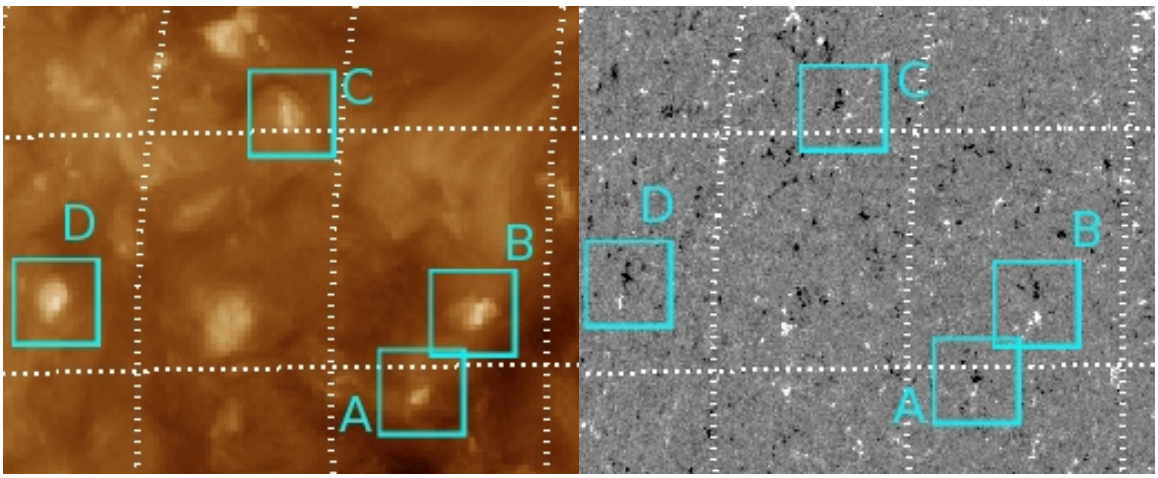

Figure 8: Enlargements of the Area Shown in Figure 7. Left: AIA. Right: HMI. Bright points labeled A, B, C and D in the AIA image are outlined. Their corresponding regions of bipolar magnetic flux are shown in the boxes labled A through D in the HMI image.

population of bright points and areas with high concentration of discrete opposite polarity flux. (Harvey, 1985)

The flares produced by bright points resemble microflares and tend to have an effect on any nearby loop structures (Strong et al. 1992). This would indicate some degree of interconnectedness with the magnetic structure of nearby features and could further support the notion of the bright points as the result of magnetic phenomena.

As ephemeral and dynamic as they are, they appear to be relatively cool coronal structures, only somewhat hotter than the average surrounding plasma (Saar et al., 2009). Also bright points appear to be one of the simplest magnetic driven structures in the corona (Longcope et al., 2001). This structural simplicity makes thermal analysis of them over an area in their core fairly straightforward. Furthermore, they are a property of a region of plasma rather than a discrete structure, as is a coronal loop, for example. As a result, subtraction of any background emission is not necessary. The lack of necessity for such a time consuming and uncertain process can make for speedy and reliable results. Taken as a whole these qualities make bright points a good choice for the cross-calibration of instruments that operate in their spectral range.

4. Data and Analysis

This analysis will use concurrent data from both EIS and XRT in order to cross-calibrate XRT and EIS. Thus it will first be necessary to find spatially overlapping data that is essentially taken at the same time.

An EIS scan was found on 13 May, 2007 at 16:17:20 UT. The field of view of this image is shown on an Extreme-ultraviolet Imaging Telescope (EIT) full disk image in figure 9. Figure 10 shows the EIS spectrum averaged over this range.

A set of full disc XRT images using different filters taken during this scan was found. The acquisition times and filter combinations of these XRT images are summarized in table 3. Figure 11 shows one of these XRT images.

The EIS and XRT data were processed and calibrated using standard software available in the SolarSoft libraries (SSW). The resulting XRT images were cropped to the EIS field of view and all the images were then coaligned using proprietary software written in-house for this purpose. The IDL code for this software is presented in Appendix A. Regions for analysis are then selected, and the intensity values of the pixels in these regions are recorded using the same software. In the EIS data a particular spectral line was eliminated from the set in the case that it had any missing pixels in the region selected. An EIS image in one spectral line and the corresponding region of an XRT image in one filter are shown in figure 12.

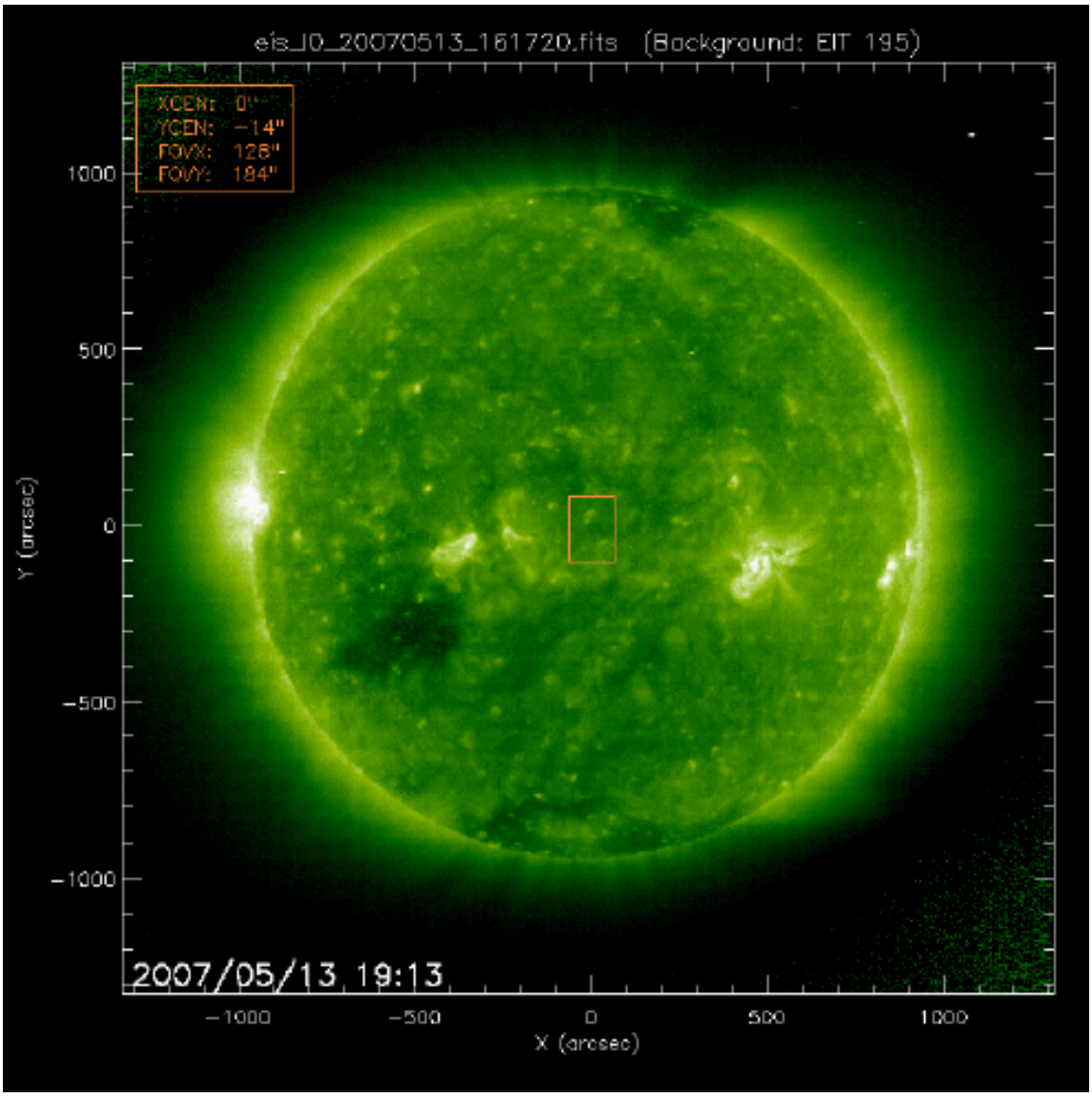

Figure 9 EUV Image of the Region. EUV image from 13 May, 2007 of the full disk of the sun from EIT. The box shows the field of view of EIS. Image courtesy of the SOHO EIT Consortium; SOHO is a joint ESA-NASA program.

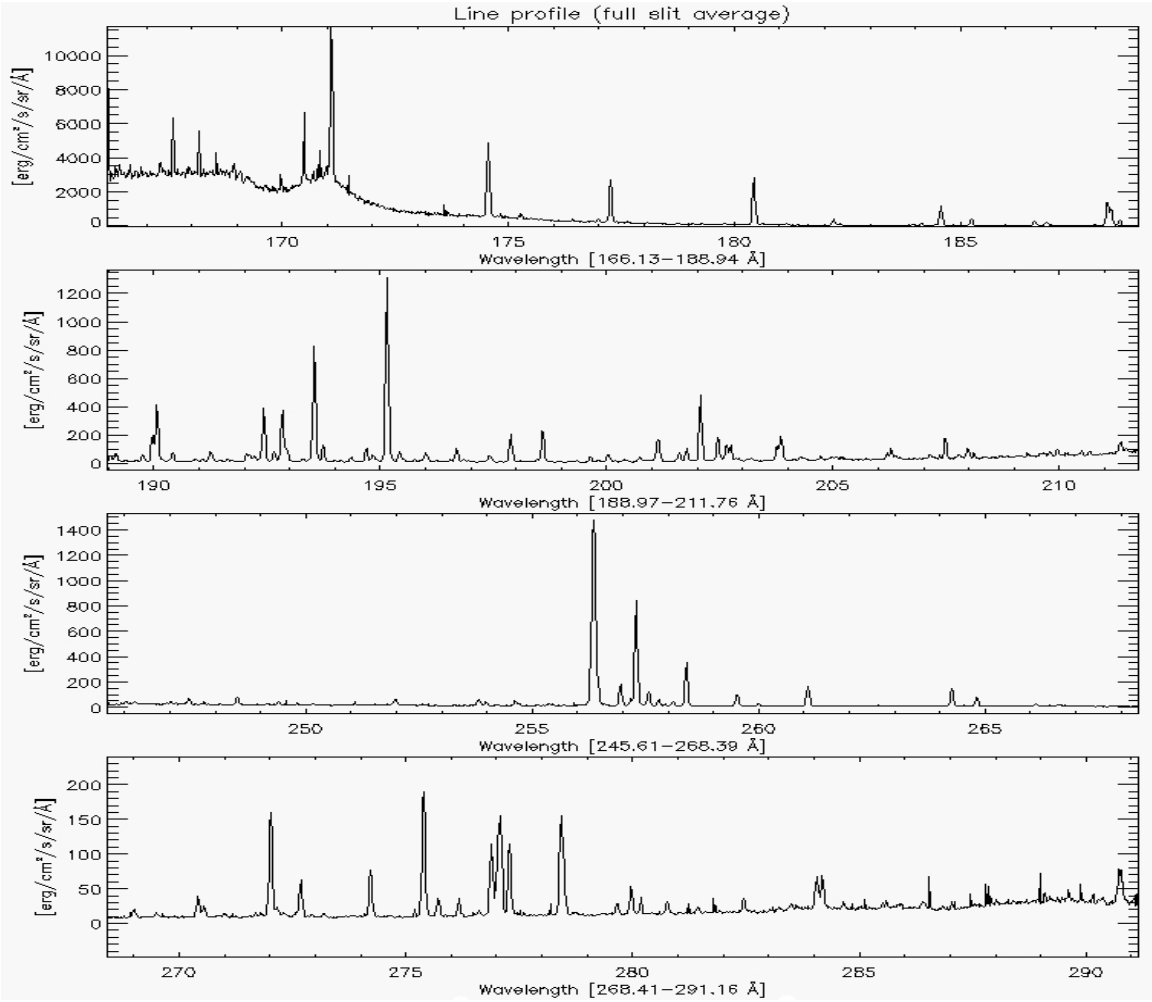

Figure 10 - Averaged EIS Spectrum. 166.13 - 211.76 Angstroms and 245.61 - 291.16 Angstroms over the field of view on 13 May, 2007.

Table 3 XRT Image Information. Acquisition times and filter combinations.

FILTER A	FILTER B	TIME
Open	Al - Mesh	16:21:05
C - Poly	Open	16:26:20
Open	Ti - Poly	16:19:52
C - Poly	Ti - Poly	16:24:38
Al - Poly	Ti - Poly	16:22:55
Thin Be	Open	16:28:49
Med Be	Open	16:30:23
Open	Thick Al	16:18:44
Open	Thick Be	16:17:02

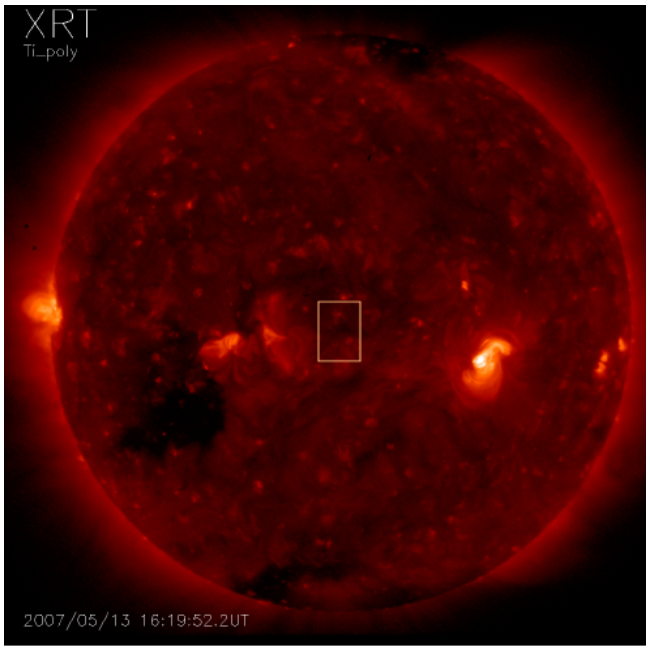

Figure 11 - XRT Image with Ti-Poly Filter. From data set. The box is added to show the field of view of EIS at this time.

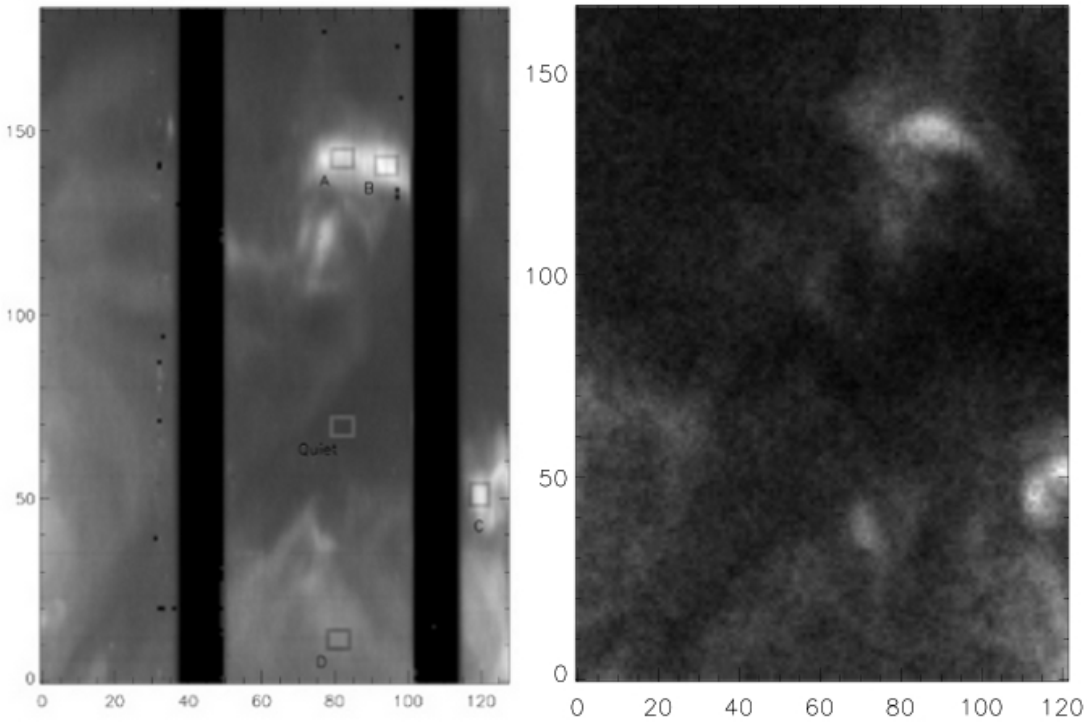

Figure 12 EIS Field of View in Both Instruments. To the left: EIS image in Fe XII 195. Regions A, B and C outlined are the bright points to be analyzed. Region D and the Quiet region are shown for comparison. To the right: The corresponding XRT image in the Ti-Poly filter, coaligned with the EIS image.

The intensities from the individual pixels are then averaged over each region to provide mean intensities for each bright point, A, B and C. In the case of EIS data there is an uncertainty for each spectral line in each pixel arising from the spectral fitting process, and these are propagated to produce the error values. For XRT, the standard deviation from each averaged region are used as the errors. These averaged intensities are summarized in Tables 4 through 7.

Table 4 Intensity Values from EIS Averaged over BP A. Intensity is in ergs / cm^2 / s / steradian. Peak Formation Temperature is Log of Temperature, in Kelvins.

Element	Ion	Wavelength	Peak Form T	Intensity	Error	Notes
Fe	VIII	185.213	5.6	51.43	15.24	
Fe	X	177.240	6.0	315.3	292.8	
Fe	X	184.537	6.0	175.4	16.92	
Fe	XI	180.408	6.1	619.5	105.3	
Fe	XI	182.169	6.1	95.39	19.39	
Fe	XII	192.394	6.1	113.1	8.796	
Fe	XII	193.509	6.1	238.0	8.672	
Fe	XII	195.119	6.1	397.2	11.12	
Fe	XIII	201.128	6.2	76.22	25.66	
Fe	XIII	202.044	6.2	132.4	12.65	
Fe	XIII	203.828	6.2	91.24	26.87	
Fe	XIV	264.790	6.3			Missing Pixels
Fe	XV	284.163	6.3			Missing Pixels

Table 5 Intensity Values from EIS Averaged over BP B. Intensity is in ergs / cm^2 / s / steradian. Peak Formation Temperature is Log of Temperature, in Kelvins.

Element	Ion	Wavelength	Peak Form T	Intensity	Error	Notes
Fe	VIII	185.213	5.6	47.97	15.01	
Fe	X	177.240	6.0	334.0	166.1	
Fe	X	184.537	6.0	176.1	23.57	
Fe	XI	180.408	6.1	607.0	87.79	
Fe	XI	182.169	6.1	101.1	26.80	
Fe	XII	192.394	6.1	125.9	9.668	
Fe	XII	193.509	6.1	282.0	9.447	
Fe	XII	195.119	6.1	436.7	11.36	
Fe	XIII	201.128	6.2	86.91	21.73	
Fe	XIII	202.044	6.2	161.7	15.41	
Fe	XIII	203.828	6.2	121.4	24.90	
Fe	XIV	264.790	6.3			Missing Pixels
Fe	XV	284.163	6.3			Missing Pixels

Table 6 Intensity Values from EIS Averaged over BP C. Intensity is in ergs / cm^2 / s / steradian. Peak Formation Temperature is Log of Temperature, in Kelvins.

Element	Ion	Wavelength	Peak Form T	Intensity	Error	Notes
Fe	VIII	185.213	5.6	113.9	15.0	
Fe	X	177.240	6.0			Missing Pixels
Fe	X	184.537	6.0	183.5	17.91	
Fe	XI	180.408	6.1	545.2	32.52	
Fe	XI	182.169	6.1	102.2	18.71	
Fe	XII	192.394	6.1	116.6	11.13	
Fe	XII	193.509	6.1	258.9	13.91	
Fe	XII	195.119	6.1	419.2	14.53	
Fe	XIII	201.128	6.2	84.34	12.35	
Fe	XIII	202.044	6.2	138.9	15.19	
Fe	XIII	203.828	6.2	217.5	20.49	
Fe	XIV	264.790	6.3	41.18	10.61	
Fe	XV	284.163	6.3	28.40	12.75	

Table 7 XRT Intensities Averaged over Each BP for Each Filter. Intensities are in Data Number / s / pixel.

FILTER B	A INT	A ERROR	B INT	B ERROR	C INT	C ERROR
Al - Mesh	12.07	1.147	19.413	1.717	22.49	1.844
C - Poly	6.085	0.877	10.91	1.031	11.72	1.238
Ti - Poly	4.164	0.727	7.653	1.092	8.559	0.612
C-Poly/Ti-Poly	1.552	0.195	2.744	0.428	3.514	0.587
Al-Poly/Ti-Poly	0.821	0.204	1.352	0.353	2.444	0.376
Thin Be	0.064	0.096	0.152	0.082	0.302	0.161
Med Be	0.001	0.044	-0.010	0.028	0.004	0.033
Thick Al	0.001	0.024	0.000	0.023	0.006	0.028
Thick Be	0.004	0.015	0.000	0.017	0.000	0.017

Each spectral line detected by EIS has a contribution function which is the sensitivity of that line as a function of temperature. This is obtained from the CHIANTI atomic physics database (Dere et al. 1997) version 6.0.1 (Dere et al. 2009). The contribution functions of the spectral lines used in the analysis of bright point A are shown in figure 13. For each XRT filter there is a response function which is the sensitivity of that filter as a function of temperature. This is taken from the SolarSoft instrument database, and these are shown in figure 14.

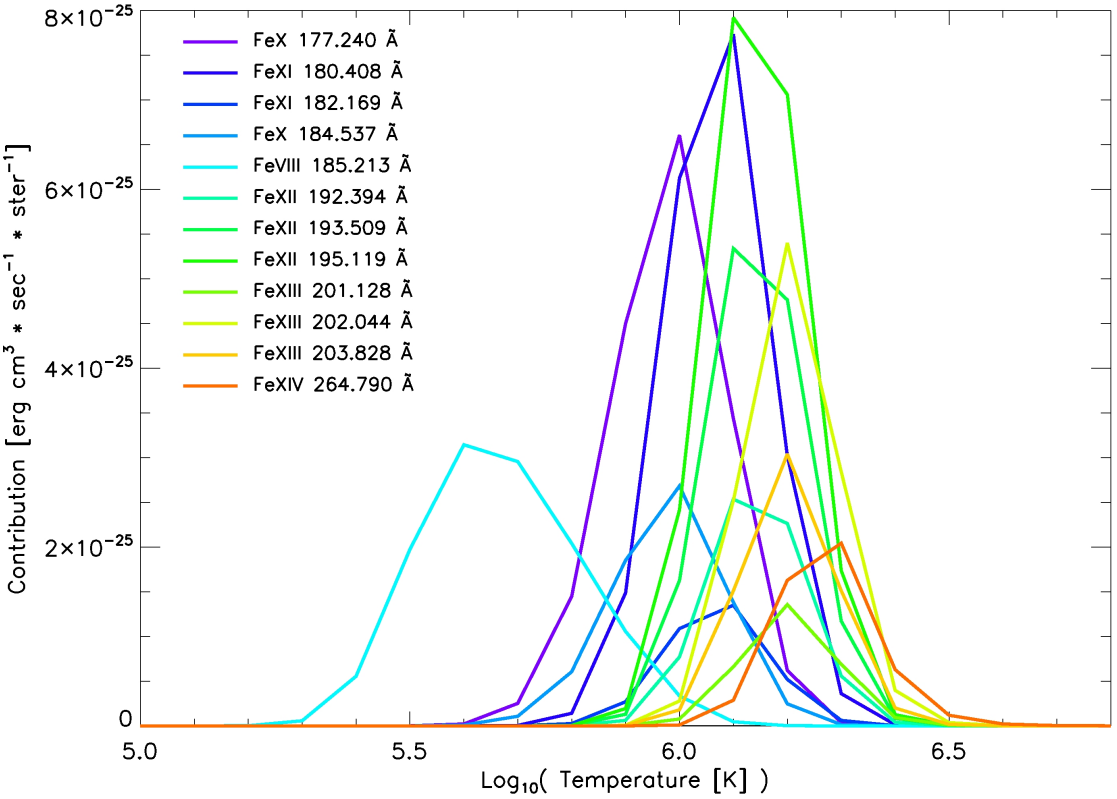

Figure 13 Contribution Functions for the Spectral Lines of BP A. These curves are for the coronal abundances and the Mazotta et al. ion equilibrium, and specific to the electron density calculated for the region inside bright point A.

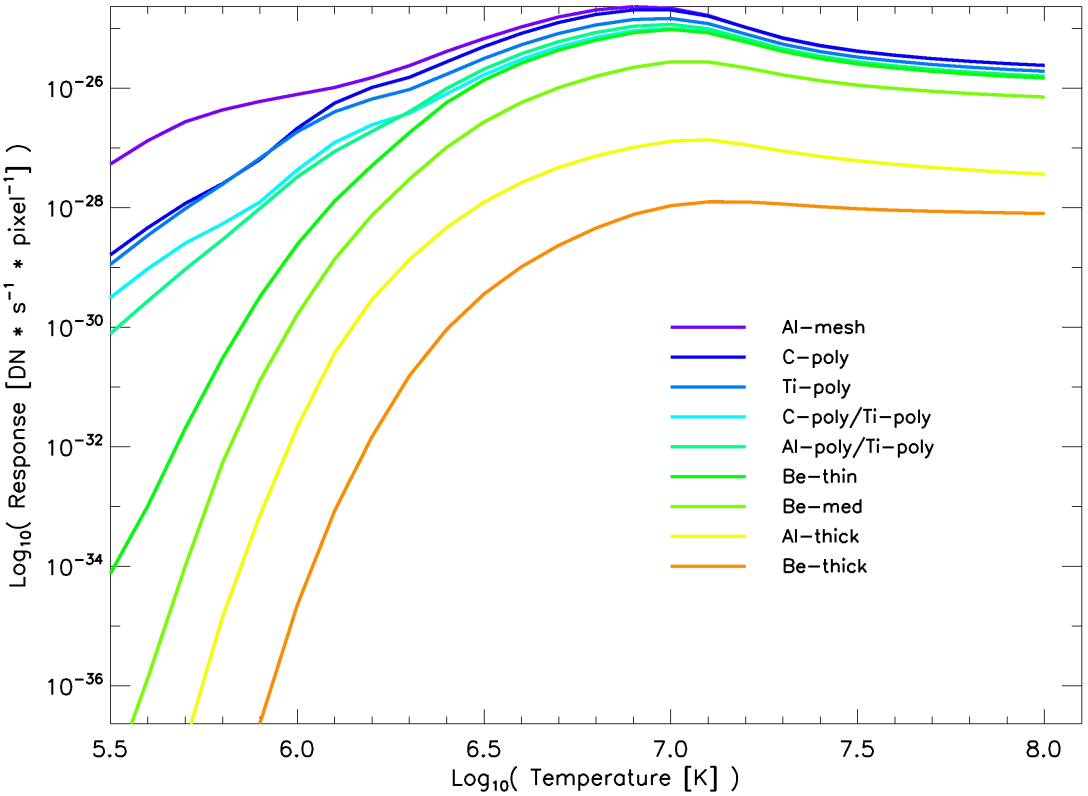

Figure 14 Response Functions for the XRT Filters Used. These curves are specific to the XRT instrument, and assume the coronal abundances, but are the same for all the bright points.

The standard XRT response functions assume CHIANTI coronal abundances (Feldman, 1992) and the ionization balance calculations of Mazzota et al (1998). The contribution functions for the EIS lines were calculated with the same assumptions.

For the electron density, pairs of density sensitive spectral lines obtained from EIS were used. Ratios of their intensities were then used to infer electron density in the emitting plasma. An example of a plot of such a ratio is given in figure 15. These resultant values are then averaged to produce the density used in analysis. A summary of the density data is shown in Table 8.

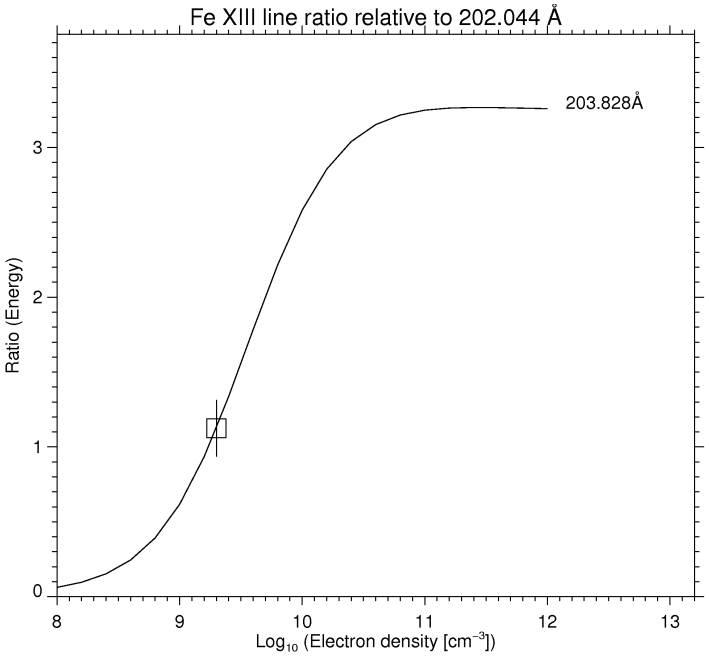

Figure 15 – Line Ratio for Density. Plot of the ratio of intensities of Fe XIII spectral lines 202 and 203 vs electron density.

Table 8 Electron Density Results. From four intensity ratios of density sensitive pairs of lines. Ratio describes which spectral lines are being used in the calculation. Density values for the three bright points are in cm^{-3}.

Ion	Ratio	BP A	BP B	BP C
Fe XIII	203 / 202	2.00E+09	2.09E+09	3.16E+09
	203 / 201	7.94E+08	8.71E+08	1.10E+09
Fe XI	182 / 180	5.28E+08	7.24E+08	1.15E+09
Fe XII	186 / 195	2.10E+08	2.35E+08	4.02E+08
	Mean	8.83E+08	9.80E+08	1.45E+09

If the plasma is isothermal, then the intensity is equal to the instrument sensitivity times the emission measure (EM), the amount of emitting material. For EIS, Intensity $\propto$ G(T) $\times$ EM where G(T) is the contribution function of the spectral line. For XRT, Intensity $\propto$ Resp(T) $\times$ EM, where Resp(T) is the response function for each filter. Solving each of these equations for the emission measure gives us a set of Emission Measure Loci plots, which will give an idea of the comparability of the data. An automated procedure for producing these plots was written in-house, and its code is given in Appendix B. Figure 16 shows a comparison using this technique between the EIS and the XRT data for bright point A.

The intersections of the curves representing different spectral lines or filters in the EM Loci plots show the agreement of the data with the isothermal approximation inherent in the technique. Ideally, if the approximation holds, all the lines would intersect at the same spot. Since this is not the case for these data, we will drop the isothermal approximation and do a full multithermal analysis.

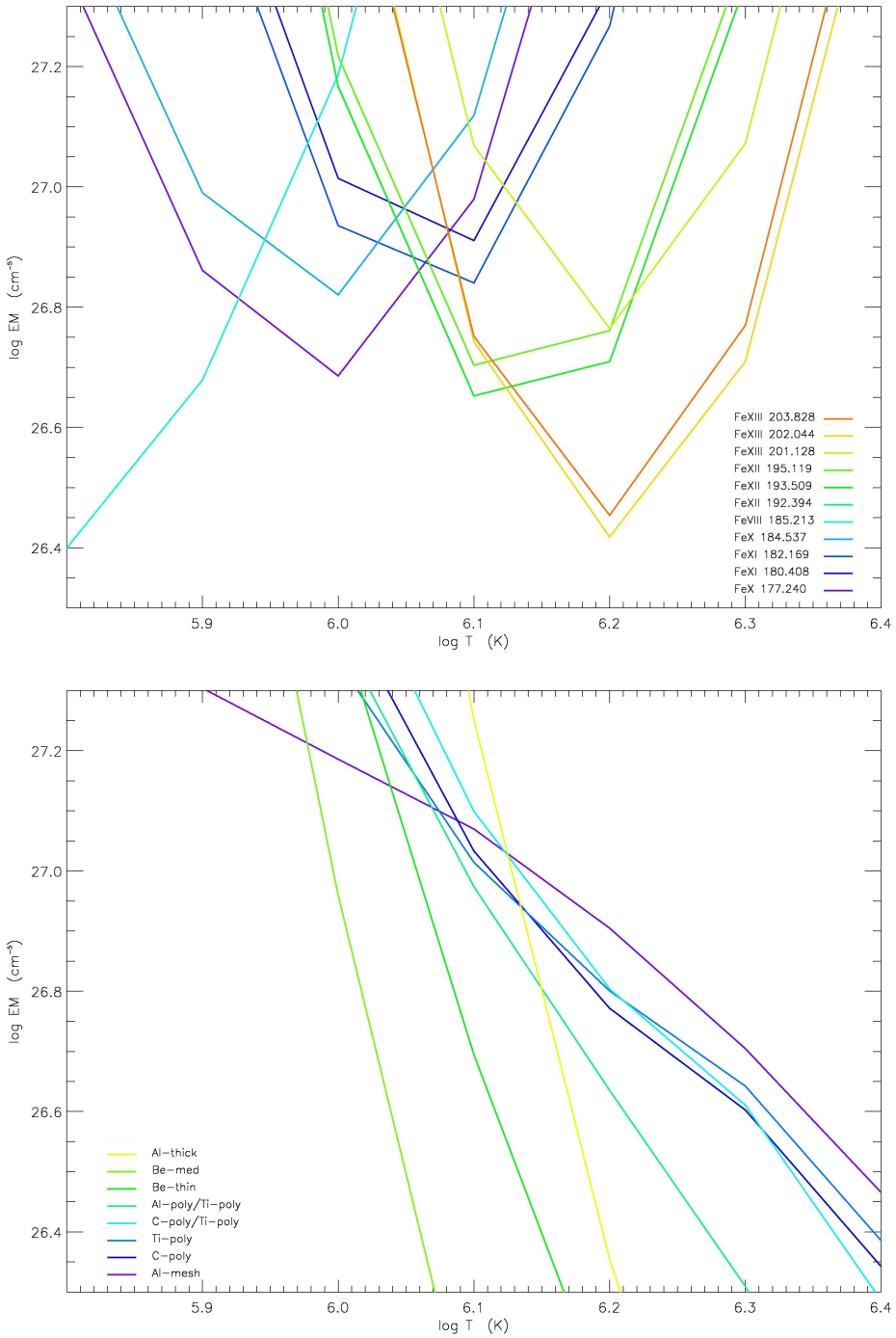

Figure 16 EM Loci Plots of Intensity Data for BP A. Top: EIS. Bottom: XRT. These plots show the shape of the contribution (in the case of EIS) and response (in the case of XRT) functions, adjusted with respect to each other by intensity.

Multithermal plasma requires a Differential Emission Measure (DEM) analysis. This method will begin with the EIS data. For EIS, Intensity $\propto \sum G(T)\, DEM(T)\, \Delta T$, where the DEM is the emission measure for the temperature range ΔT. The DEM distribution is produced from the spectral line data using a forward folding method. A model DEM curve is generated and theoretical intensities are calculated for each spectral line using the equation above. In this manual method of forward folding, a human operator manipulates the DEM curve and observes the results of this manipulation on the error plot. The error plot shows the ratios of the theoretical intensities of the spectral lines and the observed values. Guided by the ratio points, the error bars, and a χ^2 value, this process is continued until a DEM is obtained which gives results consistent with the observed values. A software procedure written in IDL for manipulating these DEM curves and observing the ratio plots in real time was produced in-house, and its code is given in Appendix C.

The EIS data from the three bright points were analyzed in this manner to produce DEM curves characterizing the bright point plasma. The resulting DEM curves and the ratio error plots are shown in figures 17, 18 and 19 for bright points A, B and C respectively.

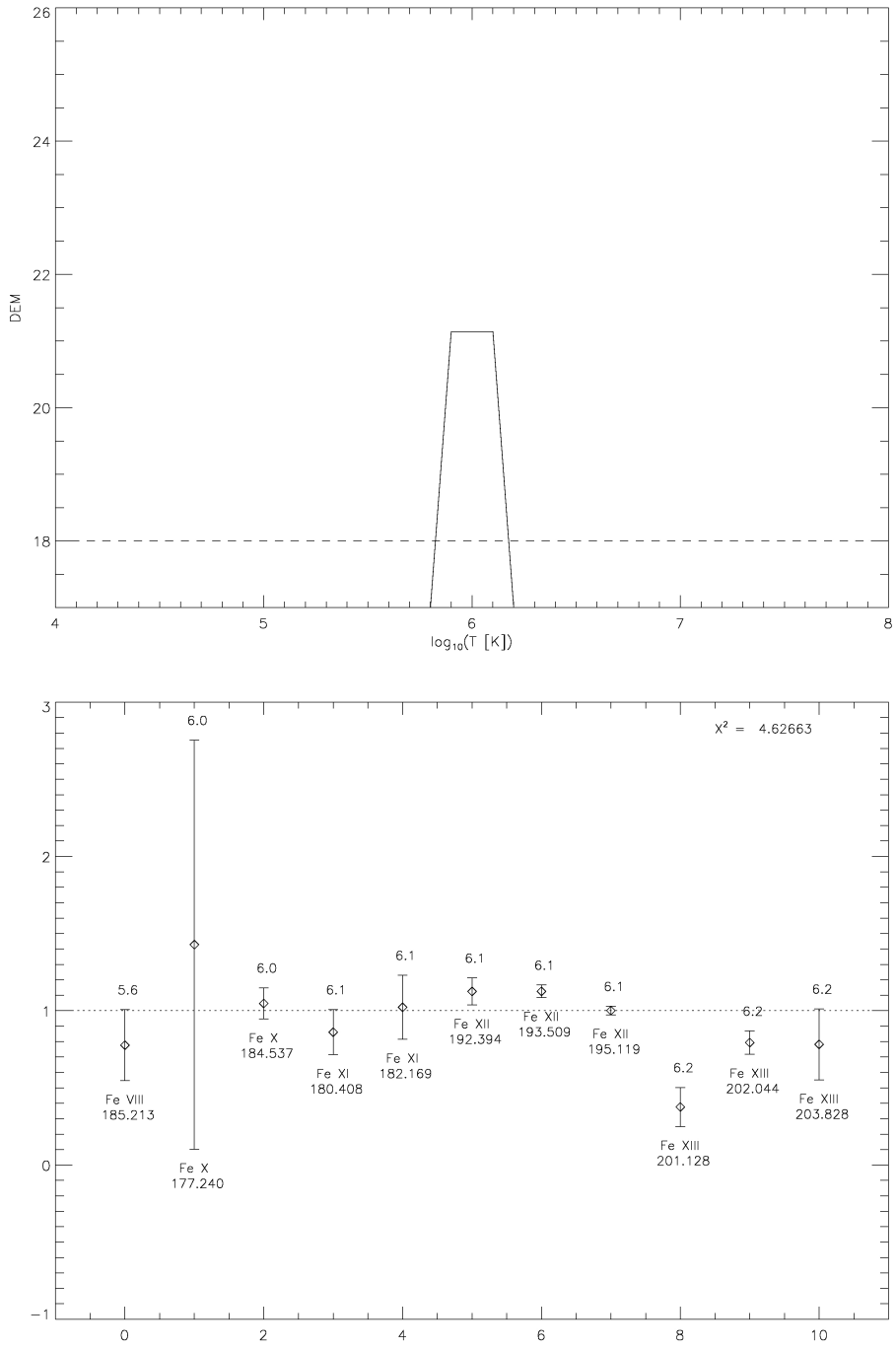

Figure 17 EIS Results for BP A. Top: Differential Emission Measure curve for BP A. Bottom: Error plot for each spectral line for the fit of the DEM to the observed data.

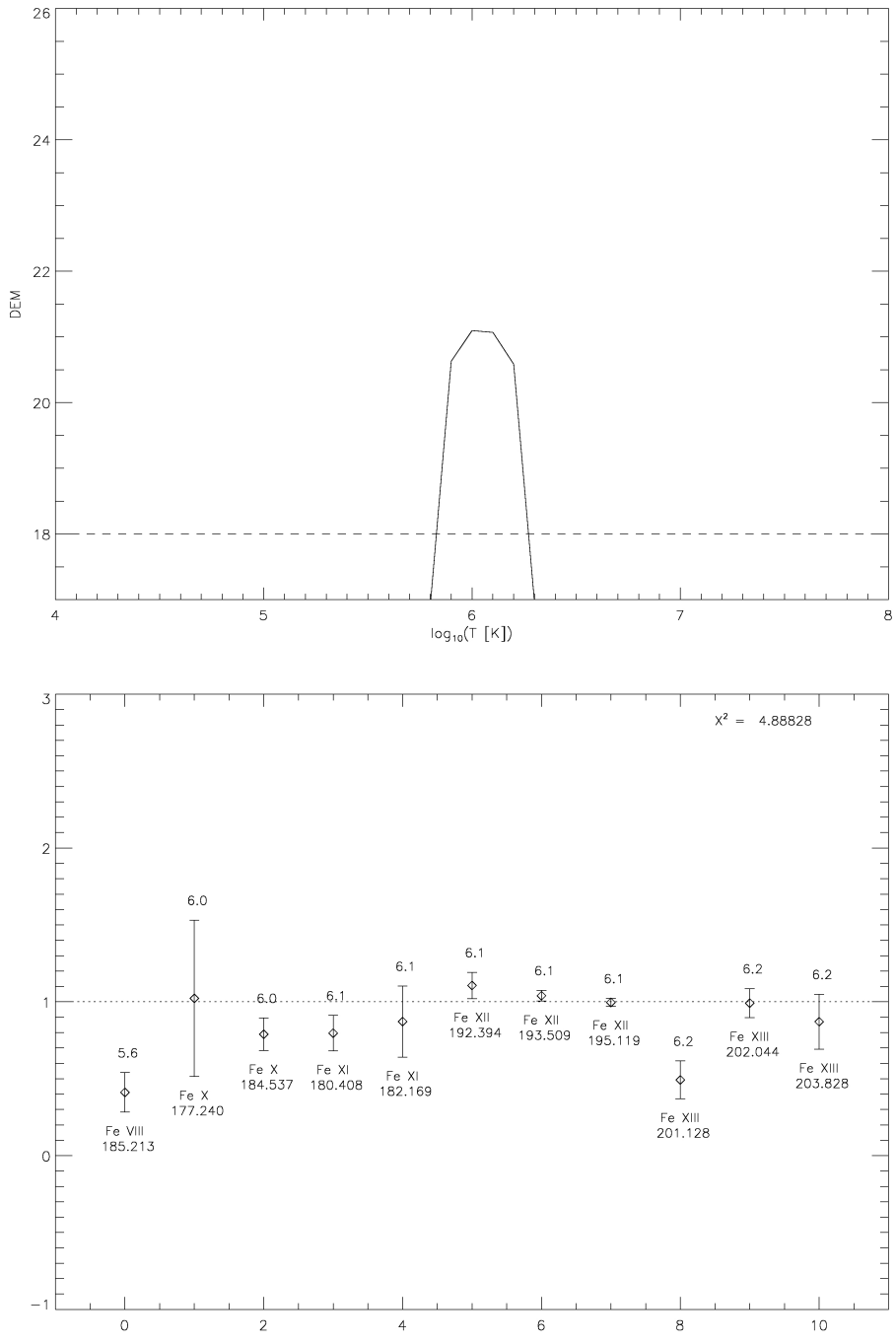

Figure 18 EIS Results for BP B. Top: Differential Emission Measure curve for BP B. Bottom: Error plot for each spectral line for the fit of the DEM to the observed data.

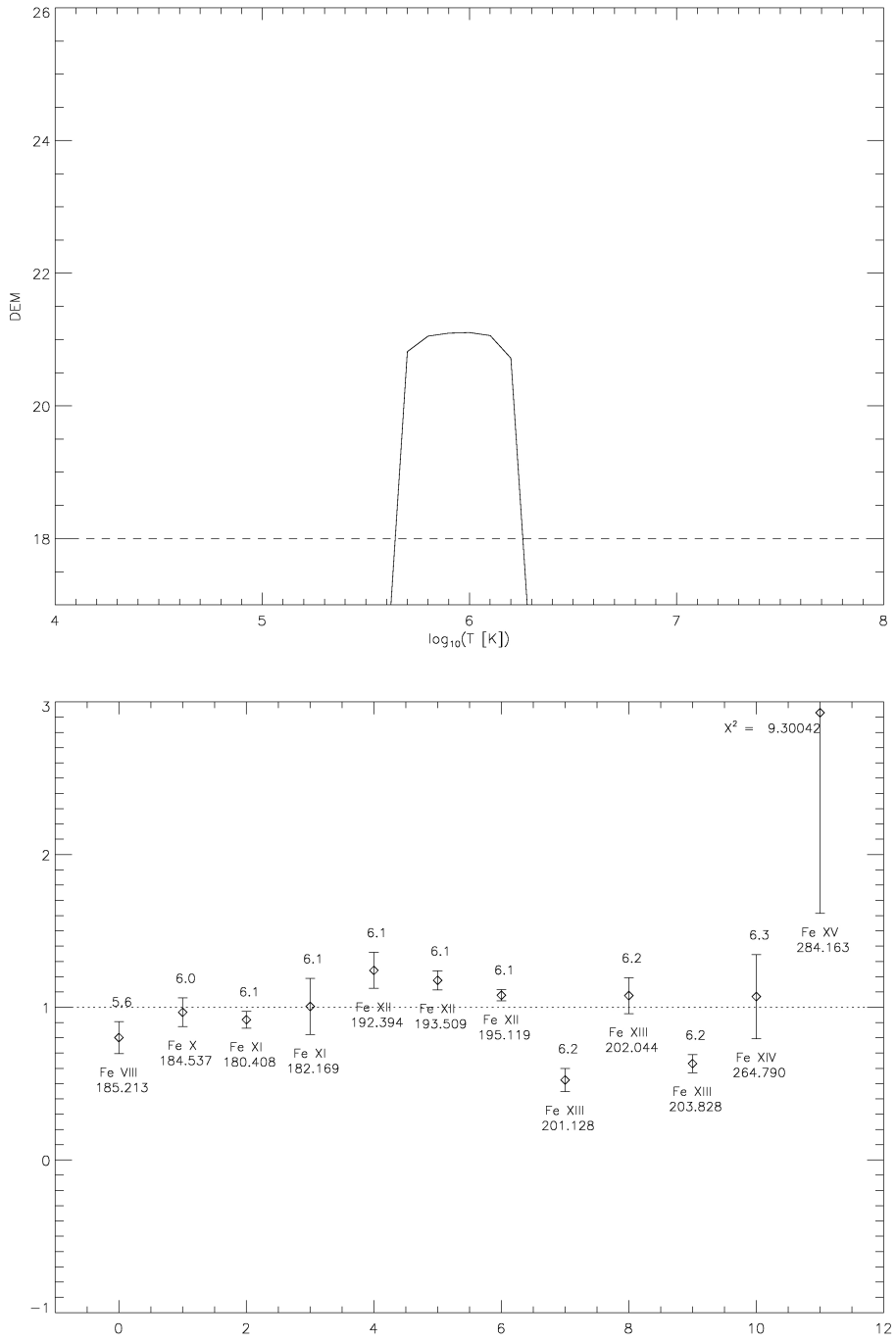

Figure 19 EIS Results for BP C. Top: Differential Emission Measure curve for BP C. Bottom: Error plot for each spectral line for the fit of the DEM to the observed data.

These DEM curves were then used to predict the XRT intensity in each of the bright points. For XRT, Intensity $\propto \sum$ Resp(T) DEM(T) ΔT, where Resp(T) is the response function of the particular filter combination. The result is a set of predicted intensities, one for each XRT filter. These are then divided by the observed intensities to produce the ratio plots shown in figures 20, 21 and 22. The resulting ratios are shown in table 9. Taking a weighted mean for the different filters and bright points gives the instrument cross-calibration factor of .25 +/- .01.

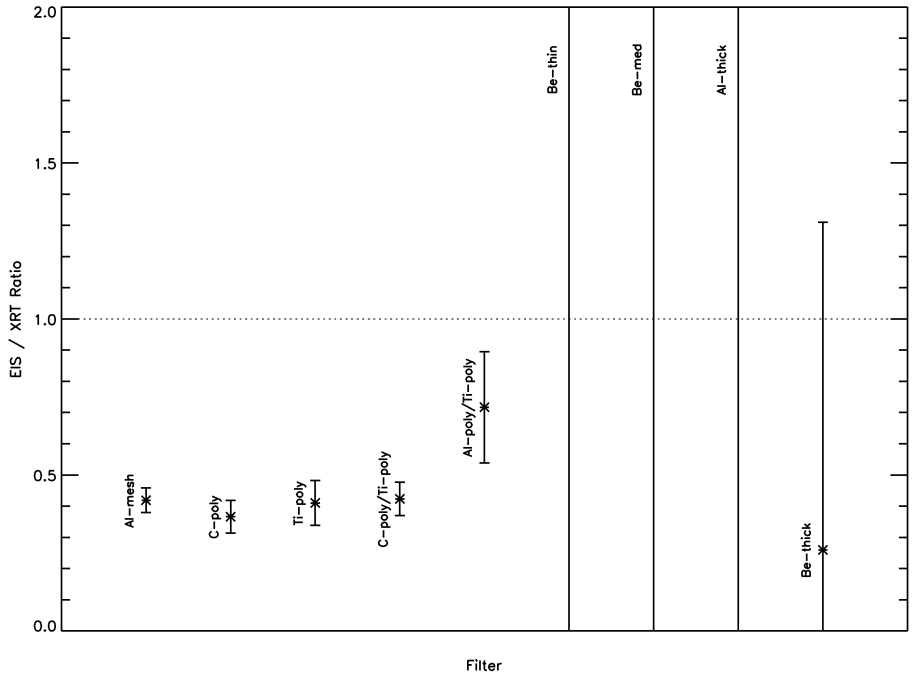

Figure 20 EIS / XRT Ratios for BP A. Plot of the final ratios between the theoretical response of XRT to the data produced by EIS and the observed XRT data. Data shown is for BP A. Values from this and subsequent two plots are used to produce a weighted mean ratio value.

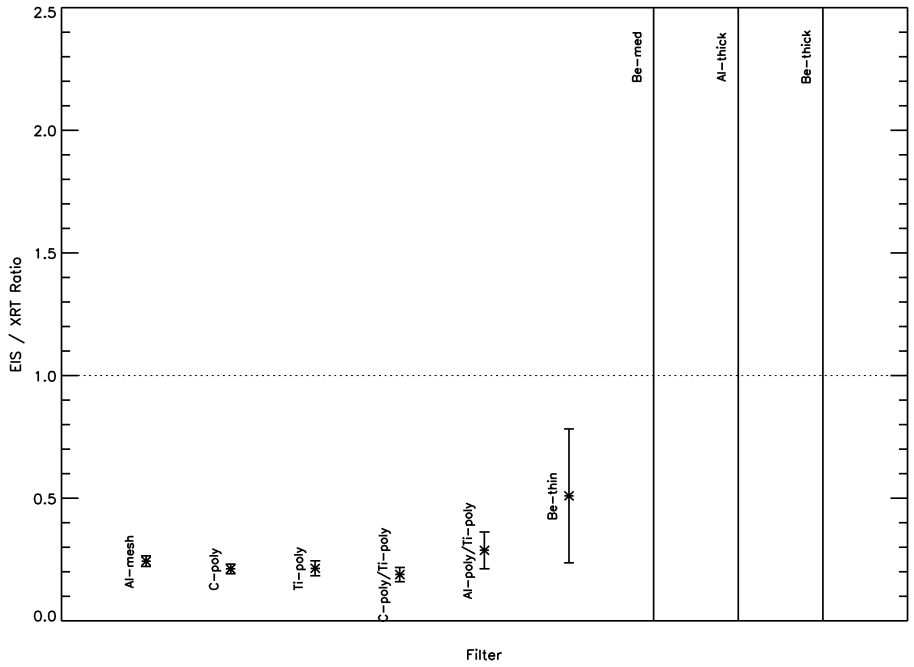

Figure 21 EIS/XRT Ratios for BP B.

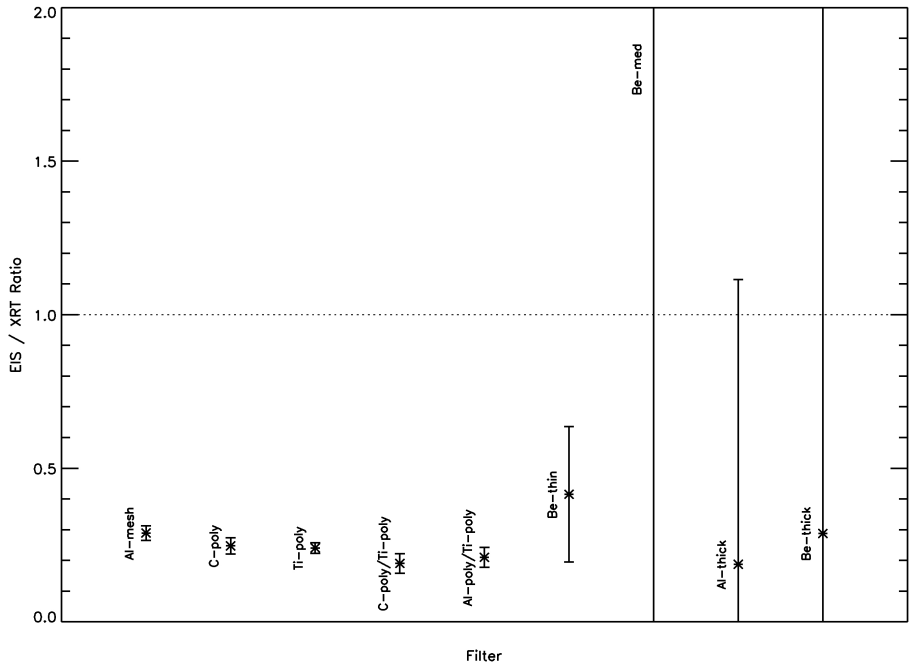

Figure 22 EIS/XRT Ratios for BP C.

Table 9 Final Response Ratios. The response ratios between EIS and XRT, their errors and the final weighted mean and propagated error.

	BP A	BP A ERR	BP B	BP B ERR	BP C	BP C ERR
Al-mesh	0.4189	0.0397	0.2433	0.0215	0.2892	0.0237
C-poly	0.3661	0.0528	0.2117	0.0200	0.2473	0.0261
Ti-poly	0.4103	0.0717	0.2141	0.0306	0.2406	0.0172
C-poly/Ti-poly	0.4232	0.0532	0.1888	0.0295	0.1904	0.0318
Al-poly/Ti-poly	0.7171	0.1784	0.2876	0.0752	0.2100	0.0323
Be-thin	5.0203	7.5107	0.5096	0.2734	0.4152	0.2210
Be-med	80.0726	2.351E+3	1.097E+6	3.047E+12	6.7478	61.0535
Al-thick	9.2066	332.7653	4.076E+4	9.224E+10	0.1874	0.9272
Be-thick	0.2597	1.0508	1.115E+3	1.927E+9	0.2874	14.4654
	Weighted Mean	.25		Error	.01	

This value of .25 represents the EIS-XRT instrument cross-calibration factor, the goal of the present study. Thus, in order to use XRT and EIS data together, the observed XRT intensities must be multiplied by a factor of .25, or conversely, the EIS intensities must by multiplied by a factor of 4.0.

This allows XRT and EIS to be used concurrently on the same data set, which is an obvious goal of the *Hinode* mission, as the instruments are in many ways complimentary. This is the first time an XRT-EIS cross-calibration factor has been found using x-ray bright points. Now that such a procedure is established, and with the tools available, it should not be difficult to duplicate the procedure for other bright point data sets and attempt to verify this result. A detailed examination of table 9 will show that the results for bright point A are about a factor of two different from those of bright points B and C. The most plausible explanation for this is that there was a small flare in bright

point A during the observations. Future work will add more simultaneously observed bright points to the cross-calibration analysis to determine the validity of this explanation.

REFERENCES

Culhane, J.L., Harra, L.K., James, A.M., Al-Janabi, K., Bradley, L.J., Chaudry, R.A., *et al.*: 2007, *Solar Phys.* **243**, 19-61.

Feldman, U: 1992, *Phys. Scr.*, **46**, 202

Golub, L., DeLuca, E., Austin, G., Bookbinder, J., Caldwell, D., Cheimets, P., *et al.*: 2007, *Solar Phys.* **243**, 63-83.

Golub, L., Harvey, K. L., Herant, M., Webb, D. F.: 1989, *Solar Physics.* **124**, 211-217.

Golub and Pasachoff. *Nearest Star: The Surprising Science of Our Sun.* Cambridge, Massachusetts: Harvard University Press, 2001.

Golub and Pasachoff. *The Solar Corona*. Cambridge, Massachusetts: Cambridge University Press, 1997.

Harvey, Karen L.: 1985, *Ast. J. Phys*. **38**, 875-83.

Kariyappa, R., Deluca, E. E., Saar, S. H., Golub, L., Damé, L., Pevtsov, A. A., Varghese, B. A.: 2011, *Astronomy and Astrophysics.* **526**, id.A78.

Kosugi, T., Matsuzaki, K., Sakao, T.,Shimizu, T., Sone, Y., Tachikawa, S., *et al.*: 2007, *Solar Phys.* **243**, 3-17.

Kutner, Marc L. *Astronomy: A Physical Perspective.* 2nd Ed. Cambridge, Massachusetts Cambridge University Press, 2003.

Longcope, D. W., Kankelborg, C. C., Nelson, J. L., Pevtsov, A. A.: 2001, *The Astrophysical Journal.* **553**, Issue 1, 429-439.

Mazzotta, P., Mazzitelli, G., Colafrancesco, S., Vittorio, N.: 1998, *Astronomy and Astrophysics Supplement.* **133**, 403-409.

NAOJ: National Astronomical Observatory of Japan. *SOT: Solar Optical Telescope - Optics*. <http://solar-b.nao.ac.jp/sot_e/optics_e.shtml>: retrieved 05/14/2010.

NASA: National Aeronautics and Space Administration. *Hinode: Mission to the Sun: Scientific and Technical Info.* <http://www.nasa.gov/mission_pages/hinode/instruments.html#spectrometer>: retrieved 5/14/2010.

Phillips, Kenneth J. H. *Guide to the Sun.* Cambridge, Massachusetts: Cambridge University Press, 1995.

Saar, S., Farid, S., Deluca, E.: 2009, *Cool Stars, Stellar Systems and the Sun.* Proceedings of the 15th Cambridge Workshop on Cool Stars, Stellar Systems and the Sun. AIP Conference Proceedings. **1094**, 756-759.

Seeds, Michael A. *Foundations of Astronomy.* 8th Ed. Toronto, Ontario: Thomson Brooks/Cole, 2005.

SolarSoft Software Library, The (SSW). <http://www.lmsal.com/solarsoft/>: retrieved 4/1/2010.

Strong, Keith T., Harvey, Karen, Hirayama, Tadashi, Nitta, Nariaki, Shimizu, Toshifumi, Tsuneta, Saku: 1992, *Astron. Soc. Japan.* **44**, L161-L166.

TCD SolarMonitor.org team. *SolarMonitor.org*. <http://www.solarmonitor.org/index.php?date=20101219&indexnum=1>: retrieved 2/11/2011.

Appendix A
Pixel_Picker: Coalignment and Selection Software

```
;Written for IDL using widgets for the X display
;PIXEL_PICKER INSTRUCTIONS

;You'll need to build a multidimensional image array.  Array needs to be of the form  [x,
      y, n], where x is x coordinates, y is y coordinates, and n is the number of images.

;Names array is a list of the names of the filters or spectral lines in which the images are
      taken.  It can be any identifier, but its numbering must match the numbering n of
      the image array.  Just build the arrays in order; if you mix up an image and its
      name, the program will believe you.

;     img = [ [[imga]], [[imgb]], [[imgc]], [[imgd]], [[imge]], [[imgf]] ]
;     Or whatever your image arrays are called

;     names = ['94','131','171','193','211','335']
;     Or whatever your images are

;     IDL> help, names
;     You should get these
;     NAMES          STRING    = Array[6]
;     IDL> help, img
;     IMG            FLOAT     = Array[200, 200, 6]

;     a = pixel_picker(img, names = names)
;Calling sequence

;Names could also be an INT or FLOAT array.  It shouldn't really care.  Names is optional
      and can be left out:

;     a = pixel_picker(img)
;Calling sequence without names array

;After you click 'DONE', the output, a, will be of the form:

;     ** Structure <29c4204>, 8 tags, length=960256, data length=960256, refs=1:
;       SUBIMG         FLOAT     Array[200, 200, 6]
;       RANGEX         LONG      Array[2]
;       RANGEY         LONG      Array[2]
;       OFFSET         LONG      Array[2, 6]
;       SELECTED_ABS   LONG      Array[2, 3]
;       SELECTED_SUB   LONG      Array[2, 3]
```

```
;         INTENSITIES     FLOAT     Array[3, 6]
;         NAMES           STRING    Array[6

;A.SUBIMG will be the coaligned, zoomed images that were current when 'DONE' was
      clicked.

;A.RANGEX and A.RANGEY will be the coordinate ranges, in terms of absolute
      coordinates in the original image array, that the output subimages cover.

;A.OFFSET is the value of the offset for each image.  [c, n] where c=0 => x, c=1 => y.
      To get the absolute coordinates of a coordinate of a pixel in a subarray, you would
      add the minimum range value and subtract the offset value.  (Images moved 1 pixel
      to the right now reflect x coordinates of -1.)

;A.SELECTED_ABS is the set of coordinates of selected pixels in terms of the original
      image array.  It is of the form [c, m], where m is the number of the selected pixel,
      and c is the x-y coordinates.  It is not specified for each image, and as such, you
      would have to subtract the offset of that image to get the coordinates of the selected
      pixel for that image.

;A.SELECTED_SUB is the set of coordinates of selected pixels in terms of the returned
      subarray.  They are already offset, and so they represent the actual coordinates
      (OFFSET already included) of each selected pixel in the returned subarray.

;A.INTENSITIES is the set of intensity values for the selected pixels.  They are in the
      form [m, n], where m is the number of the selected pixel, and n is the number of the
      image.  A.INTENSITIES[1, 3] is the intensity value of the second selected pixel in
      the fourth image.

;A.NAMES is the same array of names you may or may not have given PIXEL_PICKER
      when you started it.  It is here as a reminder of which image is which.

; end of comments

FUNCTION IMGevent, ev, m    ;

cursor, x, y, /nowait, /data
x = long(x + .499 )
y = long(y +.499 )

if m.zoomed eq 1 then begin
  if x gt m.zoomx[1] - m.zoomx[0]  then x = m.zoomx[1] - m.zoomx[0] ;+
       m.xoffset[m.index]
  if y gt m.zoomy[1] - m.zoomy[0]  then y = m.zoomy[1] - m.zoomy[0] ;+
```

```
        m.yoffset[m.index]
  if x lt 0 + m.xoffset[m.index] then x = 0
  if y lt 0 + m.yoffset[m.index] then y  = 0
endif
if m.zoomed eq 0 then begin
  if x gt n_elements(m.datai[*,0]) + m.xoffset[m.index] - 1 then x =
        n_elements(m.datai[*,0]) + m.xoffset[m.index] - 1
  if y gt n_elements(m.datai[0,*]) + m.yoffset[m.index] - 1 then y =
        n_elements(m.datai[0,*]) + m.yoffset[m.index] - 1
  if x lt 0 + m.xoffset[m.index] then x = 0 + m.xoffset[m.index]
  if y lt 0 + m.yoffset[m.index] then y  = 0 + m.yoffset[m.index]
endif

xcoord = widget_info(ev.top, find_by_uname = 'XCOORD')
ycoord = widget_info(ev.top, find_by_uname = 'YCOORD')
xabs = widget_info(ev.top, find_by_uname = 'XABS')
yabs = widget_info(ev.top, find_by_uname = 'YABS')
int = widget_info(ev.top, find_by_uname = 'INT')
widget_control, xcoord, set_value = strcompress(x)
widget_control, ycoord, set_value = strcompress(y)
xa = x + m.zoomx[0];
ya = y + m.zoomy[0];
xao = xa - m.xoffset[m.index]
yao = ya - m.yoffset[m.index]
widget_control, xabs, set_value = strcompress(xao)
widget_control, yabs, set_value = strcompress(yao)
widget_control, int, set_value = strcompress(m.datai[xao,yao])
zxmin=widget_info(ev.top, find_by_uname = 'ZOOMXMIN')
zymin=widget_info(ev.top, find_by_uname = 'ZOOMYMIN')
zxmax=widget_info(ev.top, find_by_uname = 'ZOOMXMAX')
zymax=widget_info(ev.top, find_by_uname = 'ZOOMYMAX')
widget_control, zxmin, set_value = strcompress(m.zoomx[0])
widget_control, zxmax, set_value = strcompress(m.zoomx[1])
widget_control, zymin, set_value = strcompress(m.zoomy[0])
widget_control, zymax, set_value = strcompress(m.zoomy[1])

if m.mode eq 0 then begin
  if ev.clicks eq 1 then begin
    plots, x, y, psym = 1, symsize = 7
    m.storex1 = xa
    m.storey1 = ya
  endif
  if ev.release eq 1 then begin
    m.doplot = 1
```

```
    if xa eq m.storex1 or ya eq m.storey1 then goto, nozoom
    if m.storex1 lt xa then m.zoomx = [m.storex1, xa] else m.zoomx = [xa, m.storex1]
    if m.storey1 lt ya then m.zoomy = [m.storey1, ya] else m.zoomy = [ya, m.storey1]
    m.zoomed = 1
   endif
   NOZOOM:
endif

if m.mode eq 1 then if ev.clicks eq 1 then begin
;  if m.zoomed eq 1 then begin
;    x = x + m.zoomx[0]
;    y = y + m.zoomy[0]
;  endif
  if m.map[xa, ya] eq 0 then m.map[xa,ya] = 1 else m.map[xa,ya] = 0
  if m.map[xa, ya] eq 0 then msg = ' unselected.' else msg = ' selected.'
  print, '  (',strcompress(x),', ',strcompress(y),')  ABSOLUTE (', strcompress(xa),', ',
        strcompress(ya), ')   INTENSITY(', strcompress(m.names[m.index]) ,') = ',
        strcompress(m.datai[x+m.zoomx[0]-m.xoffset[m.index],y+m.zoomy[0]-
        m.yoffset[m.index]]), msg
  m.doplot = 1
endif

if m.mode eq 2 then begin
  if ev.press gt 0 then begin
    if ev.press eq 4 then begin
      m.index = m.index + 1
      if m.index gt m.number then m.index = 0
      m.datai = m.data[*,*,m.index]
      label = widget_info(ev.top, find_by_uname = 'LAYERlabel')
      widget_control, label, set_value = strcompress(m.index)
      m.doplot = 1
      xoffset = widget_info(ev.top, find_by_uname = 'OFFX')
      widget_control, xoffset, set_value = strcompress(m.xoffset[m.index])
      yoffset = widget_info(ev.top, find_by_uname = 'OFFY')
      widget_control, yoffset, set_value = strcompress(m.yoffset[m.index])
    endif else begin
      m.index = m.index - 1
      if m.index lt 0 then m.index = m.number
      m.datai = m.data[*,*,m.index]
      label = widget_info(ev.top, find_by_uname = 'LAYERlabel')
      widget_control, label, set_value = strcompress(m.index)
      m.doplot = 1
      xoffset = widget_info(ev.top, find_by_uname = 'OFFX')
      widget_control, xoffset, set_value = strcompress(m.xoffset[m.index])
```

```
    yoffset = widget_info(ev.top, find_by_uname = 'OFFY')
    widget_control, yoffset, set_value = strcompress(m.yoffset[m.index])
   endelse
  endif
endif

return, m

END



PRO pixel_picker_event, ev
COMMON PPBlock, output

widget_control, ev.top, get_uvalue = m
widget_control, ev.id, get_uvalue = uval
mode = widget_info(ev.top, find_by_uname = 'MODElist')
widget_control, mode, get_value = mode
m.mode = mode
m.doplot = 0

doquit = 0

CASE uval of
  'QUIT' : DOQUIT = 1
  'IMGplot' : m = IMGevent(ev, m)
  'ZOUT' : begin
     m.zoomed = 0
     m.doplot = 1
     m.zoomx = [0, n_elements(m.datai[*,0]) - 1]
     m.zoomy = [0, n_elements(m.datai[0,*]) - 1]
   endcase
  'NEXT' : begin
     m.index = m.index + 1
     if m.index gt m.number then m.index = 0
     m.datai = m.data[*,*,m.index]
     label = widget_info(ev.top, find_by_uname = 'LAYERlabel')
     widget_control, label, set_value = strcompress(m.index)
     m.doplot = 1
     xoffset = widget_info(ev.top, find_by_uname = 'OFFX')
     widget_control, xoffset, set_value = strcompress(m.xoffset[m.index])
     yoffset = widget_info(ev.top, find_by_uname = 'OFFY')
```

```
    widget_control, yoffset, set_value = strcompress(m.yoffset[m.index])
  endcase
'PREV' : begin
    m.index = m.index - 1
    if m.index lt 0 then m.index = m.number
    m.datai = m.data[*,*,m.index]
    label = widget_info(ev.top, find_by_uname = 'LAYERlabel')
    widget_control, label, set_value = strcompress(m.index)
    m.doplot = 1
    xoffset = widget_info(ev.top, find_by_uname = 'OFFX')
    widget_control, xoffset, set_value = strcompress(m.xoffset[m.index])
    yoffset = widget_info(ev.top, find_by_uname = 'OFFY')
    widget_control, yoffset, set_value = strcompress(m.yoffset[m.index])
  endcase
'LEFT' : begin
    m.xoffset[m.index] = m.xoffset[m.index] - 1
    OFFX = widget_info(ev.top, find_by_uname = 'OFFX')
    widget_control, OFFX, set_value = strcompress(m.xoffset[m.index])
    m.doplot = 1
  endcase
'RIGHT' : begin
    m.xoffset[m.index] = m.xoffset[m.index] + 1
    OFFX = widget_info(ev.top, find_by_uname = 'OFFX')
    widget_control, OFFX, set_value = strcompress(m.xoffset[m.index])
    m.doplot = 1
  endcase
'UP' : begin
    m.yoffset[m.index] = m.yoffset[m.index] + 1
    OFFY = widget_info(ev.top, find_by_uname = 'OFFY')
    widget_control, OFFY, set_value = strcompress(m.yoffset[m.index])
    m.doplot = 1
  endcase
'DOWN' : begin
    m.yoffset[m.index] = m.yoffset[m.index] - 1
    OFFY = widget_info(ev.top, find_by_uname = 'OFFY')
    widget_control, OFFY, set_value = strcompress(m.yoffset[m.index])
    m.doplot = 1
  endcase
'COLORlist' : begin
    widget_control, ev.id, get_value = color
    if color eq 2 then color = 3
    m.color = color
    m.doplot = 1
  endcase
```

```
  'CONTRAST' : begin
     widget_control, ev.id, get_value = contrast
     m.contrast = 1 - (.01 * contrast)
     m.doplot = 1
    endcase
  ELSE :
endcase

if m.doplot eq 1 then begin
  img = shift(m.datai, m.xoffset[m.index], m.yoffset[m.index])
  minimg = min(m.datai)
  if m.yoffset[m.index] ne 0 then for i = 0, abs(m.yoffset[m.index]) - 1 do begin
    if m.yoffset[m.index] lt 0 then img[*, n_elements(img[0, *]) - 1 - i] = minimg else
        img[*, 0 + i] = minimg
  endfor
  if m.xoffset[m.index] ne 0 then for i = 0, abs(m.xoffset[m.index]) - 1 do begin
    if m.xoffset[m.index] lt 0 then img[n_elements(img[*, 0]) - 1 - i, *] = minimg else
        img[0 + i, *] = minimg
  endfor
  if m.zoomed eq 1 then begin
    sz = 1000 / ( 7 *
        n_elements(img[m.zoomx[0]:m.zoomx[1],0])>n_elements(img[0,m.zoomy[0]:m.zo
        omy[1]]) )        ; This is about the best I can do - don't know of a way to make this
        more consistent
    loadct, m.color, /silent
    plot_image, img[m.zoomx[0]:m.zoomx[1], m.zoomy[0]:m.zoomy[1]]^m.contrast,
        /isotropic, title = strcompress(m.index) + ' - ' + strcompress(m.names[m.index])
    if m.color eq 0 or m.color eq 1 then loadct, 11, /silent else loadct, 24, /silent
    if m.color eq 0 or m.color eq 1 then cval = 255 else cval = 25
    indices = where(m.map[m.zoomx[0]:m.zoomx[1], m.zoomy[0]:m.zoomy[1]] eq 1,
        count)
    if count ne 0 then begin
      arind = array_indices(m.map[m.zoomx[0]:m.zoomx[1], m.zoomy[0]:m.zoomy[1]],
        indices)
      arind[0,*] = arind[0, *]; + m.xoffset[m.index]
      arind[1,*] = arind[1, *]; + m.yoffset[m.index]
      plots, arind, psym = 6, symsize = sz, color = cval
    endif
    loadct, m.color, /silent
  endif
  if m.zoomed eq 0 then begin
    sz = 1000 / ( 7 * n_elements(img[*,0])>n_elements(img[0,*]) )
    loadct, m.color, /silent
    plot_image, img^m.contrast, /isotropic, title = strcompress(m.index) + ' - ' +
```

```
        strcompress(m.names[m.index])
    if m.color eq 0 or m.color eq 1 then loadct, 11, /silent else loadct, 24, /silent
    if m.color eq 0 or m.color eq 1 then cval = 255 else cval = 25
    indices = where(m.map eq 1, count)
    if count ne 0 then begin
      arind = array_indices(m.map, indices)
      arind[0,*] = arind[0, *]; + m.xoffset[m.index]
      arind[1,*] = arind[1, *]; + m.yoffset[m.index]
      plots, arind, psym = 6, symsize = sz, color = cval
    endif
    loadct, m.color, /silent
  endif
endif

widget_control, ev.top, set_uvalue = m

if DOQUIT eq 1 then begin
  subimg = 0
  rangex = 0
  rangey = 0
  offset = 0
  intensities = 'na'

  rangex = m.zoomx
  rangey = m.zoomy

  selected = where(m.map eq 1, count)
  if count ne 0 then selected_absolute = array_indices(m.map, selected) else
        selected_absolute = 'NONE'
  selected_abs = selected_absolute
  for i = 0, m.number  do begin
    img = shift(m.data[*,*,i], m.xoffset[i], m.yoffset[i])
    minimg = min(m.data[*,*,i])
    if m.yoffset[i] ne 0 then for j = 0, abs(m.yoffset[i]) - 1 do begin
      if m.yoffset[i] lt 0 then img[*, n_elements(img[0, *]) - 1 - j] = minimg else img[*, 0 +
        j] = minimg
    endfor
    if m.xoffset[i] ne 0 then for j = 0, abs(m.xoffset[i]) - 1 do begin
      if m.xoffset[i] lt 0 then img[n_elements(img[*, 0]) - 1 - j, *] = minimg else img[0 + j,
        *] = minimg
    endfor
    if m.zoomed eq 1 then begin
      subimgi = img[m.zoomx[0]:m.zoomx[1], m.zoomy[0]:m.zoomy[1]]
      rangexi = m.zoomx
```

```
   rangeyi = m.zoomy
  endif else begin
   subimgi = img
   rangexi = [0, n_elements(img[*, 0]) - 1]
   rangeyi = [0, n_elements(img[0, *]) - 1]
  endelse
  if n_elements(subimg) eq 1 then subimg = subimgi else subimg = [ [[subimg]],
      [[subimgi]] ]
;  if n_elements(rangex) eq 1 then rangex = rangexi else rangex = [ [rangex], [rangexi] ]
;  if n_elements(rangey) eq 1 then rangey = rangeyi else rangey = [ [rangey], [rangeyi] ]
  if n_elements(offset) eq 1 then offset = [ m.xoffset[i], m.yoffset[i] ] else offset =
      [ [offset], [m.xoffset[i], m.yoffset[i]] ]

  if count ne 0 then for j = 0, count - 1 do begin
   xa = selected_absolute[0,j] - m.xoffset[i]
   ya = selected_absolute[1,j] - m.yoffset[i]
   selected_abs[0,j] = xa
   selected_abs[1,j] = ya
   if j eq 0 then intensity = m.data[xa, ya, i] else intensity = [ intensity, m.data[xa, ya,
      i ] ]
  endfor
  if count ne 0 then begin
   if i eq 0 then intensities = intensity else intensities = [  [intensities] ,  [intensity]  ]
  endif

endfor

selected_sub = selected_absolute
if count ne 0 then begin
  selected_sub[0, *] = selected_sub[0,*] - m.zoomx[0]
  selected_sub[1, *] = selected_sub[1,*] - m.zoomy[0]
endif

;selected_sub = 0
;if m.zoomed eq 1 and count ne 0 then begin
;  selected = where(m.map[m.zoomx[0]:m.zoomx[1], m.zoomy[0]:m.zoomy[1]] eq 1,
      count)
;  ind = array_indices(m.map[m.zoomx[0]:m.zoomx[1], m.zoomy[0]:m.zoomy[1]],
      selected)
;  for i = 0, count - 1 do begin
;    selected_subi = ind[*,i]
;    if n_elements(selected_sub) eq 1 then selected_sub = selected_subi else selected_sub
      = [ [selected_sub], [selected_subi] ]
;    for j = 0, m.number - 1 do if n_elements(intensitiesi) eq 1 then intensitiesi =
```

```
        m.data[m.zoomx[0]:m.zoomx[1], m.zoomy[0]:m.zoomy[1], j]
  ;  endfor
  ;endif else selected_sub = selected_absolute

  output = {subimg:subimg, rangex:rangex, rangey:rangey, offset:offset,
        selected_abs:selected_absolute, selected_sub:selected_sub, intensities:intensities,
        names:m.names}
  widget_control, ev.top, /destroy
endif

END



FUNCTION pixel_picker, data, names = names

COMMON PPblock, output

dim = size(data)

if dim[0] lt 2 or dim[0] gt 3 then begin
  print, 'ERROR - Array must be 2 or 3 dimensional. [X, Y] or [X, Y, N]'
  return, 0
endif

if dim[0] eq 2 then number = 0
if dim[0] eq 3 then number = dim[3] - 1

loadct, 0, /silent

PPbase = widget_base(Title = 'Pixel Picker', column = 2)
CONTROLbase = widget_base(PPbase, /column)  ; , /kbrd_focus_events
DISPLAYbase = widget_base(PPbase, /column)

IMGplot = widget_draw(DISPLAYbase, xsize = 1000, ysize = 1000, uvalue = 'IMGplot',
        uname = 'IMGplot', /button_events, /motion_events)

MODElist = cw_bgroup(CONTROLbase, ['ZOOM', 'SELECT', 'STEP'], uname =
        'MODElist', uvalue = 'MODElist', set_value = 0, /exclusive)

DISPbase = widget_base(CONTROLbase, frame = 3, /column)
COLORgroup = cw_bgroup(DISPbase, ['GREY', 'BLUE', 'RED'], uname = 'COLORlist',
        uvalue = 'COLORlist', set_value = 0, label_top = 'COLOR', /exclusive, /row)
CONTRASTslider = widget_slider(DISPbase, title = 'BRIGHT CORRECTION', value =
```

```
    0, uvalue = 'CONTRAST', uname = 'CONTRAST', minimum = -50, maximum =
    99, /align_center)


ZOOMbase = widget_base(CONTROLbase, frame = 3, /column)
ZOOMlabel = widget_label(ZOOMbase, value = 'ZOOMED:', /align_center)
ZOOMbase2 = widget_base(ZOOMbase, row = 2)
ZOOMXmindisp = cw_field(ZOOMbase2, title = 'X', value = '0', uvalue =
    'ZOOMXMIN', uname = 'ZOOMXMIN', xsize = 4)
ZOOMXmaxdisp = cw_field(ZOOMbase2, title = '', value = '0', uvalue =
    'ZOOMXMAX', uname = 'ZOOMXMAX', xsize = 4)
ZOOMYmindisp = cw_field(ZOOMbase2, title = 'Y', value = '0', uvalue =
    'ZOOMYMIN', uname = 'ZOOMYMIN', xsize = 4)
ZOOMYmaxdisp = cw_field(ZOOMbase2, title = '', value = '0', uvalue =
    'ZOOMYMAX', uname = 'ZOOMYMAX', xsize = 4)

ZObutton = widget_button(CONTROLbase, value = 'ZOOM OUT', uvalue = 'ZOUT',
    uname = 'ZOUT')

LDbase = widget_base(CONTROLbase, frame = 3, /column)
LAYERtitle = widget_label(LDbase, value = 'IMAGE LAYER')
LAYERbase = widget_base(LDbase, /row)
PREVbutton = widget_button(LAYERbase, value = 'PREV', uvalue = 'PREV', uname =
    'PREV', sensitive = number)
LAYERlabel = widget_label(LAYERbase, value = strcompress(0), uvalue =
    'LAYERlabel', uname = 'LAYERlabel')
NEXTbutton = widget_button(LAYERbase, value = 'NEXT', uvalue = 'NEXT', uname =
    'NEXT', sensitive = number)

OFFSETbase = widget_base(CONTROLbase, frame = 3, /column)
OFFSETlabel = widget_label(OFFSETbase, value = 'IMAGE OFFSET')
OFFXbase = widget_base(OFFSETbase, column = 3, /base_align_center)
OFFYbase = widget_base(OFFSETbase, column = 3, /base_align_center)
DOWNbutton = widget_button(OFFYbase, value = 'DOWN', uvalue = 'DOWN', uname
    = 'DOWN', sensitive = number)
OFFYlabel = widget_text(OFFYbase, value = '0', uvalue = 'OFFY', uname = 'OFFY',
    xsize = 4)
UPbutton = widget_button(OFFYbase, value = ' UP ', uvalue = 'UP', uname = 'UP',
    sensitive = number)
LEFTbutton = widget_button(OFFXbase, value = 'LEFT', uvalue = 'LEFT', uname =
    'LEFT', sensitive = number)
OFFXlabel = widget_text(OFFXbase, value = '0', uvalue = 'OFFX', uname = 'OFFX',
    xsize = 4)
RIGHTbutton = widget_button(OFFXbase, value = 'RIGHT', uvalue = 'RIGHT', uname =
```

```
        'RIGHT', sensitive = number)

COORDbase = widget_base(CONTROLbase, frame = 3, column = 2, /align_center)
xlabel = widget_label(COORDbase, value = 'X')
xcoord = widget_text(COORDbase, value = '', uvalue = 'XCOORD', uname =
        'XCOORD', xsize = 5)
xabslabel = widget_label(COORDbase, value = 'ABS X')
xabscoord = widget_text(COORDbase, value = '', uvalue = 'XABS', uname = 'XABS',
        xsize = 5)
ylabel = widget_label(COORDbase, value = 'Y')
ycoord = widget_text(COORDbase, value = '', uvalue = 'YCOORD', uname =
        'YCOORD', xsize = 5)
yabslabel = widget_label(COORDbase, value = 'ABS Y')
yabscoord = widget_text(COORDbase, value = '', uvalue = 'YABS', uname = 'YABS',
        xsize = 5)

INTvalue = cw_field(CONTROLbase, title = 'INTENSITY: ', value = '', uname = 'INT',
        uvalue = 'INT', xsize = 9)

QUITbutton = widget_button(CONTROLbase, value = 'DONE', uvalue = 'QUIT', uname
        = 'QUIT')  ;, sensitive=0

widget_control, PPbase, /realize
widget_control, IMGplot, get_value = plotvalue
wset, plotvalue

map = make_array(dim[1], dim[2])

if keyword_set(names) then layernames = names else layernames =
        strcompress(indgen(n_elements(data[0,0,*])))

if dim[0] eq 2 then datai = data else datai = data[*,*,0]
m = {data:data, datai:datai, names:layernames, map:map, mode:0, plotvalue:plotvalue,
        storex1:0, storey1:0, zoomx:[0,dim[1]], zoomy:[0,dim[2]], zoomed:0, doplot:0,
        index:0, number:number, xoffset:lonarr(number + 1), yoffset:lonarr(number + 1),
        color:0, contrast:1.0}
widget_control, PPbase, set_uvalue = m

plot_image, m.datai, /isotropic, title = strcompress(m.index) + ' - ' +
        strcompress(m.names[m.index])
xmanager, 'pixel_picker', PPbase

loadct, 0, /silent
```

```
help, output, /str

sz = size(output.intensities)

print, ''

if size(output.intensities, /type) ne '7' then begin
  disp = [ 'x', 'y', string(output.names) ]
  len = 7 > max(strlen(strcompress(output.names))) + 2
  for i = 0, sz[1] - 1 do disp = [ [disp], [strcompress(output.selected_abs[0,i]),
        strcompress(output.selected_abs[1,i]),
        strcompress(transpose(output.intensities[i,*])) ] ]
  for i = 0, n_elements(disp[*,0]) - 1 do begin
    for j = 0, n_elements(disp[0, *]) - 1 do begin
      for k = 0, len - strlen(disp[i,j]) do disp[i,j] = ' ' + disp[i,j]
    endfor
  endfor
  print, disp
endif

return, output

END
```

Appendix B
EML_Plotter

```
; EML_PLOTTER
; Written by Jason Kimble - Solarlab - University of Memphis - September 10, 2010
; This is a widget driven program to create Emission Measure Loci plots
; It will accept .GENX files containing a data structure called LINES with tags: logt[],
      iobs, iobs_err, emis[], element, ion and wavelength.
; These are the same type of files generated by GENX_MAKER and are the same type
      accepted by DEM_INTERACTIVE.
; Will also save EML plots to an IDL save file and restore these plots.  Will also generate
      PostScript output.
; It is hoped that this program will be sufficiently versatile to generate plots suitable for
      publications directly, thus saving time scripting these plots by hand.
; This program replaces the more limited GENX_EML_PLOTTER.
; Calling sequence is 'EML_PLOTTER'.  It is a procedure and takes no arguments.



FUNCTION EML_MAKE_ARRAYS, lines
; function to create and populate the ARRAYS structure to be put in the MANIP structure

ARRAYS = {EM:lines[0].logt, temp:lines[0].logt}                    ; Defines the ARRAYS
      structure based on the number of elements in the LINES data
ARRAYS = REPLICATE(arrays, n_elements(lines))                     ; Extends the
      ARRAYS structure to match the dimensions of the LINES structure

for i = 0, n_elements(lines) - 1 do begin                         ; For each element in LINES
  for j = 0, n_elements(lines[i].logt) - 1 do begin               ; For each element in the data
      arrays
   if lines[i].emis[j] eq 0 then arrays[i].EM[j] = 0 else arrays[i].EM[j] =
      alog10(lines[i].iobs) - alog10(lines[i].emis[j])            ; Defines the elements of the
      ARRAYS data
  endfor
endfor

return, arrays                                                    ; Sends the resulting ARRAYS
      structure back to the starting routine
END
```

```
FUNCTION EML_ASK, eventop, MSG
; Generates a yes/no dialog to check whether the user wants to quit the program

QUITbase = widget_base(Title = 'ARE YOU SURE?', group_leader = eventop, /modal,
      column = 1)
QUITmsg = widget_label(QUITbase, value = MSG)
BUTTONbase = widget_base(QUITbase, column = 2)
YESbutton = widget_button(BUTTONbase, value = 'YES', uvalue = 'YES', uname =
      'YES')
NObutton = widget_button(BUTTONbase, value = 'NO', uvalue = 'YES', uname = 'YES')

widget_control, QUITbase, /realize
ev = widget_event(QUITbase)
widget_control, ev.id, get_value = choice
widget_control, ev.top, /destroy
return, choice

END




FUNCTION EML_FETCH_PARAMS, manip, eventop, thisevent
; Extracts all the entered parameters from the widget interface to be used in generating
      the plot

wiids = manip.wiids                                                    ; Extracts the
      structure containing widget ids

widget_control, wiids.xmin, get_value = xmin & manip.xmin = xmin              ; Gets
      the Minimum X value from the widget interface
widget_control, wiids.xmax, get_value = xmax & manip.xmax = xmax              ;
      Maximum X value
widget_control, wiids.ymin, get_value = ymin & manip.ymin = ymin              ;
      Minimum Y value
widget_control, wiids.ymax, get_value = ymax & manip.ymax = ymax              ;
      Maximum Y value

widget_control, wiids.widthslider, get_value = width & manip.width = width      ; Gets
      the line width value
widget_control, wiids.styleselect, get_value = style & manip.style = style ; Line style
      value
widget_control, wiids.keyswitch, get_value = keyon & manip.keyon = keyon      ;
```

```
        Legend switch
widget_control, wiids.keyselect, get_value = keypos & manip.keypos = keypos    ;
        Legend position

widget_control, wiids.paletteslider, get_value = palette & manip.palette = palette ; Color
        palette value
widget_control, wiids.offsetslider, get_value = offset & manip.offset = offset                ;
        Color offset value

widget_control, wiids.titleswitch, get_value = titleon & manip.titleon = titleon     ; Show
        title switch
widget_control, wiids.titlefield, get_value = title & manip.title = title                  ; Title
        value
widget_control, wiids.interswitch, get_value = interswitch & manip.interswitch =
        interswitch        ; Plot-intersection switch

return, manip                                                                          ; Sends MANIP
        structure containing these values back to the event handling routine
END




FUNCTION EML_LOAD, manip, filename, eventop
; Loads a saved file and sets all parameters, except for the plot and widget ids, which are
        retained

winid = manip.winid
wiids = manip.wiids
restore, filename
manip.winid = winid
manip.wiids = wiids

widget_control, wiids.xmin, set_value = manip.xmin; & manip.xmin = xmin    ; sets the
        Minimum X value from the widget interface
widget_control, wiids.xmax, set_value = manip.xmax; & manip.xmax = xmax    ;
        Maximum X value
widget_control, wiids.ymin, set_value = manip.ymin; & manip.ymin = ymin    ;
        Minimum Y value
widget_control, wiids.ymax, set_value = manip.ymax; & manip.ymax = ymax    ;
        Maximum Y value

widget_control, wiids.widthslider, set_value = manip.width ;&; manip.width = width  ;
```

```
        sets the line width value
widget_control, wiids.styleselect, set_value = manip.style ;&; manip.style = style  ; Line
        style value
widget_control, wiids.keyswitch, set_value = manip.keyon ;&; manip.keyon = keyon  ;
        Legend switch
widget_control, wiids.keyselect, set_value = manip.keypos ;&; manip.keypos = keypos ;
        Legend position

widget_control, wiids.paletteslider, set_value = manip.palette ;&; manip.palette =
        palette  ; Color palette value
widget_control, wiids.offsetslider, set_value = manip.offset ;&; manip.offset = offset    ;
        Color offset value

widget_control, wiids.titleswitch, set_value = manip.titleon ;&; manip.titleon = titleon  ;
        Show title switch
widget_control, wiids.titlefield, set_value = manip.title ;&; manip.title = title   ; Title
        value
widget_control, wiids.interswitch, set_value = manip.interswitch ;& manip.interswitch =
        interswitch  ; Plot-intersection switch

return, manip
END




FUNCTION EMLplot, manip, ps=ps
; This creates the plot from the data and settings in the MANIP routine

if not(keyword_set(ps)) then wset, manip.winid                                    ; Only
        sets the window (for good measure) if it is not plotting to a postscript file
loadct, 0, /silent                                                                ; Sets the color
        table to 0

if (manip.xmin ge manip.xmax or manip.ymin ge manip.ymax) then begin              ;
        Finds max and min range values if entered values are not valid - minimum greater
        than maximum, or equal to each other
  xmax = max(manip.arrays.temp)
  xmin = min(manip.arrays.temp)
  ymax = max(manip.arrays.em)
  ytest = 5000                                                                    ; Looks for a
        minimum value for y that is greater than 0
  for i = 0, n_elements(manip.arrays) - 1 do begin
```

```
    yt = where(manip.arrays[i].em gt 0)
    ytest = [ytest, manip.arrays[i].em[yt] ]
  endfor
  ymin = min(ytest)
endif else begin                                                    ; Otherwise, if
        the range values are valid, extracts them
  xmax = manip.xmax
  ymax = manip.ymax
  xmin = manip.xmin
  ymin = manip.ymin
endelse

if manip.titleon eq 0 then plot, [-50] , [-50], xrange = [xmin, xmax], yrange = [ymin,
        ymax], xtitle="log T  (K)", ytitle="log EM  (cm!S!E-5!N)", /xstyle, /ystyle else
        plot, [-50], [-50], xrange = [xmin, xmax], yrange = [ymin, ymax], title =
        manip.title, xtitle="log T  (K)", ytitle="log EM  (cm!S!E-5!N)", charsize = 1.3,
        /xstyle, /ystyle   ; Creates plot, dependent on whether the title is set to be displayed

for i = 0, n_elements(manip.arrays) - 1 do begin                    ; Steps through
  X = 0                                                             ; Incrimentally
        creates X and Y plotting arrays containing no zeroes
  Y = 0
  for j = 0, n_elements(manip.arrays[i].EM) - 1 do begin
    if manip.arrays[i].EM[j] ne 0 then begin
      if Y[0] NE 0 then Y = [Y, manip.arrays[i].EM[j] ] else Y = manip.arrays[i].EM[j]
      if X[0] NE 0 then X = [X, manip.arrays[i].temp[j] ] else X = manip.arrays[i].temp[j]
    endif
  endfor
  style = manip.style                                               ; Extracts style
        value

  if manip.keyon eq 1 then begin                                    ; When Legend
        is set to plot
    case manip.keypos of                                            ; Checks
        chosen legend position
      0 : begin                                                     ; Top left
        Legend position
        XKey = [ xmin + ( ( xmax - xmin ) / 20), xmin + 1.7 *((xmax - xmin)/20) ]  ; Finds x
        ranges for line sample
        YKey = [ ymax - ( (i+1) * (ymax - ymin) / 35 ) , ymax - ( (i+1) * (ymax - ymin) /
        35 ) ]     ; Finds y ranges for line sample
        LabelX = XKey[1] + ( (xmax - xmin) / 50 )                   ; Finds starting
        X coordinate for Legend Label
        LabelY = YKey[0] - ( (ymax - ymin) / 35 ) * .1                      ; Finds
```

```
      starting Y coordinate for Legend Label
    endcase
    1 : begin                                                    ; As previous,
      but for bottom left position
     XKey = [ xmin + ( ( xmax - xmin ) / 20), xmin + 1.7 * ((xmax - xmin)/20) ]
     YKey = [ ymin + ( (i+1) * (ymax - ymin) / 35 ), ymin + ( (i+1) * (ymax - ymin) /
      35 ) ]
     LabelX = XKey[1] + ( (xmax - xmin) / 50 )
     LabelY = YKey[0] - ( (ymax - ymin) / 35 ) * .1
    endcase
    2 : begin                                                    ; As previous,
      but for Top Right potion
     XKey = [ xmax - 1.4 * ( ( xmax - xmin ) / 20 ), xmax - .7*((xmax-xmin)/20) ]
     YKey = [ ymax - ( (i+1) * (ymax - ymin) / 35 ) , ymax - ( (i+1) * (ymax - ymin) /
      35 ) ]
     LabelX = XKey[0] - ( (xmax - xmin) / 50 ) * 5.5
     LabelY = YKey[0] - ( (ymax - ymin) / 35 ) * .1
    endcase
    3 : begin                                                    ; As previous,
      but for Bottom Right
     XKey = [ xmax - 1.4 * ( ( xmax - xmin ) / 20 ), xmax - .7*((xmax-xmin)/20) ]
     YKey = [ ymin + ( (i+1) * (ymax - ymin) / 35 ), ymin + ( (i+1) * (ymax - ymin) /
      35 ) ]
     LabelX = XKey[0] - ( (xmax - xmin) / 50 ) * 5.5
     LabelY = YKey[0] - ( (ymax - ymin) / 35 ) * .1
    endcase
  endcase
endif
case style of                                                    ; Determines which
      style of line differentiation was selected
  0 : oplot, X, Y, thick = manip.width                           ; Plots with no
      stylistic differences
  1 : begin                                                      ; Differentiated
      by color
    loadct, manip.palette, /silent                               ; Loads the
      color table selected
    oplot, X, Y, thick = manip.width, color = ( i * ( 254 / ( n_elements(manip.arrays) ) ) )
      + manip.offset   ; plots using that color
    if manip.keyon eq 1 then begin
     oplot, XKey, YKey, thick = manip.width, color = ( i * ( 254 /
      ( n_elements(manip.arrays) ) ) ) + manip.offset          ; plots the Legend, if selected
     loadct, 0, /silent                                          ; Reloads color table
      zero
     if NOT(KEYWORD_SET(PS)) then xyouts, LabelX, LabelY, manip.name[i], size =
```

```
        1.2 else xyouts, LabelX, LabelY, manip.name[i], size = .8     ; Plots the Labels in
          the Legend
       endif
     endcase
     2 : begin                                                        ; Differentiated
          by line style
       if i lt 6 then oplot, x, y, linestyle = i, thick = manip.width else oplot, x, y, psym = ( 7 -
          i ), linestyle = 0, symsize = 1.5, thick = manip.width ; plots different line styles for
          the first 6 lines, and then connected symbols for any further ones
       if manip.keyon eq 1 then begin                                 ; If Legend is
          set to plot
        if i lt 6 then oplot, XKey, YKey, linestyle = i, thick = manip.width else oplot, XKey,
          YKey, psym = ( 7 - i ), linestyle = 0, symsize = 1.5, thick = manip.width     ; plots
          line samples in different line styles for the first 6 lines, and then connected symbols
          for any further ones
        if NOT(keyword_set(ps)) then xyouts, LabelX, LabelY, manip.name[i], size = 1.2
          else xyouts, LabelX, LabelY, manip.names[i], size = .8                    ; Plots
          Labels
       endif
     endcase
   endcase
endfor

if manip.interswitch eq 1 then begin                                ; When set to
        plot intersection
  loadct, 11, /silent                                               ; Loads a red
        table
  oplot, [xmin, xmax], [manip.intery, manip.intery]                 ; Plots the
        horizontal line in red
  oplot, [manip.interx, manip.interx], [ymin, ymax]                 ; Plots the
        vertical line in red
  loadct, 0, /silent                                                ; Reloads color
        table zero
endif

return, 1                                                           ; Sends back
        the value 1, indicating a successful plot
end




PRO EML_PLOTTER_event, ev
```

```
; Event handler routine.  Main routine while the widget interface is active.
COMMON EML_block, h, c, number, number_points, lines, text, plotid            ; ###
      REMOVE ???

doquit = 'NO'                                                                 ; Initializes the
      end program indicator
widget_control, ev.top, get_uvalue = manip                                    ; Gets the MANIP
      structure from the top level widget's uvalue
widget_control, ev.id, get_uvalue = uval                                      ; Gets the
      uvalue identifier of the widget that generated the event

case uval of                                                                  ; Determines
      what kind of event
 'QUIT' : doquit = EML_ASK(ev.top, 'Are you sure you want to quit?')
                  ; QUIT button clicked - ask to end program
 'PLOTwindow' : if manip.interswitch eq 1 and ev.press eq 1 then begin        ; Plot
      window clicked in, only relevant if set to plot an intersection
  cursor, x1, y1, /nowait, /data                                              ; gets the
      coordinates clicked on - not a good way to do this with widget graphics
  manip.interx = x1 & manip.intery = y1                                       ; Sets the
      intersection coordinates
  print, '( ', strcompress(x1), ' , ', strcompress(y1), ' )'                  ; Prints the
      intersection coordinates just chosen
 endif
 'SAVE' : begin                                                               ; Save button
      clicked
  savefile = dialog_pickfile(title = 'Please enter a filename to save.', filter =
      ['*_EML.sav;*_EML.SAV;*_eml.sav;*_eml.SAV'], /write, /overwrite_prompt)     ;
      Prompts for a save file name
  if savefile eq '' then GOTO, NOSAVE                                         ; If
      cancel clicked in the file selection dialog, nul string causes save routine to abort
  saveext = strsplit(savefile, '_', /extract)
  saveext = saveext[n_elements(saveext) - 1]
  saveext = strupcase(saveext)
  if saveext ne 'EML.SAV' then savefile = savefile + '_EML.SAV'               ; Checks file
      name and extension for format
  print, 'Saving ', savefile, '.'                                             ; Prints saving message
  save, manip, filename = savefile                                            ; Saves MANIP
      structure in save file
  NOSAVE:                                                                     ; Routine exit
      point
 endcase
 'PS' : begin                                                                 ; PostScript
      button clicked
```

```
   psfile = dialog_pickfile(title = 'Please select or enter title of Postscript file.', filter =
       ['*.ps;*.PS'], /write, /overwrite_prompt)        ; PostScript File name dialog
   if psfile eq '' then GOTO, NOPS                                    ; If cancel
       clicked in file selection, nul string causes PS routine to abort
   psext = strsplit(psfile, '.', /extract)
   psext = psext[n_elements(psext) - 1]
   psext = strupcase(psext)
   if psext ne 'PS' then psfile = psfile + '.PS'                      ; Checks for correct
       file extension
   print, 'Creating Postscript file: ', psfile, '.'                   ; Prints saving message
   SET_PLOT, 'PS'                                                     ; sets the plot to
       postscript
   DEVICE, FILE=psfile, /landscape, /medium, /COLOR, BITS=8                    ;
       Sets the device characteristics and filename
   pssuccess = EMLplot(manip, /ps)                                    ; Invokes plot
       function with PS keyword set to keep wset from atempting to set the window
   device, /close                                                     ; finalizes the
       file
   set_plot, 'x'                                                      ; Sets plot back to the x
       server
   NOPS:                                                              ; routine exit
       point
  endcase

  'LOAD' : begin                ; LOAD saved EML button clicked
   filename = dialog_pickfile(Title = 'SAVED EML FILE TO LOAD?', filter =
       '*_EML.SAV;*_eml.SAV;*_EML.sav;*_eml.sav', /read, /must_exist)
   if filename eq '' then GOTO, NOLOAD              ; Nul string indicates cancel clicked,
       and skips routine
   doload = EML_ASK(ev.top, 'Are you sure you want to overwrite your current EML?
       Unsaved plot will be lost!')    ; Double checks that current plot is no longer wanted
   if doload eq 'YES' then manip = EML_LOAD(manip, filename, ev.top)      ; Calls the
       LOAD function, which resets all data and parameters, if yes was selected
   NOLOAD:
  endcase

  else : manip = EML_FETCH_PARAMS(manip, ev.top, ev.id)                       ;
       Any other widget event triggers a refresh of parameters
endcase

plotsuccess = EMLplot(manip)                                                  ; Plots
       during each event

widget_control, ev.top, set_uvalue = manip                                    ; Puts current
```

```
        version of MANIP structure with current parameters into the top level widget
        uvalue

if doquit eq 'YES' then begin                                   ; When quit
        program indicator set, which happens when QUIT button clicked
  loadct, 0, /silent                                            ; Reloads color
        table zero
  widget_control, ev.top, /destroy                              ; Destroys top
        level widget and all its children
endif

end




PRO EML_PLOTTER
; Main routine. More is done in the event handler, but this is the parent routine. Widgets
        are defined and created, and all parameters are defined and initialized.
COMMON EML_block, h, c, number, number_points, lines, text, plotid

filename = dialog_pickfile(title = 'Please select a GENX file containing LINES data.',
        filter = ['*.genx;*.GENX;*.GenX;*.GENx;*.genX'], /must_exist)      ; prompts for
        the file to use
if filename eq '' then GOTO, THEEND                                        ; When
        cancel button in file dialog is clicked, nul string causes the entire program to abort
rd_genx, filename, lines, header = header, text = text                     ; Reads
        the selected .GENX file, placing the data into the LINES structure, the header into
        a header variable, and the index into index variable

h = 6.62620e-27   ;  erg * seconds                  ; ### REMOVE ???  Probably not
        necessary
c = 2.9979250e10  ;  cm / second                    ; ### REMOVE ???  Probably not
        necessary

number = n_elements(lines)                          ; ### REMOVE ???  Don't know if
        this is used
number_points = n_elements(lines[0].logt)           ; ### REMOVE ??? Don't know if
        this gets used

EMLbase = widget_base(Title = 'EML Plotter: ' + file_basename(filename), column = 2)
          ; Defines top level widget
CONTROLbase = widget_base(EMLbase, /column)
```

```
        ; Defines container widget for the controls
PLOTbase = widget_base(EMLbase,  column = 1)                                    ;
      Defines container widget for the plot window

RANGEbase = widget_base(CONTROLbase, column = 1, frame = 5, /base_align_center)
                ; Defines container widget for the range widgets
RANGElabel = widget_label(RANGEbase, value = 'RANGE', /align_center)
        ; Label for the range fields
Xbase = widget_base(RANGEbase, /row)                                            ;
      Container for the x range fields
XMin = cw_field(Xbase, Title = 'X Min: ', uvalue = 'RANGE', uname = 'XMIN',
      /floating, /return_events)        ; Minimum x value field widget
XMax = cw_field(Xbase, Title = 'X Max: ', uvalue = 'RANGE', uname = 'XMAX',
      /floating, /return_events)        ; Maximum x value field widget
Ybase = widget_base(RANGEbase, /row)                                            ;
      Container for the y range fields
YMin = cw_field(Ybase, Title = 'Y Min: ', uvalue = 'RANGE', uname = 'YMIN',
      /floating, /return_events)        ; Minimum y value field widget
YMax = cw_field(Ybase, Title = 'Y Max: ', uvalue = 'RANGE', uname = 'YMAX',
      /floating, /return_events)        ; Maximum y value field

STYLEbase = widget_base(CONTROLbase,  row = 2)
        ; Container widget for the style controls
LINESTYLEbase = widget_base(STYLEbase, frame = 5, /column)
        ; Container widget for the line style controls
STYLElabel = widget_label(LINESTYLEbase, value = 'LINE STYLE')
        ; Label widget for the style selector
STYLEselect = cw_bgroup(LINESTYLEbase, ['None', 'Line Colour', 'Line Style'],
      uvalue = 'STYLE', uname = 'STYLE', set_value = 0, /exclusive)     ; Style selector
      widget (radio buttons)

PALETTEbase = widget_base(STYLEbase, /column, frame = 5, /base_align_center)
        ; Container widget for the controls to select color palette
PALETTElabel = widget_label(PALETTEbase, value = 'PALETTE')
        ; Label widget for the color palette controls
PALETTEslider = widget_slider(PALETTEbase, minimum = 12, maximum = 38, value =
      34, xsize = 100, uvalue = 'PALETTE', uname = 'PALETTE', /drag)  ; Slider control
      widget for color palette selection
OFFSETlabel = widget_label(PALETTEbase, value = 'COLOR OFFSET')
                ; Color offset label widget
OFFSETslider = widget_slider(PALETTEbase, minimum = 0, maximum = 255, value =
      0, xsize = 100, uvalue = 'OFFSET', uname = 'OFFSET', /drag)       ; Color offset
      slider control widget
```

```
WIDTHbase = widget_base(STYLEbase, /column, frame = 5, /base_align_center)
        ; Container widget for the line width control
WIDTHlabel = widget_label(WIDTHbase, value = 'LINE WIDTH')
        ; Label widget for the line width control
WIDTHslider = widget_slider(WIDTHbase, minimum = 0, maximum = 5, value = 1,
      xsize = 50, uvalue = 'WIDTH', uname = 'WIDTH', /drag)     ; Line width slider
      control widget

TITLEbase = widget_base(STYLEbase, frame = 5, /column)
        ; Container widget for the Title controls
TITLElabel = widget_label(TITLEbase, value = 'TITLE')                              ;
      Label widget for the title controls
TITLEfield = cw_field(TITLEbase, uvalue = 'TITLE', uname = 'TITLE', title = '',
      /string, /return_events) ; Title entry field widget
TITLEswitch = cw_bgroup(TITLEbase, ['SHOW TITLE'], uvalue = 'TITLESWITCH',
      uname = 'TITLESWITCH', /nonexclusive)     ; Title display switch widget
      (checkbox)

KEYbase = widget_base(STYLEbase, frame = 5, /column)                               ;
      Legend controls container widget
KEYlabel = widget_label(KEYbase, value = 'LEGEND')                                 ;
      Label widget for the legend controls
KEYswitch = cw_bgroup(KEYbase, ['SHOW LEGEND'], uvalue = 'KEYSWITCH',
      uname = 'KEYSWITCH', /nonexclusive)       ; Legend display switch widget
      (checkbox)
KEYselect = cw_bgroup(KEYbase, ['Upper Left', 'Lower Left', 'Upper Right', 'Lower
      Right'], label_top = 'Position', column = 2, set_value = 0, uvalue = 'KEYSELECT',
      uname = 'KEYSELECT', /exclusive)   ; Label location selection widget (radio
      buttons)

EXTRAbase = widget_base(CONTROLbase, /row)                                         ;
      Container widget for left over controls
OUTbase = widget_base(EXTRAbase, frame = 5, /column)
        ; Output controls container widget
OUTlabel = widget_label(OUTbase, value = 'OUTPUT')                                 ;
      Output label widget
SAVEbutton = widget_button(OUTbase, value = 'SAVE PLOT DATA', uvalue = 'SAVE',
      uname = 'SAVE')         ; Save data button widget
PSbutton = widget_button(OUTbase, value = 'MAKE POSTSCRIPT', uvalue = 'PS',
      uname = 'PS')    ; Create PostScript file button widget

LASTbase = widget_base(EXTRAbase, /column)                                         ;
      Container for leftover bits
INTERswitch = cw_bgroup(LASTbase, ['PLOT INTERSECTION'], uvalue =
```

```
        'INTERSWITCH', uname = 'INTERSWITCH', /nonexclusive)        ; Intersection
        plotting switch control widget (checkbox)
LOADbutton = widget_button(LASTbase, value = 'LOAD SAVED EML', uvalue =
        'LOAD', uname = 'LOAD') ; LOAD saved EML button
QUITbutton = widget_button(LASTbase, value = 'QUIT', uvalue = 'QUIT', uname =
        'QUIT')  ; QUIT program button widget

TEXTlabel = widget_label(CONTROLbase, value = 'FILE INFO')
          ; File info display widget label
FILEtext = widget_text(CONTROLbase, value = text, ysize = 6, /wrap, /scroll)              ;
        File info display widget

PLOTwindow = widget_draw(PLOTbase, xsize = 1000, ysize = 1000, uname =
        'PLOTwindow', uvalue = 'PLOTwindow', /button_events)     ; Plot window widget

wiids = {Xmin:Xmin, Xmax:Xmax, Ymin:ymin, ymax:ymax, styleselect:styleselect,
        paletteslider:paletteslider, offsetslider:offsetslider, widthslider:widthslider,
        keyswitch:keyswitch, keyselect:keyselect, titleswitch:titleswitch, titlefield:titlefield,
        interswitch:interswitch, text:text}        ; Structure that contains widget IDs for all
        control widgets (not containers or labels)

widget_control, EMLbase, /realize                                                          ;
        Realizes the top level widget
widget_control, PLOTwindow, get_value = winid                                              ;
        Gets the window number for the plot window widget

if where(tag_names(lines) eq 'INSTRUMENT') ne -1 then begin
  if lines[0].instrument eq 'XRT' then name = lines.elem else name = lines.elem +
        lines.ion + strcompress(lines.wave)
endif else name = lines.elem + lines.ion + strcompress(lines.wave)

arrays = EML_MAKE_ARRAYS(lines)
          ; Invokes the function to build the ARRAYS structure
manip = {winid:winid, arrays:arrays, name:name, xmin:0.0, xmax:0.0, ymin:0.0,
        ymax:0.0, width:1, style:0, keypos:0, keyon:0, palette:34, offset:0, wiids:wiids,
        title:'', titleon:0, interx:0.0, intery:0.0, interswitch:0}  ; Builds the MANIP structure
        and initializes its tags
manip.winid = winid                                                                        ; passes
        the Window ID for the plot widget to its tag in the MANIP structure
plotsuccess = EMLplot(manip)                                                               ;
        Invokes the first plot

widget_control, EMLbase, set_uvalue = manip                                                ;
        Puts the MANIP array safely into the top level widget's uvalue for passing to the
```

```
        handler routine
xmanager, 'EML_PLOTTER', EMLbase                                    ;
        Starts the event handler

THEEND:                                                             ;
        Program exit point
END
```

Appendix C
ECHIDNA: A DEM Forward Folder

```
; written by Jason Kimble, Solar Lab, University of Memphis, Summer 2010.
; IDL - Written for X display and 'nix directory structure, requires use of widgets.
; A forward folder for the manipulation of Differential Emission Measures.
; Will return a plot of predicted verses observed intensities.
; Calling sequence - 'ECHIDNA'.
; Accepts *.genx files containing data structures named lines, containing tags: logt[], iobs,
      iobs_err, emis[], element, ion and wavelength.
; These files are the ones created by a program called GENX_MAKER, and are of the
      same type as accepted by DEM_INTERACTIVE.
; Uses emissivity arrays obtained from the CHIANTI atomic physics database.

; "[...] another monster, irresistible, in no wise like either to mortal men or to the undying
      gods, even the goddess fierce Echidna who is half a nymph with glancing eyes and
      fair cheeks, and half again a huge snake,
;  great and awful, with speckled skin, eating raw flesh beneath the secret parts of the
      holy earth. And there she has a cave deep down under a hollow rock far from the
      deathless gods and mortal men.
;  There, then, did the gods appoint her a glorious house to dwell in: and she keeps guard
      in Arima beneath the earth, grim Echidna, a nymph who dies not nor grows old all
      her days."
;              - Hesiod - Theogony  [300 - 305]

function ECHIDNA_arrange_lines, lines
; arranges the lines structure entries by their peak formation temperatures, and then for
      entries with the same peak formation
; temperatures, by wavelength

; arranges the lines entries by peak formation temperature (hopefully)
maxindex = make_array(n_elements(lines))          ; creates array to hold indices of
      maximum contribution function for each line
for i = 0, n_elements(lines) - 1 do maxindex[i] = where(lines[i].emis eq
      max(lines[i].emis))              ; finds index of the maximum of the emissivity array
      for each line
sortindex = sort(maxindex)             ; puts the placement order of the array of indices of
      maximum emissivity into an array
lines_sorted = lines                   ; creates a temporary structure to hold the sorted
      lines structure entries
for i = 0, n_elements(lines) - 1 do lines_sorted[i] = lines[sortindex[i]]       ; puts in each
      element of the temporary structure the ordered element of the lines structure
lines = lines_sorted                   ; copies the sorted lines back into the lines structure
```

```
; arranges the lines entries of equivalent formation temperature by wavelength (I think)
for i = 0, n_elements(lines) - 1 do maxindex[i] = where(lines[i].emis eq
      max(lines[i].emis))      ; refills the maxindex array to match the previously sorted
      lines
number_peak_temps = 1                              ; sets the initial number of identical
      peak formation temperatures to at least 1
for i = 1, n_elements(maxindex) - 1 do if maxindex[i] ne maxindex[i - 1] then
      number_peak_temps = number_peak_temps + 1          ; counts the number of
      unique peak formation temperatures
peak_temps = make_array(number_peak_temps)    ; makes an array to hold the different
      temperatures
count = 1                                          ; initializes a counting variable
tempgroup = 0                                      ; initializes a variable to indicate which
      group of identical temperatures
for i = 1, n_elements(maxindex) - 1 do begin              ; counts how many entries
      there are with each temperature value
  if maxindex[i - 1] eq maxindex[i] then count = count + 1 else begin
    peak_temps[tempgroup] = count
    tempgroup = tempgroup + 1
    count = 1
  endelse
  peak_temps[tempgroup] = count
endfor
startindex = total(peak_temps, /cumulative) ; this and following places the starting index
      for each group of identical temperatures
startindex = startindex - peak_temps
for j = 0, number_peak_temps - 1 do begin   ; sorts individual groups of same
      temperatures by wavelength entry values
  if peak_temps[j] gt 1 then begin                 ; no point if unique temperature
    wavesortarray_index = make_array(peak_temps[j])       ; arrays for sorting by
      wavelength
    wavesortarray = wavesortarray_index
    for i = 0, peak_temps[j] - 1 do wavesortarray_index[i] = startindex[j] + i         ; puts
      indices of same temperatures in array
    for i = 0, peak_temps[j] - 1 do wavesortarray[i] = lines[wavesortarray_index[i]].wave ;
      pust wavelengths in array
    wavesortindexorder = sort(wavesortarray)          ; gets the arrangement order of
      indices of wavelengths in each group
    for i = 0, peak_temps[j] - 1 do lines_sorted[wavesortarray_index[i]] =
      lines[wavesortarray_index[wavesortindexorder[i]]]   ; puts values in by sorted order
  endif
endfor
lines = lines_sorted              ; copies sorted lines entries into lines structure
return, lines                     ; sends back newly sorted lines structure
```

```
end



function ECHIDNA_plot, manip, eventtop, displayonly = displayonly, ps = ps
; creates the DEM and ratio plots from current data in the manipulation structure
; entire ratio plot recalculated each time - not very efficient but probably necessary
; modifies the manipulation structure entries
; requires manipulation structure and id of top level widget. also accepts a display only
        switch that will prevent from sending
; the structure back to the top level widget's uvalue

if keyword_set(PS) then errlabelsize = .9 else errlabelsize = 1.5
if not keyword_set(displayonly) then plottype = manip.plottype else plottype = 0

binstep = abs(manip.bins[1] - manip.bins[0])    ; finds difference between temperature
        bins - they must be evenly spaced for this to work right
integral = make_array(n_elements(manip.lines))    ; calculates the sum of the products of
        the contribution functions, the dem value and the temperature bin widths for each
        line
for i = 0, n_elements(manip.lines) - 1 do begin
  for j = 0, n_elements(manip.bins) - 1 do begin
    integral[i] = integral[i] + (  ( 10 ^ manip.dem[j] ) * manip.lines[i].emis[j] *
        manip.binwidths[j] )
  endfor
endfor
manip.predicted = integral        ; places the integrals for each line in a predicted array in
        the manipulation structure
ratio_y = manip.predicted/manip.observed    ; calculates the predicted over observed
        values
;ratio_err = ( ( manip.predicted - manip.observed )  / manip.lines.iobs_err ) ^ 2       ;
        calculatees error values for the ratios - got this one from Monster
; been using this one: ratio_err = abs( manip.predicted / ( manip.observed -
        manip.lines.iobs_err ) - manip.predicted / manip.observed )  ; calculates error
        values for the ratios from observational errors - direct, brute force approach
ratio_err = (manip.lines.iobs_err * manip.predicted) / (manip.observed ^ 2)      ;
        calculates error values for the ratios - got this one from the CHIANTI user's guide -
        matches little red stats book
;X2 = ( (manip.observed - manip.predicted) ^ 2 ) / ( ( ratio_err ) ^ 2)
X2 = ( ( ratio_y - 1 ) ^ 2 ) / ( ( ratio_err ) ^ 2 )
X2 = TOTAL(X2)
RX2 = X2 / ( n_elements(manip.lines) - 1 )
```

```
manip.ratio_y = ratio_y
manip.ratio_err = ratio_err
manip.chi_squared = RX2
if not keyword_set(ps) then wset, manip.RATIOdraw     ; sets the plotting window to the
      ratio plot
ploterr, ratio_y, ratio_err, xrange = [-1, n_elements(manip.observed)], psym = 4, /xs,
      yrange = [-1, 3], /xstyle, /ystyle   ; plots the ratios with error bars
oplot, [-2, n_elements(manip.observed) + 2 ], [1, 1], linestyle = 1         ; overplots a
      horizontal line at y = 1
for i = 0, n_elements(manip.lines) - 1 do begin     ; outputs individual line info onto the
      ratio plot
 elem = manip.lines[i].elem + ' ' + manip.lines[i].ion
 if where(tag_names(manip.lines) eq 'WAVE') ne -1 then wave = manip.lines.wave
 if where(tag_names(manip.lines) eq 'INSTRUMENT') ne -1 then begin
  if manip.lines[0].instrument eq 'XRT' then begin
   elem = manip.lines[i].elem
   wave = strarr(n_elements(manip.lines.wave))
  endif
 endif
 xyouts, i, ratio_y[i] - (ratio_err[i] + .15), elem, alignment = .5, charsize = errlabelsize
 peakformationtemp = manip.bins[where(manip.lines[i].emis eq
      max(manip.lines[i].emis))]
 peakformationtemp = strtrim(peakformationtemp)
 peakformationtemp = strsplit(peakformationtemp, '.', /extract)
 peakformationtemp = peakformationtemp[0] + '.' + strmid(peakformationtemp[1], 0, 1)
 peakformationtemp = strcompress(peakformationtemp)
 xyouts, i, ratio_y[i] + (ratio_err[i] + .1), peakformationtemp, alignment = .5, charsize =
      errlabelsize
 xyouts, i, ratio_y[i] - (ratio_err[i] + .25), strcompress(wave[i]), alignment = .5, charsize
      = errlabelsize
endfor
xyouts, n_elements(manip.observed) - 2.5, 2.8, 'X!U2!N = ' + strcompress(RX2),
      charsize = errlabelsize
if keyword_set(ps) then begin
 device, /close
 DEVICE, FILE=ps, /landscape, /medium, /COLOR, BITS=8
endif
if not keyword_set(ps) then wset, manip.DEMdraw   ; sets the plot window to the DEM
      plot - must be set last to make that plot sensitive to pointer control
if plottype eq 0 then begin
 plot, manip.bins, manip.last_dem, yrange = [manip.ymin, manip.ymax]
      ,/xs,/ys,ylog=ylog,xtitle='log!d10!n(T [K])',ytitle='DEM', xmargin = 10, ymargin =
      5, charsize = 1, linestyle = 2, /device;, _extra=e ; plots old DEM in dotted line
 oplot, manip.bins, manip.dem      ; overplots current DEM
```

```
endif else if plottype eq 1 then begin
  plot, manip.bins, alog10( (10^manip.last_dem) * manip.binwidths ),  yrange =
        [manip.ymin + 4, manip.ymax + 5] ,/xs,/ys,ylog=ylog,xtitle='log!d10!n(T
        [K])',ytitle='EM', xmargin = 10, ymargin = 5, charsize = 1, linestyle = 2, /device;,
        _extra=e  ; plots old DEM in dotted line
  oplot, manip.bins, alog10( (10^manip.dem) * manip.binwidths )     ; overplots current
        DEM
endif
if manip.SHOWbins eq 1 then BEGIN   ; conditionally plots the boundaries of the
        temperature bins in dotted lines, depending on an entry in the manip structure
  binwidths = alog10(manip.binwidths)         ; gets the width for each temperature bin in
        plot space
  dem = manip.dem
  if manip.plottype eq 1 then dem = dem + alog10(manip.binwidths)
  for i = 0, n_elements(manip.bins) - 1 do begin    ; plots two vertical and one horizontal
        side for each temp bin
    ls = 1
    oplot, [(manip.bins[i] - binstep/2), (manip.bins[i] - binstep/2)], [0, dem[i]], linestyle =
        ls
    oplot, [(manip.bins[i] + binstep/2), (manip.bins[i] + binstep/2)], [0, dem[i]], linestyle =
        ls
    oplot, [(manip.bins[i] - binstep/2), (manip.bins[i] + binstep/2)], [dem[i], dem[i]],
        linestyle = ls
  endfor
endif
if not keyword_set(displayonly) then widget_control, eventtop, set_uvalue = manip ; puts
        the manip structure back in top level widget's uvalue unless told not to
return, manip           ; sends back the manipulation structure
end
```

```
function ECHIDNA_YorN, title = title, msg = msg, eventop = eventop
; function to produce an interrupting (modal) yes or no dialog box and return the selection

ECHIDNA_YorN_base = widget_base(title = title, /modal, group_leader = eventop,
        uvalue = 'NO', column = 1, /base_align_center)      ; defines widgets
message_label = widget_label(ECHIDNA_YorN_base, value = msg)
button_container = widget_base(ECHIDNA_YorN_base, frame = 2, column = 2)
yes_button = widget_button(button_container, value = '     YES     ', uvalue = 'YES',
        uname = 'YES', /align_center)
no_button = widget_button(button_container, value = '      NO     ', uvalue = 'NO', uname
```

```
        = 'NO', /align_center)
widget_control, ECHIDNA_YorN_base,
        /realize                                                    ; realizes widgets
ev =
        widget_event(ECHIDNA_YorN_base)
        ; waits for and retrieves widget event
widget_control, ev.id, get_uvalue =
        choice                                                      ; fetches the value of
        the clicked button
widget_control, ev.top,
        /destroy                                                    ; destroys
        widgets
return, choice                                                      ;
        returns the choice made
end



function display_info, manip
; displays information from the header and text entries in the .GENX file providing the
        lines information being studied in a separate text window

tnames = tag_names(manip.lines)                             ; gets an array of tags from the
        lines structure
if where(tnames eq 'INSTRUMENT') ne -1 then begin           ; makes sure that it
        containes the tag 'INSTRUMENT,' a newer tag from a newer version of
        GENX_MAKER
  ; finds and prepares the instrument name
  instr_txt = 'INSTRUMENT: ' + manip.lines[0].instrument
  ; finds and prepares the abundance file name
  Abund_file = manip.lines[0].abundance_filename
  abund_file = strsplit(abund_file, '/', /extract)
  abund_file = abund_file[n_elements(abund_file) - 1]
  abund_txt = 'ABUND FILE: ' + abund_file
  ; finds and prepares the ion equilibrium file name
  ioneq_file = manip.lines[0].ioneq_filename
  ioneq_file = strsplit(ioneq_file, '/', /extract)
  ioneq_file = ioneq_file[n_elements(ioneq_file) - 1]
  ioneq_txt = 'IONEQ FILE: ' + ioneq_file
  ; finds and prepares the electron density value
  density_txt = 'ELEC DENS: '+strcompress(manip.lines[0].density)
  note_txt = manip.text                                     ; finds any additional notes provided
```

```
        by the user at file creation
  info_txt = [instr_txt, abund_txt, ioneq_txt, density_txt, note_txt]  ; puts all informational
        elements into an array for display
endif else begin                                        ; if the 'INSTRUMENT' tag (and
        therefor other informational tags) do not exist, indicates that there is no available
        information
  info_txt = 'DATA NOT AVAILABLE'
endelse
; defines widgets to display file information
ECHIDNA_info_box = widget_base(title = 'Data Info')   ;, xsize = 400, ysize = 400
INFObox = widget_text(ECHIDNA_info_box, value = info_txt, /wrap, xsize = 65, ysize
        = 12, /scroll)
widget_control, ECHIDNA_info_box, /realize

return, 1
end



function ECHIDNA_load_DEM, manip, eventop
; allows the loading of a presaved (or separately created) DEM curve

DEM_load_file = dialog_pickfile(title = 'Please select saved DEM to load.', filter =
        ['*.SAV;*.sav;*.GENX;*.GenX;*.GENx;*.genX;*.genx'], path =
        manip.genx_path, /read, /must_exist)  ; prompts for the file containing the DEM -
        allows .GENX of IDL's .SAV formats
if DEM_load_file eq '' then goto,
        LOADEND
        ; checks to see that a file has been selected/entered
tempname = strsplit(DEM_load_file, '.',
        /extract)
        ; separates off the file extension
tempname = tempname[n_elements(tempname)
        -1]                                                                    ;
        gets the file extension
if tempname eq 'SAV' or tempname eq 'sav' then restore, filename = DEM_load_file else
        rd_genx, DEM_load_file, DEM, text=NOTE                          ; restores
        IDL .SAV files or reads .GENX files
if keyword_set(DEM) then
        begin                                                          ; checks to
        make sure the DEM structure exists in the loaded file
  check = tag_names(DEM)
  if where(check eq 'DEM') ne -1 then begin
    if array_equal(dem.line, manip.lines.chianti) ne 1 then a =
```

```
        DIALOG_MESSAGE('WARNING: Loaded DEM is for different set of
        lines!')                ; provides warning message if line information in the loaded
        DEM does not match the current working lines
    if array_equal(DEM.logt, manip.bins) ne 1 then a =
        DIALOG_MESSAGE('WARNING: Loaded DEM has different temperature
        resolution or range!')          ; provides warning message if the temperature array
        resolution is different in the file
    if keyword_set(header) then NOTE = [NOTE,
        header]                                                 ; if the file
        contains a header, adds it to the notational information
    DEM_load_file = strsplit(DEM_load_file, '/',
        /extract)                                               ; separates the
        loaded filename from the path
    DEM_load_file = DEM_load_file[n_elements(DEM_load_file) -
        1]                                                      ; sets the loaded filename
    ECHIDNA_info_box = widget_base(title = 'Loaded DEM Info - ' +
        DEM_load_file)                                          ; creates a base widget
        to display the information
    INFObox = widget_text(ECHIDNA_info_box, value = NOTE, /wrap, xsize = 65, ysize
        = 12, /scroll)                                  ; creates a text widget in the base
        widget in which to display the information
    widget_control, ECHIDNA_info_box,
        /realize                                                ; realizes
        the widgets - this displays the notational information contained within the loaded
        DEM file
    manip.dem = dem.DEM                                         ; sets
        the new DEM
    manip.bins = dem.logt                                       ; sets the
        new temperature array (in case it is different from the previous)
    widget_control, eventop, set_uvalue = manip                 ;
        places the manip structure, with the new DEM, into the uvalue of the top level
        widget
    manip = ECHIDNA_plot(manip,
        eventop)                                                ; replots the DEM
  endif else a = DIALOG_MESSAGE('FILE ERROR: File does not contain recognizable
        DEM curve.')
endif else a = DIALOG_MESSAGE('FILE ERROR: File does not contain recognizable
        DEM curve.')            ; displays error message if the file does not contain DEM
        and temperature arrays
LOADEND:
return, manip
end
```

```
pro ECHIDNA_save_event, ev
; event handler routine for ECHIDNA_save function - uses information from the save
      menue and prompts for filename to save DEM and line data

widget_control, ev.top, get_uvalue = manip                    ; fetches the manip structure
      from he top level widget's uvalue
widget_control, ev.id, get_uvalue = uval                      ; fetches the uvalue of the
      widget generating the even (either cancel or accept)
if uval eq 'CANCEL' then widget_control, ev.top, /destroy     ; ends routine when
      cancel is clicked
if uval eq 'ACCEPT' then begin                                ; starts process if accept is
      clicked
 textID = widget_info(ev.top, find_by_uname = 'INFOBOX')      ; fetches the widget
      id of the text widget containing any notational information entered by the user
 selectionID = widget_info(ev.top, find_by_uname = 'FILE_TYPE')   ; fetches the
      widget id of the selector for the type of file being created
 widget_control, textID, get_value = note                     ; puts contents of the notation
      widget into the variable 'note'
 widget_control, selectionID, get_value = selection           ; gets the type of file being
      created
 ; if informational tags already exist, passes them to the NOTE array
 if where(tag_names(manip.lines) eq 'INSTRUMENT') ne -1 then  NOTE = [NOTE,
      'INSTRUMENT: ' + manip.lines[0].instrument, 'ABUND FILE: ' +
      manip.lines[0].abundance_filename, 'IONEQ FILE: ' +
      manip.lines[0].ioneq_filename, 'ELEC DENSITY: ' +
      strcompress(manip.lines[0].density)] else NOTE = [NOTE, 'Missing Further Info.']
 if selection eq 0 then fil = ['*.SAV;*.sav'] else if selection eq 1 then fil =
      ['*.GENX;*.genx;*.GenX;*.genX;*.GENx']       ; selection 0 sets the save file
      filter to .GENX type, 1 makes a .SAV file
 savename = dialog_pickfile(title = 'DEM file to save?', /write, /overwrite_prompt, filter
      = fil, path = manip.genx_path)      ; prompts for name of file to be saved
 if savename ne '' then
      begin                                                   ; checks to
      make sure a file name has been entered
  savename2 = strsplit(savename, '.',
      /extract)                                               ; separates off the file
      extension from indicated file name
  savename3 =
      savename2[0]                                                ; sets
      the first part of the save file
  if n_elements(savename2) gt 2 then for i = 1, n_elements(savename2) - 2 do
      savename3 = [savename3, '.', savename2[i] ]      ; rejoins the rest of the file, minus
      the file extension (in case a previous file has been selected)
```

```
   savename3 = strjoin(savename3)
   savename1 = strsplit(savename3, '_',
       /extract)                                                    ; separates the file name
       into portions that may contain the '_DEM' indicator
   if savename1[n_elements(savename1) - 1] eq 'DEM' then
       begin                                                        ; determines the existence of the
       '_DEM' indicator in the file name and removes if found, in case of overwriting a
       previous file
     savename3 = savename1[0]
     if n_elements(savename1) gt 2 then for i = 1, n_elements(savename1) - 2 do
       savename3 = [savename3, '_', savename1[i]]
   endif
   savename3 =
       strjoin(savename3)                                                    ;
       puts filename back together and removes resulting whitespace
   savename3 = strcompress(savename3)
   if selection eq 0 then savename = savename3 + '_DEM.SAV' else if selection eq 1 then
       savename = savename3 + '_DEM.GENX'      ; adds the '_DEM' indicator to
       filename and appropriate extension
   header = 'Saved DEM information. Created by ECHIDNA on ' + systime() + ' from the
       file ' + manip.genx_name + '.'              ; creates file header containing the time and
       filename of the lines data file
   DEM = {DEM:manip.DEM, EM:(manip.DEM + alog10(manip.binwidths)),
       LOGT:manip.bins, LINE:manip.lines.chianti, RATIOS:manip.ratio_y,
       RATIO_ERRORS:manip.ratio_err, CHI_SQUARED:manip.chi_squared}         ;
       creates structure for DEM data, including the CHIANTI identification of the lines
   if selection eq 0 then save, filename = savename, header, DEM,
       NOTE                                                 ; invokes appropriate save procedure for
       the file type being created
   if selection eq 1 then wrt_genx, savename, DEM, text= [header, NOTE], /xdr, /replace
   print, 'Saving DEM structure in ' + savename +
       '.'                                                          ; indicates the save is being
       completed
 endif
 widget_control, ev.top, /destroy
endif

end




function ECHIDNA_save, manip, groupleader
```

```
; function to save DEM curve in a structure in a file - this section creates the widgets and
        calls the event routine

; defines the widgets in the save menu
SAVEbase = widget_base(/column, uvalue = manip, group_leader = groupleader, /modal)
file_type_label = widget_label(SAVEbase, value = 'Select type of save file.')
file_type = cw_bgroup(SAVEbase, ['IDL save file', 'GENX file'], uvalue = 'FILE_TYPE',
        uname = 'FILE_TYPE', /exclusive)
infobox_label = widget_label(SAVEbase, value = 'Enter any additional notes here.')
infobox = widget_text(SAVEbase, uvalue = 'INFOBOX', uname = 'INFOBOX', /editable,
        ysize = 3)
enter_button = widget_button(SAVEbase, value = 'ACCEPT', uvalue = 'ACCEPT',
        uname = 'ACCEPT')
cancel_button = widget_button(SAVEbase, value = 'CANCEL', uvalue = 'CANCEL',
        uname = 'CANCEL')

; realizes the widgets and starts the handler routine
widget_control, SAVEbase, /realize
xmanager, 'ECHIDNA_save', SAVEbase
return, 1
end




function ECHIDNA_reset_mode, manip, eventtop
; checks to see what mode is selected and resets to that mode after a transient mode
        opperation
; requires manipulation structure and id of top level widget

  MESSAGEtext = widget_info(eventtop, find_by_uname = 'MESSAGEtext')              ;
        finds widget id of the message box
  MODElist = widget_info(eventtop, find_by_uname = 'MODElist')                    ;
        finds widget id of the mode list box
  ind = widget_info(MODElist, /list_select)                                       ; gets current
        selection of the mode list
  case ind of                                                                     ; sets the mode based
        on selected index of mode list
    0 : manip.mode = 'CLICK'
    1 : manip.mode = 'BINDRAG'
    2 : manip.mode = 'DRAW'
  endcase
  case ind of                                                                     ; gets text for message
```

```
        box based on mode list selection
   0 : newtext = manip.message.CLICK
   1 : newtext = manip.message.BINDRAG
   2 : newtext = manip.message.DRAW
  endcase
  widget_control, MESSAGEtext, set_value = newtext                    ; sets message
        box text
  widget_control, eventtop, set_uvalue = manip                        ; sends
        manipulation structure back to top level widget uvalue
  return, manip                                                  ; sends back
        manipulation structure
end




pro ECHIDNA_EVENT, ev
; event handling procedure for the program - determines what manipulation is being done,
        and which instructions are being issued by user
; interprets menus and pointer activity
; procedure is called whenever an event is detected in a widget

;print, ev.id    ###REMOVE
widget_control, ev.TOP, get_uvalue = manip          ; retrieves the manipulation
        structure from the uvalue of the top level widget
widget_control, ev.ID, get_uvalue=uval              ; gets the identity of which
        widget has been activated - whether plot window or control

DEMdraw = widget_info(ev.top, find_by_uname = 'DEMdraw')        ; finds id of the
        DEM plot window
RATIOdraw = widget_info(ev.top, find_by_uname = 'RATIOdraw')            ; finds
        id of the RATIO plot window
widget_control, DEMdraw, get_value = DEMdraw                    ; finds plot
        window value of the DEM plot
widget_control, RATIOdraw, get_value = RATIOdraw                ; finds plot
        window value of the RATIO plot
manip.DEMdraw = DEMdraw                                                 ; puts
        plot window value of DEM plot in manipulation structure
manip.RATIOdraw = RATIOdraw                                     ; puts plot
        window value of RATIO plot in manipulation structure

if where(tag_names(ev) eq 'RELEASE') ne -1 then begin      ; This is here, since the !
        MOUSE structure does not respond until the following event
```

```
  if ev.release gt 0 then begin                    ; This keeps track of the current state of the
        mouse button, and though it is unfortunately messy and complicated, seems to
        work ok.
    manip.clicked = 0
    widget_control, ev.top, set_uvalue = manip
  endif
endif

if uval eq 'MODElist' then begin                   ; calls the mode reset function if the
        mode list has been changed
  manip = ECHIDNA_reset_mode(manip, ev.TOP)
endif
if uval eq 'LOAD' then begin          ; if the load button is clicked, calls the load function
  manip = ECHIDNA_load_DEM(manip, ev.TOP)
endif
if uval eq 'ZERO' then begin                       ; sets the DEM array to baseline value
        if ZERO DEM button clicked
  manip.last_DEM = manip.dem                                ; puts current DEM in
        old DEM array for undo function
  DEM = fltarr(n_elements(manip.DEM))                       ; creates a new DEM
        array full of zeros
  DEM = DEM + manip.baseline                                ; adds the baseline
        value to new DEM array
  manip.DEM = DEM                                  ; puts new DEM array in the
        manipulation structures DEM entry
  manip = ECHIDNA_plot(manip, ev.TOP)                       ; calls plot function
        with new zeroed DEM
endif

if uval eq 'ZERObin' then begin                           ; in the event the zero
        DEM button clicked
  if manip.mode ne 'ZERObin' then begin
    manip.mode = 'ZERObin'                                  ; sets the mode to zero
        the DEM in a temperature bin
    MESSAGEtext = widget_info(ev.top, find_by_uname = 'MESSAGEtext')       ; gets
        message text to indicate what the next click will do
    widget_control, MESSAGEtext, set_value = manip.message.ZEROBIN         ; sets the
        message box to this text
    widget_control, ev.top, set_uvalue = manip                      ; puts the
        manipulation array in the top level widget's uvalue
  endif else manip = ECHIDNA_reset_mode(manip, ev.TOP)
endif

if uval eq 'UNDO' then begin                       ; in the event the undo button is
```

```
        clicked
  switch_dems = manip.dem                                   ; stores the current DEM array
  manip.dem = manip.last_dem                                ; sets current DEM
        array as previous DEM array
  manip.last_dem = switch_dems                              ; sets the old DEM
        array as stored DEM
  manip = ECHIDNA_plot(manip, ev.TOP)                       ; calls plot function
        with new DEMs
endif

if uval eq 'BINS' then begin                            ; in the event the show bins button is
        clicked
  if manip.SHOWbins eq 0 then manip.SHOWbins = 1 else manip.SHOWbins = 0          ;
        swaps the show bins tag value in the manipulation array
  manip = ECHIDNA_plot(manip, ev.TOP)                                             ; calls
        plot function with new tag value
endif

if uval eq 'INFO' then a = DISPLAY_INFO(manip)

if uval eq 'YRANGEmin' then begin                       ; in the event a new minimum y range
        value entered
  widget_control, ev.id, get_value = ymin               ; gets the new value
  manip.ymin = ymin                                     ; sets the current value to the new
        value
  manip = ECHIDNA_plot(manip, ev.TOP)                   ; calls plot function with new
        value
endif
if uval eq 'YRANGEmax' then begin                       ; in the event a new maximum y
        range value enteredMODElist = widget_list(CONTROLcontainer, value =
        ['CLICK', 'DRAG BIN', 'DRAW'], uvalue = 'MODElist', uname = 'MODElist', ysize
        = 3)

  widget_control, ev.id, get_value = ymax               ; gets the new value
  manip.ymax = ymax                                     ; sets the current value to the new
        value
  manip = ECHIDNA_plot(manip, ev.TOP)                   ; calls plot function with the
        new value
endif

if uval eq 'BASELINEUP' or uval eq 'BASELINEDOWN' then begin       ; in the event
        the baseline up or down buttons clicked
  blindex = where(manip.DEM eq manip.baseline)                      ; finds indices of all
        bins where DEM value is at baseline
```

```
  if uval eq 'BASELINEUP' then begin          ; if baseline up was clicked, then add 1 to all
        DEM values at baseline and to the baseline value
    manip.baseline = manip.baseline + 1
    if blindex[0] ne -1 then for i = 0, n_elements(blindex) - 1 do manip.DEM[blindex[i]] =
        manip.baseline
  endif
  if uval eq 'BASELINEDOWN' then begin  ; if baseline down was clicked, subtract 1
        from all DEM values at baseline and to baseline value
    manip.baseline = manip.baseline - 1
    if blindex[0] ne -1 then for i = 0, n_elements(blindex) - 1 do manip.DEM[blindex[i]] =
        manip.baseline
  endif
  widget_control, ev.top, set_uvalue = manip                    ; puts mainipulation
        array in uvalue of top level widget
  Blabel = widget_info(ev.top, find_by_uname = 'BASELINElabel')          ; finds
        id of baseline value label
  widget_control, Blabel, set_value = strcompress(manip.baseline) ; sets the baseline
        value label to the new baseline
  manip = ECHIDNA_plot(manip, ev.TOP)                           ; calls plot
        function with new baseline value
endif

if uval eq 'PS' then begin                  ; routine for creating a postscript file from the plots
  ps_name = dialog_pickfile(Title = 'Name of Postscript file?', /write, /overwrite_prompt,
        path = manip.genx_path, filter = '*.eps;*.EPS')       ; prompts for postscript file
        name
  if ps_name ne '' then
        begin                                                                    ;
        only executes if a non-null filename is entered
    tempname = strsplit(ps_name, '.', /extract)
    if n_elements(tempname) lt 2 then tempname = [tempname, '.EPS'] else
        tempname[n_elements(tempname) - 1] = '.EPS'          ; prepares the postscript file
        name for the DEM plot
    ps_ratios = tempname                                                         ;
        prepares the postscript file name for the ratio plot
    ps_ratios[n_elements(ps_ratios) - 1] = '_RATIOS.EPS'
    ps_ratios = strcompress(strjoin(ps_ratios))
    ps_name = strcompress(strjoin(tempname))
    SET_PLOT, 'PS'                                          ; sets the plot to postscript
    DEVICE, FILE=ps_ratios, /landscape, /medium, /COLOR, BITS=8             ; sets the
        plotting device to the FIRST file being created, the ratio plot
    manip = ECHIDNA_plot(manip, ev.TOP, /displayonly, ps=ps_name)           ; calls the
        plot function with no alteration of the plot data, and with the postscript name
    device, /close                                          ; finalizes the file
```

```
    set_plot, 'x'                                   ; returns the plot to the screen
    print, 'DEM plot saved to ' + ps_name + '.'
    print, 'Ratio plot saved to ' + ps_ratios + '.'
  endif
endif

if uval eq 'PRINT' then begin                       ; something seems to be
        wrong with IDL's control of the printer - not yet functional  ###REMOVE
  SET_PLOT, 'PRINTER'
  manip = ECHIDNA_plot(manip, ev.TOP, /displayonly, /ps)
  SET_PLOT, 'X'
endif

if uval eq 'SAVE' then begin                        ; calls the save function when save is
        clicked
  widget_control, ev.top, sensitive = 0
  a = ECHIDNA_save(manip, ev.top)
  widget_control, ev.top, sensitive = 1
endif

if uval eq 'PLOTTYPE' then begin
  plottype_id = widget_info(ev.top, find_by_uname = 'PLOTTYPE')
  widget_control, plottype_id, get_value = plottype
  manip.plottype = plottype
  widget_control, ev.top, set_uvalue = manip
  manip = ECHIDNA_plot(manip, ev.TOP)
endif

if uval eq 'QUIT' then begin
  QUITECHIDNA = ECHIDNA_YorN(title = 'QUIT?', msg = 'Are you sure you want to
        quit ECHIDNA?', eventop = ev.TOP)           ; if QUIT button clicked, asks
        whether to quit
  if QUITECHIDNA eq 'YES' then widget_control, ev.top, /destroy    ; if YES is selected
        to quit, kills all widgets
endif

if uval eq 'DEMdraw' then begin  ; indicates DEM plot was where the event originated
  if ev.press eq 1 then begin              ; if event is that the moust button was pressed,
        before it was released
    manip.clicked = 1
    manip.last_DEM = manip.dem               ; changes previous DEM to current DEM for
        undo function
    cursor, x1, y1, /nowait, /data  ;  This is very ill advised.  However, if the plot function
        plots the DEM curve last, then it should be the current direct graphics window, and
```

```
      the CURSOR procedure should work.
  bin_index1 = where(  min( abs(x1 - manip.bins) )  eq  abs( x1 - manip.bins)  ) ;
      determines which temperature bin has been clicked in
  if manip.plottype eq 1 then y1 = y1 - alog10(manip.binwidths[bin_index1])
  if manip.mode eq 'BINDRAG' or manip.mode eq 'DRAW' then manip.storex =
      bin_index1
  if n_elements(bin_index1) gt 1 then bin_index1 = bin_index1[0]          ; if for some
      reason it identifies two adjacent temperature bins, it just takes the lower one
    manip.DEM[bin_index1] = y1                                            ; sets the y value
      retrieved from the plot window event as the new DEM value for that temp bin
    manip = ECHIDNA_plot(manip, ev.TOP)                                   ; calls plot
      function with new DEM value
  if manip.mode eq 'ZERObin' then begin                                   ; when mode is to zero
      a bin, sets that bin's DEM value to baseline and calls plot function
    manip.DEM[bin_index1] = manip.baseline
    manip = ECHIDNA_plot(manip, ev.TOP)
  endif
endif

; Following is a separate section since the !MOUSE structure does not respond until the
      following event.  It is necessary to keep track of the current state of the mouse
      button in the context of the program, rather than use the system variables.
if manip.mode eq 'BINDRAG' or manip.mode eq 'DRAW' then begin ; when mode is
      something other than just to click
  bin_index_current = manip.storex          ; sets a current bin according to stored x value
  if manip.clicked eq 1 and ev.release ne 1 then begin        ; as long as the mouse button
      is still down (indicated by the clicked tag not being reset to 0 by a mouse release
      event), does the following
    cursor, x2, y2, /nowait, /data      ;  See above note on use of CURSOR.
    if manip.mode eq 'DRAW' then bin_index_current = where(  min( abs(x2 -
      manip.bins) )  eq  abs( x2 - manip.bins)  )  ; resets current bin index only if mode is
      draw
    if manip.plottype eq 1 then y2 = y2 - alog10(manip.binwidths[bin_index_current])
    if n_elements(bin_index_current) gt 1 then bin_index_current =
      bin_index_current[0] ; if current bin index was identified as two adjacent bins,
      takes the lower one
    manip.dem[bin_index_current] = y2       ; sets the DEM value in the current bin
      (whether or not it has been changed because of draw mode)
    manip = ECHIDNA_plot(manip, ev.TOP)       ; calls plot function with new DEM
      value
  endif
endif

if manip.mode eq 'GROUP' then begin                ; NOT YET IMPLIMENTED
```

```
 endif

 if manip.mode eq 'UNGROUP' then begin            ; NOT YET IMPLIMENTED
 endif

 if ev.release eq 1 and manip.mode eq 'ZERObin' then manip =
       ECHIDNA_reset_mode(manip, ev.TOP)                ; resets mode only after a
       mouse button release (so draw or drag won't kick in)

endif

end




pro ECHIDNA
; main procedure - reads files, creates structures and defines and calls widgets

set_plot, 'X'
!p.multi = [ 0,0,0,0,0]
genx_name = dialog_pickfile(filter = ['*.genx;*.GENX;*.GenX;*.genX;*.GENx'], Title =
       'Select .GENX file containing spectral line information.', get_path = genx_path)    ;
       prompts for genx file
if not(keyword_set(genx_name)) then goto, ECHIDNA_END
         ; ends in the event a file was not selected
rd_genx, genx_name, lines, text = text, header = header
                          ; reads the genx file

genx_name = strsplit(genx_name, '/', /extract)                    ; extracts the filename
       from the path
genx_name = genx_name[n_elements(genx_name) - 1]
genx_name = strcompress(genx_name)

lines = ECHIDNA_arrange_lines(lines)                              ; calls the function to
       arrange the lines in the lines structure

binstep = abs(lines[0].logt[1] - lines[0].logt[0])    ; finds difference between temperature
       bins - they must be evenly spaced for this to work right
binwidths = make_array(n_elements(lines[0].logt))    ; finds width of temperature bins
for i = 0, n_elements(lines[0].logt) - 1 do begin    ; creates array of temperature width for
       each bin - temperature differences are log, but bins are made linearly in log space
 binwidths[i] = ( 10^(lines[0].logt[i] + (binstep/2) ) ) - ( 10^(lines[0].logt[i] -
```

```
        (binstep/2) ) )
endfor

message = {CLICK:'Click in the DEM plot window to set a new point.',
        BINDRAG:'Click and drag individual bins up or down.', DRAW:'Click and draw
        DEM curve.', ZEROBIN:'Click in a bin to set its value to the baseline.'}      ; the text
        for the message window
manip = {lines:lines, DEM:fltarr(n_elements(lines[0].logt)),
        LAST_DEM:fltarr(n_elements(lines[0].logt)),
        predicted:make_array(n_elements(lines)),
        observed:make_array(n_elements(lines)),
        bins:make_array(n_elements(lines[0].logt)), DEMdraw:0, RATIOdraw:0,
        SHOWbins:0, BASELINE:18, MODE:'CLICK', ymin:17, ymax:26,
        message:message, header:header, text:text, genx_name:genx_name,
        genx_path:genx_path, ratio_y:fltarr(n_elements(lines)),
        ratio_err:fltarr(n_elements(lines)), chi_squared:double(0), storex:0.0, storey:0.0,
        clicked:0, plottype:0,binwidths:binwidths }              ; creates manip structure

for i = 0, n_elements(manip.lines) - 1 do manip.observed[i] = manip.lines[i].iobs ; puts
        observed values into structure
for i = 0, n_elements(manip.dem) - 1 do manip.dem[i] = manip.baseline                    ;
        initializes DEM values
manip.last_dem = manip.dem                                                               ;
        initializes previous DEM values
manip.bins = manip.lines[0].logt                                                         ; sets
        bins to the log of temp values

;  Defines widgets for holding controls and displays
scrsize = get_screen_size()                ; Determines the size of the screen
xsize = ( scrsize[0] -415 ) / 2            ; Determines the x size of each of the two plot
        windows in terms of screen size
ysize = xsize                ; Determines the ysize of the two plot windows in terms of screen
        size
menuwidth = scrsize[0] / 20
ECHIDNAbase = widget_base(title = 'DEM Editor - '+ genx_name, xoffset=200,
        yoffset=200, column=3, uvalue = manip, /base_align_center)  ;  main base widget
        defined
CONTROLcontainer = widget_base(ECHIDNAbase ,frame=5, /column, xsize = 115)
DEMcontainer = widget_base(ECHIDNAbase, /column, /grid_layout)
RATIOcontainer = widget_base(ECHIDNAbase, /column, /grid_layout)

; the list of modes
MODElist = widget_list(CONTROLcontainer, value = ['CLICK', 'DRAG BIN', 'DRAW'],
        uvalue = 'MODElist', uname = 'MODElist', ysize = 3)
```

```
; Chooses type of plot - which value is being plotted on the Y axis
plot_type = cw_bgroup(CONTROLcontainer, ['DEM Plot', 'EM Plot'], uvalue =
      'PLOTTYPE', uname = 'PLOTTYPE', set_value = 0, /exclusive)

; these are mode control buttons
UNDObutton = widget_button(CONTROLcontainer, value = 'UNDO', uvalue = 'UNDO')
ZEROBINbutton = widget_button(CONTROLcontainer, value = 'ZERO 1 BIN', uvalue =
      'ZERObin')
BINSbutton = widget_button(CONTROLcontainer, value = 'SHOW BINS', uvalue =
      'BINS')
INFObutton = widget_button(CONTROLcontainer, value = 'SHOW DATA INFO',
      uvalue = 'INFO')

; defines widgets to display and enter y range values
YRANGEcontainer = widget_base(CONTROLcontainer, frame = 5, /column,
      /base_align_center)
YRANGElabel = widget_label(YRANGEcontainer, value = 'Y-RANGE', /align_center)
YRANGEmax = cw_field(YRANGEcontainer, title = 'MAX', uvalue = 'YRANGEmax',
      value = manip.ymax, uname = 'YRANGEmax', xsize = 3, /long, /return_events)
YRANGEmin = cw_field(YRANGEcontainer, title = 'MIN', uvalue = 'YRANGEmin',
      value = manip.ymin, uname = 'YRANGEmin', xsize = 3, /long, /return_events)

; defines widgets to display and change the baseline value
BASELINEcontainer = widget_base(CONTROLcontainer, frame = 5, column=1, uvalue
      = manip.baseline)
BASELINEUPbutton = widget_button(BASELINEcontainer, value = 'RAISE
      BASELINE', uvalue = 'BASELINEUP')
BASELINElabel = widget_label(BASELINEcontainer, value =
      strcompress(manip.baseline), uvalue = 'BASELINElabel', uname =
      'BASELINElabel')
BASELINEUPbutton = widget_button(BASELINEcontainer, value = 'LOWER
      BASELINE', uvalue = 'BASELINEDOWN')

; these are more buttons
LOADDEMbutton = widget_button(CONTROLcontainer, value = 'LOAD DEM', uvalue
      = 'LOAD')
SAVEDEMbutton = widget_button(CONTROLcontainer, value = 'SAVE DEM', uvalue =
      'SAVE')
MAKEPSbutton = widget_button(CONTROLcontainer, value = 'MAKE
      POSTSCRIPTS', uvalue = 'PS')
PRINTbutton = widget_button(CONTROLcontainer, value = 'PRINT / PDF', uvalue =
      'PRINT', sensitive = 0)
ZERODEMbutton = widget_button(CONTROLcontainer, value = 'ZERO DEM', uvalue
```

```
        = 'ZERO')
QUITbutton = widget_button(CONTROLcontainer, value = 'QUIT', uvalue = 'QUIT')

; defines a message box to display current action
MESSAGEtext = widget_text(CONTROLcontainer, value = manip.message.CLICK,
        uname = 'MESSAGEtext', ysize = 5, xsize = 16,/align_left, /wrap)  ;,
        /dynamic_resize, /sunken_frame, frame = 2    xsize = 112,

; defines the plotting window widgets
DEMdraw = widget_draw(DEMcontainer, xsize = xsize, ysize = ysize, uvalue =
        'DEMdraw', uname = 'DEMdraw', /BUTTON_EVENTS, /MOTION_EVENTS)
        ;xsize = 800, ysize = 800,    , xsize = 1000, ysize = 800
RATIOdraw = widget_draw(RATIOcontainer, xsize = xsize, ysize = ysize, uvalue =
        'RATIOdraw', uname = 'RATIOdraw')  ;, xsize = 1000, ysize = 800

;  Realizes the widgets
widget_control, ECHIDNAbase, /realize

; sets initial modes and plots initial flat DEM curve
widget_control, MODElist, set_list_select = 0
widget_control, DEMdraw, get_value = DEMdraw
widget_control, RATIOdraw, get_value = RATIOdraw
manip.DEMdraw = DEMdraw
manip.RATIOdraw = RATIOdraw
widget_control, ECHIDNAbase, set_uvalue = manip
manip = ECHIDNA_plot(manip, 0, /displayonly)    ; plots the initial DEM, but does not
        place the manip array in the top level widget's uvalue, as it is not yet running

; calls the handler routine
xmanager, 'ECHIDNA', ECHIDNAbase

;dem_out = demacs(lines[0].logt)

ECHIDNA_END:                ; goto point to exit program

end
```