Development of a solar imaging array of Very Small Radio Telescopes

Ted Tsiligaridis
University of Washington, Seattle
Alan E. E. Rogers
MIT Haystack Observatory

ABSTRACT

A 3-element interferometer built using very small radio telescopes (VSRT's) was used for observing the Sun at a frequency band of 35 MHz wide centered at 12.086 GHz daily. The signals from the LNBF's are processed with a USB 2.0 video grabber and other inexpensive consumer electronics. The baselines used were based on a reference between the two adjacent dishes and thus, the baselines of the first two dishes are the same; 0.2699 meters, and the distance from that center point to the third dish is 2.9464 meters. Data from the Sun was collected over a two month period and software was used to calculate the fringe visibility amplitudes and closure phase. Modeling programs were developed to imitate the Sun's chromospheric behavior and a comparison of the closure phase between the VSRT data and the model's curves was achieved via least-squares analysis and construction of surface plots to display the sensitivity of the model as different parameters varied, which would yield solutions to the least squares problem. Various approaches were taken to examine the sunspot activity. The software was developed in two languages; Matlab and Python, and the motivation for using Python was to make the software system accessible to community colleges and introduce radio interferometry, since it's freely downloadable. The cardinal factors that contribute to the visibility of the model include the uniform Sun component, a linear brightness gradient applied to simulate dish mispointing, limb brightening and a changeable sunspot of varying intensity. We found that the more sophisticated limb brightening (adding many rings) tended to fit the actual data a lot better than the previous limb brightening profile did (only one outer ring). Although dish mispointing made it difficult to infer information about the Sun at times and the sun was not showing much activity during the summer, we were not only successfully able to detect sunspot activity using the 3-element VSRT system, but also observed a decaying trend of the sunspot's intensity over a period of six days. The hardware of the 3-element VSRT system was also documented in detail.

1 Introduction

The purpose of this paper is to describe the VSRT hardware system and explore the ways in which the VSRT's can be used for solar research. They're still being developed and it's interesting to see what can be achieved with such small dishes and a smart 3-element interferometer setup. The closure phase was observed and deviations from ideal conditions were explained.

In the following, we give an introduction on solar physics, then describe the VSRT observing hardware system, give a description of the concepts used for modeling the Sun's radio image, describe the software developed and the tools used to analyze the data collected. Finally, we discuss the results.

2 Solar Physics Background

2.1 General Physics of the Sun

The Sun can be assumed to give out radiation in three main parts; the quiet sun component (always present), the slowly varying component and the active sun component (caused by sunspots, flares). The quite sun component arises from thermal emission from the hot ionized gas. The effect of all elements other than hydrogen on the mean degree of ionization is small. These effects can be neglected and the solar atmosphere is regarded as a fully ionized hydrogen gas.

To understand from which part of the Sun's atmosphere this emission arises, one needs to consider the main opacity source at radio wavelengths. (The opacity is the measure of how much a wave gets absorbed as it travels through a medium.) The main source of opacity in the Sun's atmosphere at radio wavelengths is *electrons*. The electron density in the chromosphere is represented by an exponential distribution, as suggested by Cillié and Menzel (6), and depends on the height above the photosphere. (The larger the height, the less the intensity of the central beam.) Most of the emission comes from the region where the opacity, τ is near 1, since the higher optical depth regions cannot be penetrated and the low optical depth regions do not produce enough emission. At a frequency of 1.4 GHz (λ = 21 cm), the emission originates from the top of the chromosphere and is seen as a 100,000 °K blackbody. In physics, a blackbody is an object that absorbs all EM radiation falling into it. In other words, the amount of wavelength of EM radiation it emits is directly proportional to its temperature. At longer wavelengths, the emission arises from the corona and is a 2 million °K blackbody. This also means that the size of the sun measured at the different frequencies varies. The lower the frequency, the larger the radius of the sun appears.

The other two components are related to sunspots. The slowly varying component is also thermal in origin and arises from the region above the sunspots where the electron density is higher. The blackbody temperature of these regions can be as high as 2 million °K. Thus, the regions above the sunspots can contribute more radio emission than the total area without sunspots and increase the total radio flux relative to the quite sun. The Sun acts like an extended source and thus will be resolved on the longer baselines. The Sun's atmosphere consists of the photosphere, the chromosphere and the corona. Chromospheric frequencies are referred to those above 10 GHz, as the corona is practically transparent. At chromospheric heights, the temperature distribution is of the order 10^4 °K.

The problem of the transfer of radiation in an ionized gas has been discussed by Smerd and Westfold (7). Considering radiation only in a frequency interval, changes in the intensity of that radiation along a path, due to emission and absorption in an elementary cylinder of the medium of refractive index μ, are related by the equation of transfer. A solution is found under conditions of thermodynamic equilibrium, using the Rayleigh-Jeans radiation formula since we're concerned with radio frequencies, and finally assuming a uniform temperature region. The resulting equation indicates that the emerging intensity is directly related to 1- $e^{-\tau}$. The distribution of electron velocities in the solar atmosphere is assumed to be Maxwellian and brought about by collisions. From the discussion of transfer of RF radiation in the solar atmosphere, to evaluate the intensity of radiation emerging from the solar atmosphere, we require the ray trajectories, the optical depth (opacity) of the medium along any trajectory and the temperature distribution in the solar atmosphere.

2.2 Ray Trajectories

Just like Smerd (4) has analyzed in great detail, to examine ray trajectories, one must consider the paths through the solar atmosphere of those RF rays which can be received at the earth. The refractive index μ of an ionized medium decreases with increasing electron density. As a result, a ray passing through the solar atmosphere experiences continuous bending by refraction. If the propagation of radiant energy is in the direction of decreasing μ (towards the sun), the bending will be away from the normal to a sphere of constant μ. A point is reached where the direction of propagation is tangential to such as sphere after which propagation must be in the direction of increasing μ. The point where the direction of propagation changes from that of decreasing μ to that of increasing μ is the turning point. To aid visualization, trajectories at a frequency which can penetrate into the chromosphere are sketched:

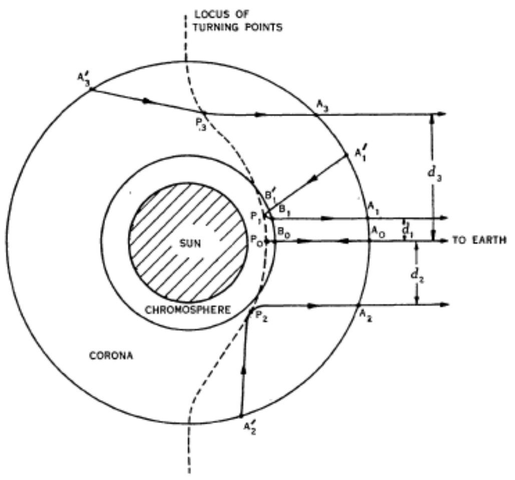


Figure 2.2.1: Ray trajectories at an "intermediate" frequency (after Smerd 1950).

After defining the difference in optical depth of two points on a trajectory as the optical thickness between the two points and taking the absorption coefficient and the refractive index from the Lorentz theory, mathematical descriptions of the optical thickness $\tau_{1,2}(d)$ have been provided by Smerd. The solution of the equation of transfer, along with the physical properties of the solar atmosphere, the ray trajectories and the optical depth, enable us to discover the *intensity* of RF rays emerging from the solar atmosphere. In fact, at a given frequency, rays near the center penetrate deeper into the solar atmosphere than those near the limb, and for a given ray position, a higher frequency penetrates more deeply than a lower one.

According to Smerd's analysis on ray trajectories, it is possible to express the emergent intensity in terms of temperature. After sketching trajectories at a frequency which can penetrate through into the chromosphere (figure 2.2.1), he found that as the distance d, of the trajectory from the center of the disk increases, the optical thickness $\tau_c(d)$ and the effective temperature $T_e(d)$ tend to increase due to the lengthening of the path between the same two heights in the atmosphere and to decrease due to the increasing height of the turning point. Since we're observing at a frequency of 12 GHz (λ = 2.5 cm), according to Smerd, we mostly have a uniform effective temperature as we range across the distance from the center of the disk. This can be inferred from the figure below:

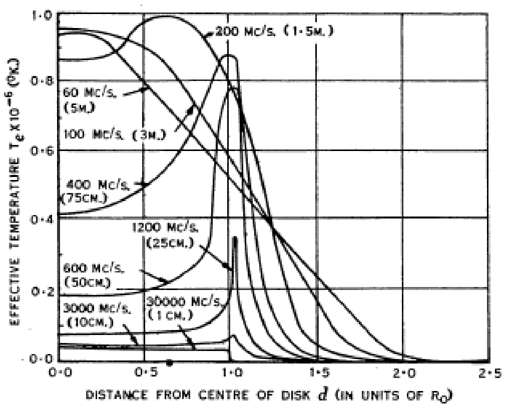


Figure 2.2.2: The computed effective temperature (brightness distributions) over the solar disk at different radio frequencies. The values used for chromospheric and coronal temperatures are 3×10^4 and 10^6 °K respectively (after Smerd 1950).

Since the effective temperature (brightness temperature) at our wavelength tends to be uniform and relatively small, we expect the opacity (optical depth) to decrease as well. Thus, from the solution of the transfer equation (assuming uniform temperature in the chromosphere), it makes sense to expect the intensity to be uniform and relatively smaller than that of other smaller frequencies (low opacity corresponds to low intensity). Another interesting observation can be seen from figure 2.2.2; as the frequency increases, the "size" of the radio disk decreases. This is due to the fact that the optical depth (τ) varies as $1/f^2$.

In summary, at chromospheric frequencies, the effective temperature of any ray within the disk ($0 < d < 1$) is just the chromospheric temperature, since the medium effectively has an infinite optical depth along such trajectories. For rays outside the disk, the optical thickness of the medium (and the effective temperature) falls off rapidly as the distance of the trajectory from the center of the disk increases. The disk at these frequencies has uniform intensity (assuming a uniform chromospheric temperature) and a well defined limb. Limb brightening across the disk would indicate a chromospheric temperature gradient.

3 Hardware System

3.1 Parts

The major parts of the 3-element VSRT system include the dishes, the feeds and low-noise amplifiers. These outdoor components as well as antenna positioning systems, coaxial cables, and connectors are essential.

3.1.1 Antennas

For the VSRT's, we use 45 cm diameter "Direct TV" satellite dishes with their satellite TV low-noise block-down converter feeds (LNBF's) which operate in the 12.2 to 12.7 GHz band. The signals from the LNBF's are processed using a USB 2.0 video grabber along with other inexpensive consumer electronics. The dishes are parabolic in order to reflect radio waves to the subreflector. Since radio-astronomical sources are far away, treating the Sun as a point source, incoming signals tend to look like plane waves and the VSRT must catch as much energy as possible from this wave and avoid as much as possible any other signals, especially local interference. The parabolic antenna concentrates all this energy into a small spot where a feed is placed. A typical antenna has a noise temperature ranging from 20 to 50 K, resulting principally from the ground noise component, which has a temperature of about 290 K. The most critical gain and hence the most critical signal losses in a satellite system occur before the LNB. The dish and the feed system must work as a team to extract the maximum signal. Dish gain is dependent on three factors: surface accuracy, dish pointing accuracy, and the match between the feedhorn and the parabolic surface. If any of these are inadequate, gain will drop rapidly as the error increases.

Mispointing can be caused by several factors. The most common problem is poorly chosen and installed actuators. Torque, cable runs and jack size all affect actuator action. Some positioning systems start and stop the motor at full power. This causes a jerking motion as the antenna begins to move and as it stops. This can shake the structure causing screws and bolts to loosen. Another factor is the structure used. In our case, we had the VSRT dishes installed on a table outside, as shown in figure 3.1.1.1. Rain causes the tables to sink, and then the dishes are mispointed. Then, we need to re-level them by changing the balance again by placing materials under the legs of the table.

The VSRT dishes used are offset fed dishes, which can have much smaller diameters than other satellite dishes. To visualize the geometry of an offset system, picture an oval cut from the upper central portion of a larger parabolic reflector. While the feed remains at the focal point of the larger antenna, only this smaller section of reflector remains. The feedhorn is therefore located below the lower edge of the offset fed reflector. Below is a picture taken of a VSRT dish:

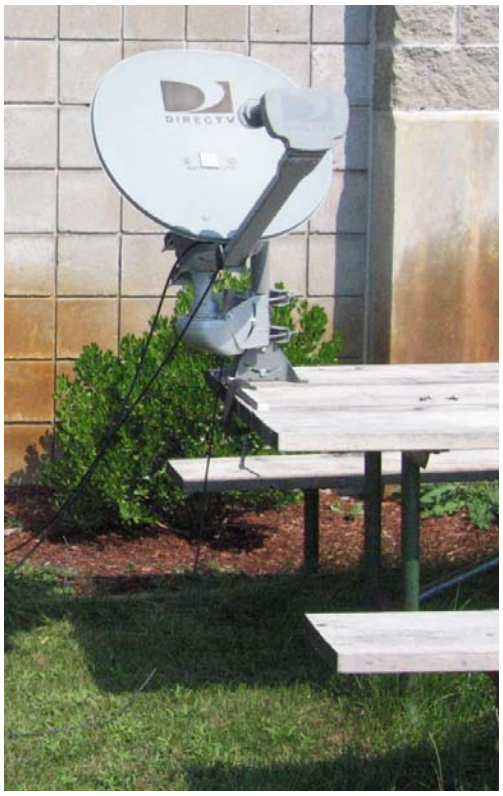


Figure 3.1.1.1: VSRT dish tracking the Sun

3.1.2 LNBF's

Mounted at the dish's focal point is a feedhorn. The feedhorn is the front-end of a waveguide that gathers the signals at the focal point and conducts them to a LNB (low-noise block-down converter), which converts the signals from EM/radio waves to electrical signals and shifts the signals to IF (in our case, the feedhorn is integrated with the LNB). The feeds of the antennas collect the radio power from the distant celestial source. A waveguide is like a hollow piece of coaxial cable and is one of the most efficient methods developed to transport high frequency signals. The transmission of EM energy depends upon reflection of signals by the internal walls of the waveguide as well as upon current transfer along its surface. To properly collect the reflected microwave energy, a feed must ideally "see" or illuminate the entire surface of an antenna and nothing else. If the feed is not matched to the f/D ratio of the dish, then it may see beyond the edge of the dish and allow earth noise to enter. When the f/D ratio and the feed are properly matched and focused, then the dish is optimally illuminated. Figure 3.1.2.1 shows the triple feed LNB, which shows that the dish is being pointed correctly. We can see that the image of the Sun is centered at the third feed, which is an indication that the pointing for this dish is good. Figure 3.1.2.2 shows how the LNBF looks like inside.

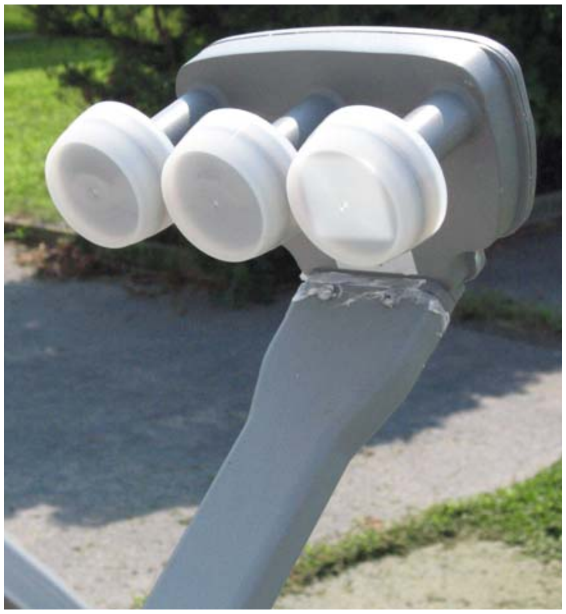


Figure 3.1.2.1: Triple feed LNB is active

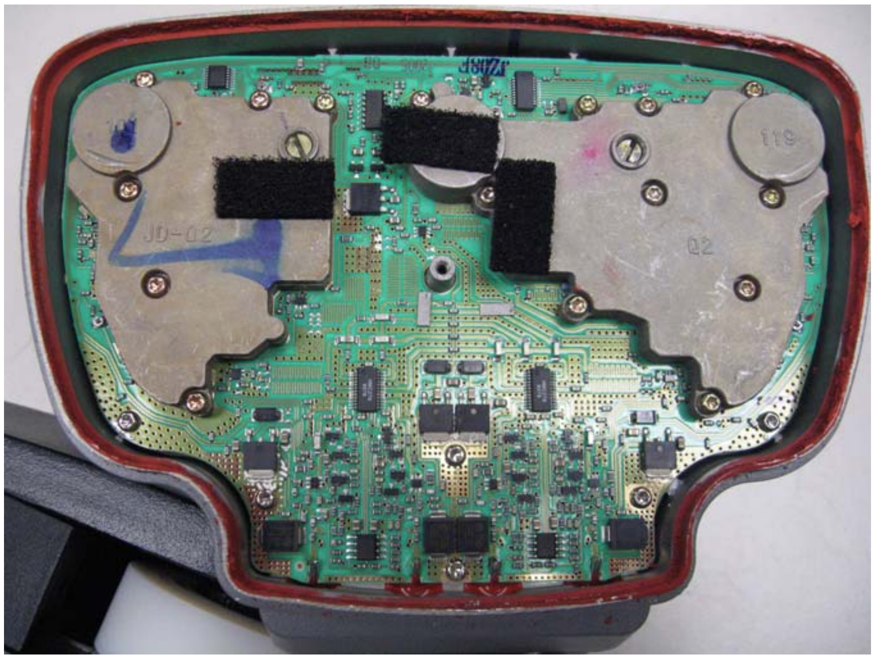


Figure 3.1.2.2: Internal circuit of the LNBF used

3.1.3 Cabling

The three antennas are connected with coaxial cable. The outer jacket protects the cable from moisture, oil, oxidation, ozone, acids, and abrasion. The outer conductor shields the inner conductor from external EM forces and functions as the ground return path. The dielectric core sets up the impedance of the cable and also insulates the center conductor from the shield. The center conductor transports the signals from one end to the other and must make good mechanical conduct at both ends using some type of connector. Below is a picture of the coaxial cabling installed at the dish structure outside:

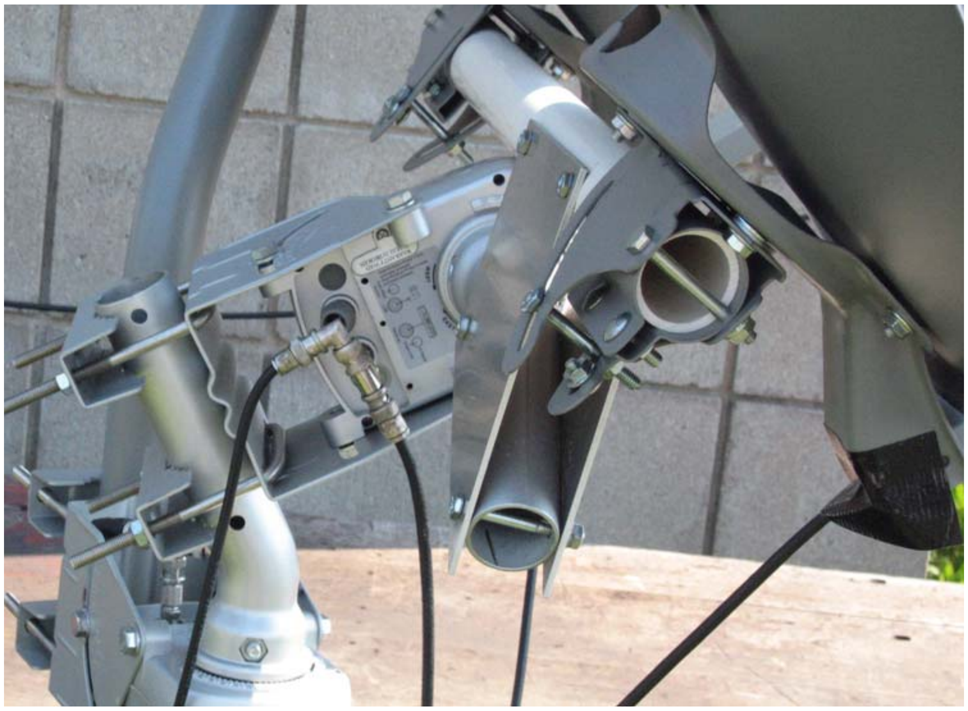


Figure 3.1.3.1: Coaxial cables installed on dish structure

3.2 Simple Block Diagram

Below is a simple block diagram of the 3-baseline VSRT interferometer system developed:

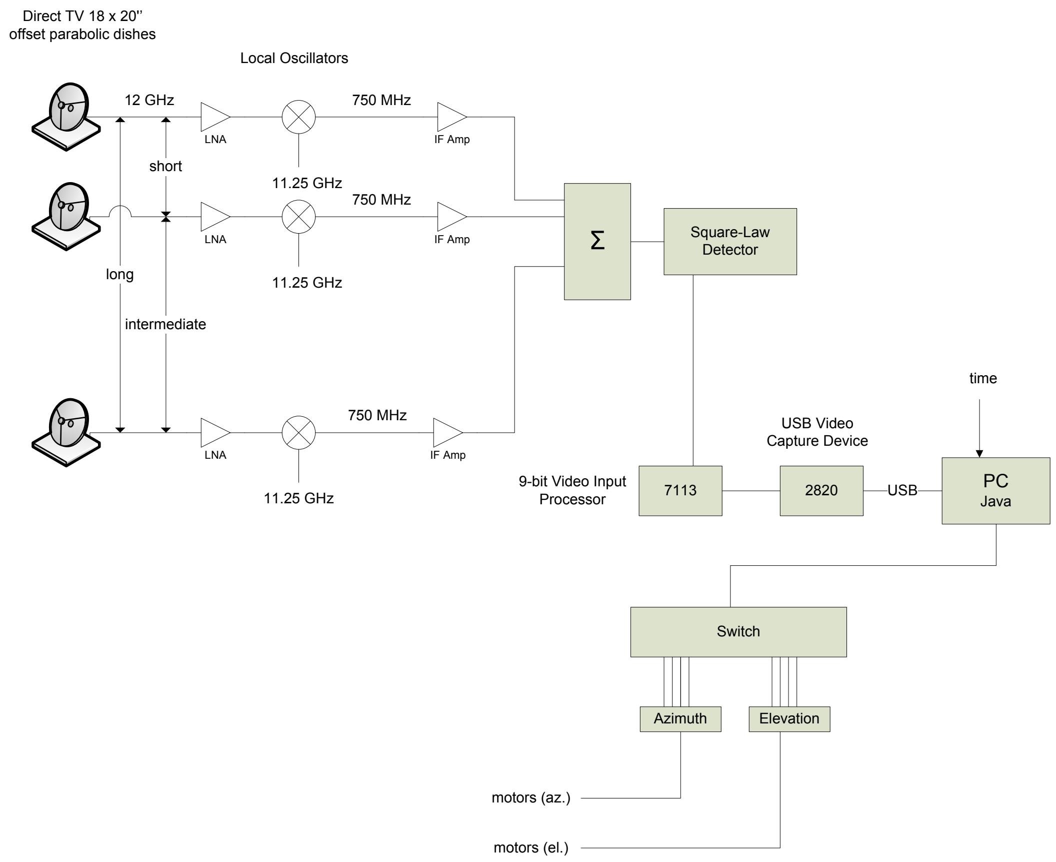


Figure 3.2.1: Simple Block Diagram

The system of coordinates used is an azimuth-elevation system. The azimuth is the horizontal angle measured from north, going clockwise to the object of interest. The altitude (elevation angle) is measured upwards from the horizon to the object. The coordinates of a celestial object in the horizon system change continuously during the day because of the earth's rotation. It is convenient to use elevation and azimuth angles in setting the VSRT's because it is steerable around vertical and horizontal axis. Below is a picture of the VSRT rotation system controlling the two adjacent dishes. The azimuth and the elevation motors can be clearly seen.

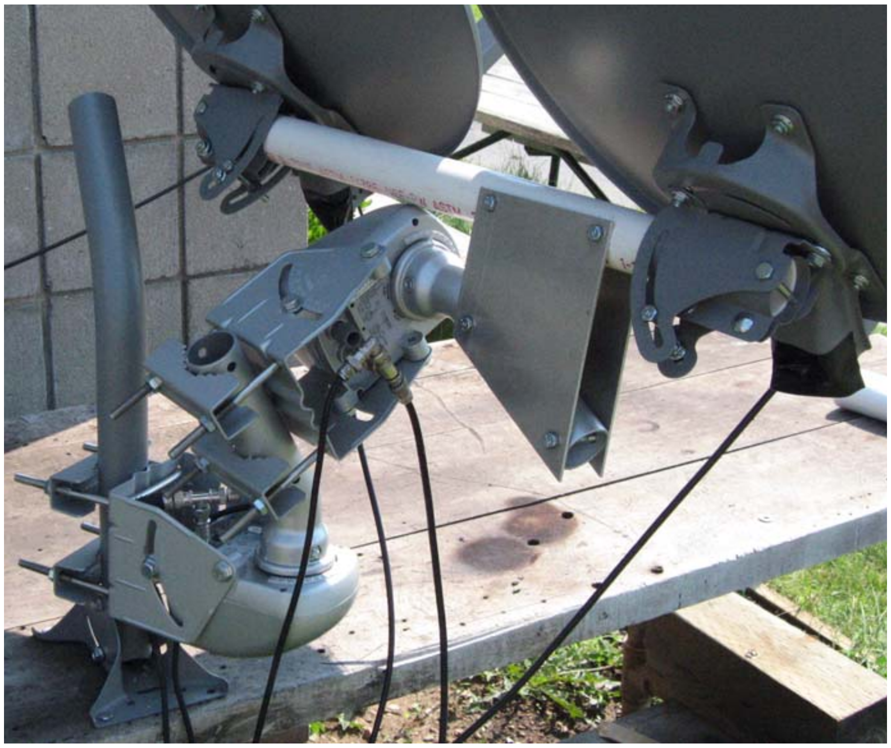


Figure 3.2.2: VSRT rotation system controlling two adjacent dishes

The local oscillator is a device used to generate a signal which is beat against the signal of interest to mix it to a different frequency. The LNB is fixed on the satellite dish. Satellites use comparatively high radio frequencies to transmit their signals. When radio signals are transmitted through coaxial cable, the higher the frequency, the more losses occur in the cable per unit length, which is the main reason that waveguides are needed. The job of the LNB is to use the superheterodyne principle to take a wide block of relatively high frequencies, amplify and convert them to similar signals carried at a much lower frequency (IF). These lower frequencies travel through cables with much less attenuation of the signal, so there’s much more signal left on the receiver end of the cable.

3.3 Detailed Block Diagram

Below is a more detailed diagram of the hardware involved in the 3-baseline interferometer system:

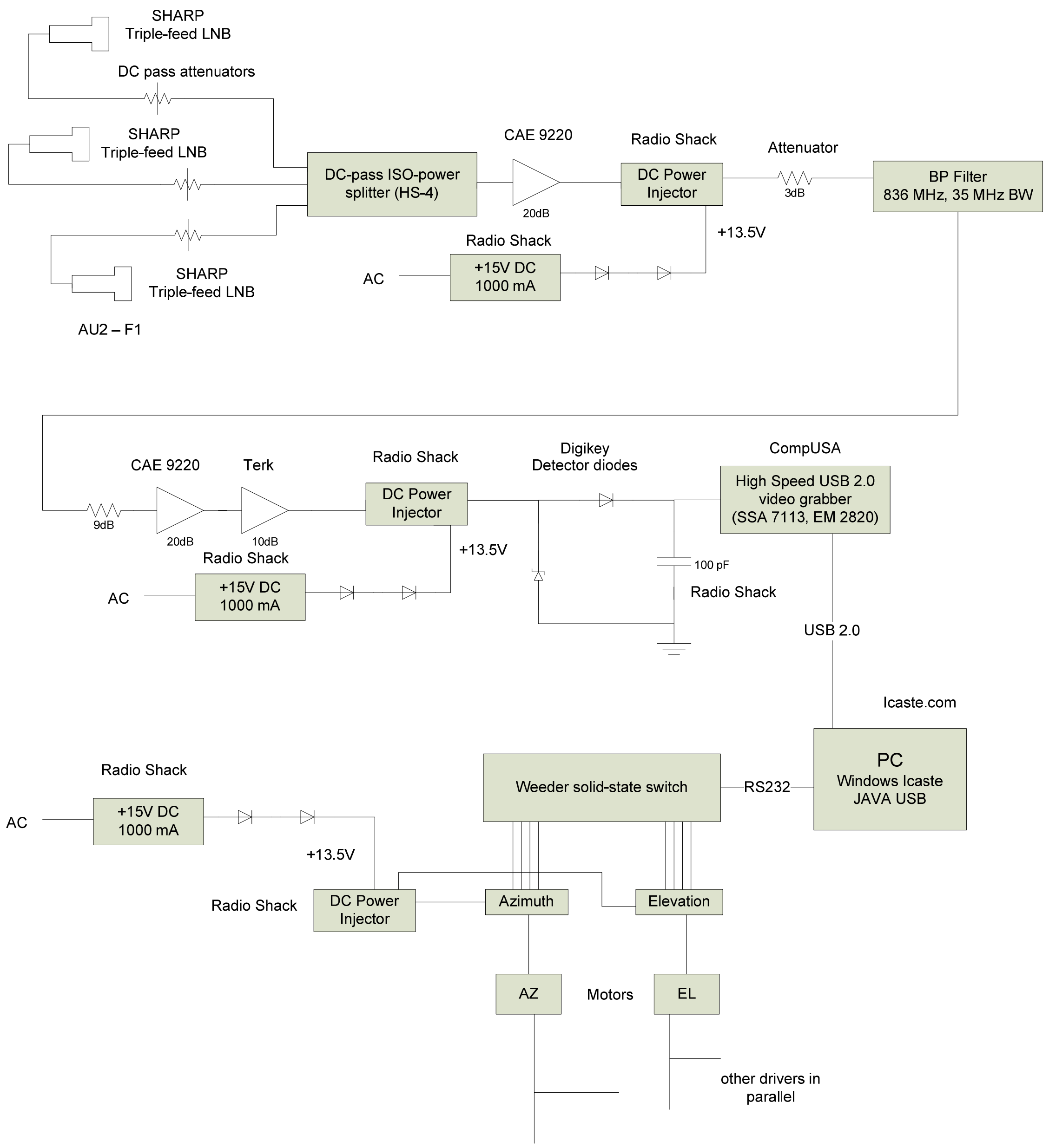


Figure 3.3.1: Detailed Hardware Schematic

The hardware include 18''x20'' DIRECTV Satellite Dish Antennas with 3 dual LNB's (AU2-F1), a DC-pass 4 way power splitter (HS-4), an Inline Amplifier (CAE 9220), several power injectors (15-1170) and 15 V AC-to-DC power adapter (273-1691), 3-Amp barrel diodes 1N5404 (276-1114), some male-to-male "F" connector adapters (278-219) and "F" connector to BNC jack adapters (278-277), coaxial adapters (BNC female/RCA male) (278-250), detector diode 1N6263 (497-2508-1-ND), M61235 6’ RG-6 cables with F connectors (M61235), 6-ft RG-59 Coaxial cable (BNC) (278-990), resistors, a 100 pF capacitor, a CompUSA Video Grabber USB 2.0 (SKU 318714) and a JcommUSB API (Personal edition). The main sources were Amazon.com, RadioShack, Icaste.com and CompUSA. As can be seen in figure 3.3.1, the detector section consists of two diodes and a capacitor, and was soldered together as shown below:

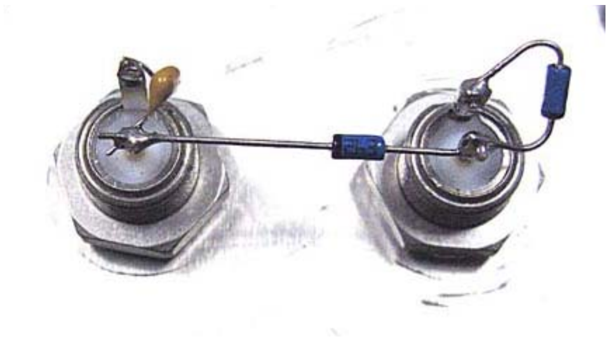


Figure 3.3.2: Detector Hardware

Also, as shown in figure 3.3.1, we need to drop the +15 V voltage down to +13.5 V using two diodes in series for the DC power injectors. Here’s how this was done:

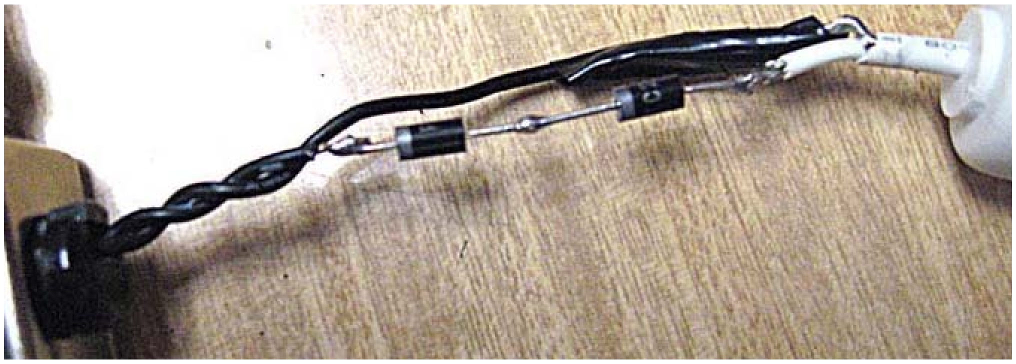


Figure 3.3.3: Diode drop for DC Power Injector

3.4 Motor control

Motor control for the Sun tracking system can be handled using a “horizon to horizon” satellite dish drive. These drives are available from an Italian company called stab. The HH90 unit costs \$80. An industry-standard communication protocol known as DiSeqc has been developed for these drives. An inexpensive solution consists of using a RS232 solid state switch module to “push” the buttons on the handheld Stab MP01 DiSeqc 1.2 control (\$39). The MP01 generates three DiSeqc 1.2 commands: move 1 step East, move 1 step West, go to zero position. Below is a detailed schematic of the motor control used in the VSRT setup.

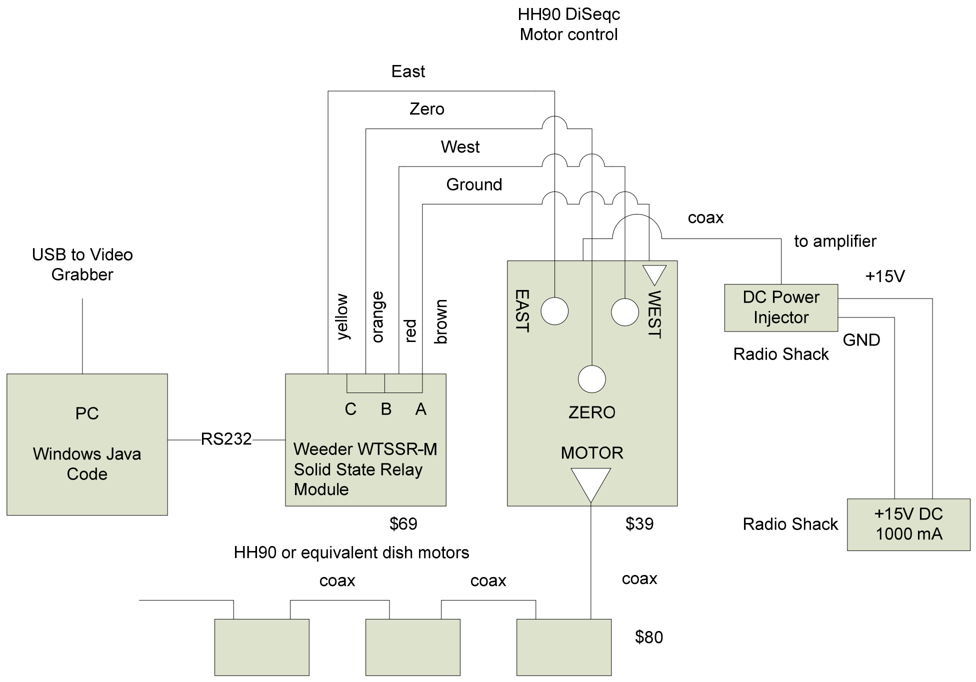


Figure 3.4.1: Motor control Block Diagram

Below is a picture of the actual Stab MP01 DiSeqc used:

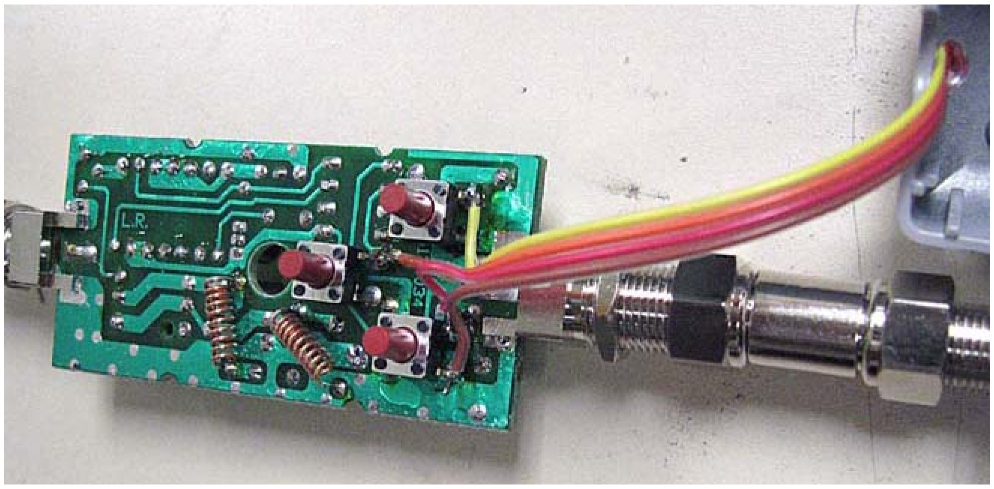


Figure 3.4.2: Stab MP01 DiSeqc

4 Concepts Used for Modeling the Sun

4.1 Visibility

By visibility, we represent the amplitude of the observed lobe pattern; the fringe amplitude (fringe visibility). The complex visibility function is equal to the Fourier transform of the source brightness distribution. The visibility of the solar disk can be expressed as a 2D integral:

$$V(z) = \int_0^R \int_0^{2\pi} B(r,\vartheta) e^{jrz(\vartheta)} r\,dr\,d\vartheta,$$

where $B(r,\vartheta)$ is the brightness as a function of the polar coordinates.

The visibility is normalized by dividing by $\int_0^R \int_0^{2\pi} B(r,\vartheta) r\,dr\,d\vartheta$. The calculation of the 2D integral can be done in rectangular coordinates as well. In this case, the visibility can be rewritten as the 2D Fourier transform of the brightness. It is convenient to approximate this calculation using the superposition of several 1D integrals, instead of using a heavily oversampled DFT

For a uniform disk, we compute the normalized visibility as follows:

$$V_{uniform}(z) = \frac{\int_0^R \sqrt{R^2 - r^2}\cos(rz)\,dr}{\frac{\pi R^2}{4}} = \frac{2J_1(Rz)}{Rz}$$

where R represents the angular radius of the sun (radians); z represents the interferometric phase in radians per radian, in fact, $z = 2\pi \frac{projbase}{\lambda}$, where $projbase = \sqrt{b\_east[base]^2 + b\_north[base]^2}$.

For reference, the simplification to a first order Bessel function was developed as follows:

$$V_{uniform}(z) = \int_0^R \int_0^{2\pi} \frac{e^{jrz\sin(\vartheta)}}{\pi R^2} r\,dr\,d\vartheta = \int_0^R \frac{2rJ_0(rz)}{R^2}\,dr = \frac{2J_1(Rz)}{Rz},$$

given that $J_0(rz) = \frac{1}{2\pi}\int_0^{2\pi} e^{jrz\sin(\vartheta)}\,d\vartheta$.

For a uniform disk plus an outer ring, we have:

$V(z) = \dfrac{\dfrac{2J_1(Rz)}{Rz} + FJ_0(Rz)}{1+F}$, where F represents the fraction of the Sun's radio output in the enhanced brightness of the limb

For a uniform disk plus many weighted rings:

$$V(z) = \frac{\dfrac{2J_1(Rz)}{Rz} + A}{1+F}, \text{ where } F = \sum_{r1}^{r2} \frac{w(r)2\pi r}{\pi R^2}\Delta r \text{ and } A = \sum_{r1}^{r2} \frac{w(r)2\pi r J_0(Rz)}{\pi R^2}\Delta r$$

When $r_1 \le r < R$, $w(r) = a_1 e^{b_1 r}$,
and when $R \le r \le r_2$, $w(r) = a_2 e^{-b_2 r}$,
where a_1, a_2, b_1, b_2, are the coefficients derived from the solution of the following two nonlinear systems of equations:

$$\begin{cases} 1.00 = a_1 e^{b_1(R - offset\_1)} \\ 1 + F = a_1 e^{b_1 R} \end{cases}$$

$$\begin{cases} 1 + F = a_2 e^{-b_2 R} \\ 0.0001 = a_2 e^{-b_2(R + offset\_2)} \end{cases}$$

In this case, F is the fraction of limb brightening. The parameters offset_1 and offset_2 give more degrees of freedom to the limb brightening profile. They can be described mathematically as follows:

$$offset\_1 = R - r_1$$
$$offset\_2 = r_2 - R$$

In the program, these were set to *offset_1* = 0.01 and *offset_2* = 0.02. The following plot shows these parameters graphically:

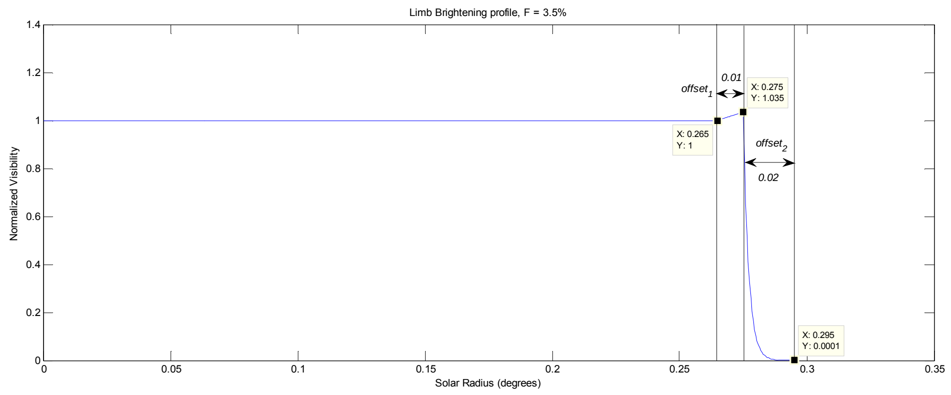


Figure 4.1.1: Limb brightening with many outer rings

We found that the model fits the data much better when we used more rings for limb brightening. So far, the visibility is $V(z) = V_{uniform}(z) + V_{\lim b}(z)$.

Note that the weighting function w(r) varies across the disk (as the radius changes) and controls the exponential increase or decay of the limb brightening profile. The main effect of this more sophisticated limb brightening technique is the widening of the closure phase.

Another factor that was added was the linear gradient applied across the disk. Even with the relatively large 4° beam of the VSRT, a brightness gradient of up to about $\pm 20\%$ from limb to limb can be introduced by miss-pointing of the dish so that the Sun lies on the edge of the beam. (This was done to simulate pointing errors.) The linear gradient was approximated numerically, since there's no nice function such as Bessel functions that fits that curve.

The normalized gradient was to be added on top of the normalized visibility, obeying superposition:

$$V_g(z) = \frac{\int_0^R \sqrt{R^2 - r^2}\,\sin(rz)\frac{\beta r}{R}\,dr}{\frac{\pi R^2}{4}}$$, where rz is in radians, and β is the gradient fraction

So far, the visibility is given by $V_{sun}(z) = V_{uniform}(z) + V_{\lim b}(z) + V_g(z)$. To add a sunspot, we simply add the sunspot's contribution by taking the magnitude and normalizing:

$V_{spot} = spotamp * a^2 \cos(\vartheta)$, where a^2 is the fraction of flux in the spot ($a = \frac{spotsize}{R}$)

$V = V_{sun}(z) + V_{spot}$, where $\vartheta = \frac{2\pi(b\_east[base]x\frac{\pi}{180} + b\_north[base]y\frac{\pi}{180})}{\lambda}$, where λ is the wavelength ($\lambda = \frac{c}{f}$) and x and y are defined as the following:

> $x = spotrad$*cos($spotang$), where $spotang$ is in radians
> $y = spotrad$*sin($spotang$)

where *spotang* = angle around the sun (radians)
spotamp = sunspot amplitude (intensity)
spotrad = distance away from the center of the sun (in units of R: set to 0.2 R)
spotsize = size of the sunspot (in units of R: set to 0.1 R)

The parameters *b_east*[*base*] and *b_north*[*base*] are the *u* and *v* projected baseline components. The projected baseline is computed by squaring, adding and taking the square root of the *u* and *v* components, as shown above. The final normalized visibility with the contribution of the sunspot is computed as follows:

$$V_{final}(z) = \frac{\sqrt{V^2 + sumi^2}}{nsum},$$
where $nsum = 1 + spotamp * a^2$ and $sumi = spotamp * a^2 \sin(\vartheta)$

4.2 Closure Phase

Suppose we have a three element interferometer set up like the following:

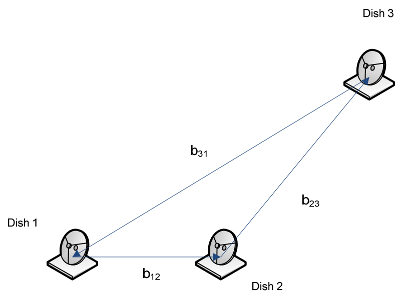


Figure 4.2.1: Simple 3-baseline Interferometer

Let the complex visibility of the fringe associated with baseline b_{jk} be denoted by V_{jk}. Then, we have:

$$V_{jk} = |V_{jk}| e^{j\phi_{jk}}$$

Denoting the measured phase of each fringe as φ_{jk}, we can describe it mathematically:

$\phi_{jk} = \phi_{jk}^{source} + \varepsilon_j - \varepsilon_k + \delta_{jk}$, where $\varepsilon_j, \varepsilon_k$ are atmosphere turbulence-induced phase errors at the jth, kth apertures, and δ_{jk} indicates measurement noise. Now, the three measured phases in a 3-element interferometer are summed to compute the closure phase φ_c:

$$\begin{aligned}\phi_c &= (\phi_{12}^{source} + \varepsilon_1 - \varepsilon_2 + \delta_{12}) + (\phi_{23}^{source} + \varepsilon_2 - \varepsilon_3 + \delta_{23}) + (\phi_{31}^{source} + \varepsilon_3 - \varepsilon_1 + \delta_{31}) \\ &= \phi_{12}^{source} + \phi_{23}^{source} + \phi_{31}^{source} + (\delta_{12} + \delta_{23} + \delta_{31})\end{aligned}$$

The closure phase is insensitive to phase errors induced by the atmosphere, but the measured closure phase is corrupted by measurement noise. The visibilities can be superimposed as we add more things on top. The closure phase can only take on values of 0 or 180° for any source that has reflection symmetry about a line through the centroid (uniform sun). However, when we add limb brightening or sunspots, the closure phase can smoothen out (at the sharp transition points) or spread out/in its transition points (from 0 to ± 180°, and vice-versa).

Least squares analysis was applied to the closure phase curves. A sum of least squares can be calculated by squaring the residuals and taking their sum. The residual is simply the difference between the observed and the predicted (model) value.

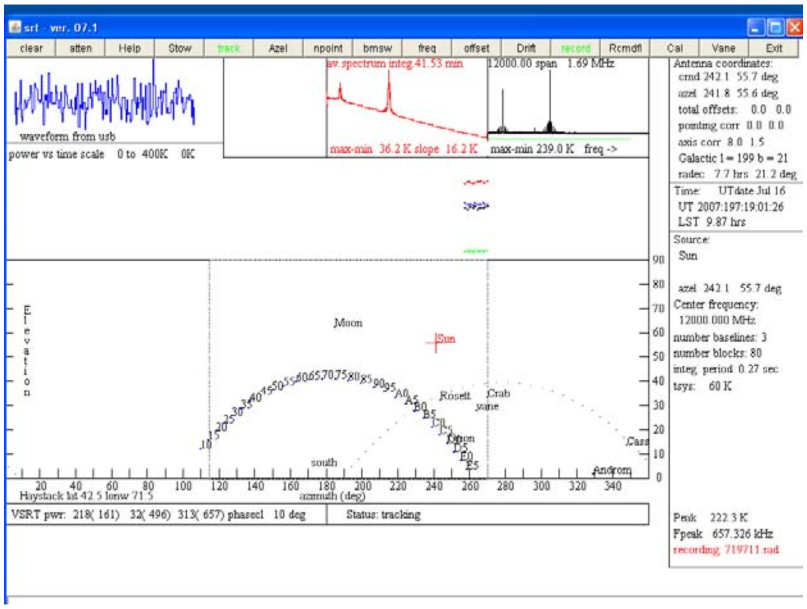
$$sum = \sum_{i=1}^{n} (y_i^{data} - y_i^{predicted})^2$$

where y_i^{data} is the actual data collected for the closure phase,

and $y_i^{predicted}$ is the predicted data computed from the model.

5 Software Development

Data was taken using a real-time Java console data acquisition program, which allowed us to record the visibility amplitudes and the closure phase in RAD files. These files were later read by Matlab or Python. Below is a screenshot taken while collecting data with the Java GUI:



power vs time scale 0 to 400K 0K

av. spectrum integ 41.53 min

max-min 36.2 K slope 16.2 K

12000.00 span 1.69 MHz

max-min 239.0 K freq ->

Haystack lat 42.5 lonw 71.5

VSRT pwr: 218(161) 32(496) 313(657) phasecl 10 deg

Status: tracking

Antenna coordinates:
cmd 242.1 55.7 deg
azel 241.8 55.6 deg
total offsets: 0.0 0.0
pointing corr 0.0 0.0
axis corr 8.0 1.5
Galactic l = 199 b = 21
radec 7.7 hrs 21.2 deg

Time: UTdate Jul 16
UT 2007:197:19:01:26
LST 9.87 hrs

Source:
Sun

azel 242.1 55.7 deg
Center frequency:
12000.000 MHz
number baselines: 3
number blocks: 80
integ. period 0.27 sec
tsys: 60 K

Peak 222.3 K
Fpeak 657.326 kHz
recording 719711.rad

Figure 5.1: Java GUI real-time data acquisition system

What we can clearly see in the figure above is that the VSRT's are tracking the Sun, and that a text file is being recorded. We can see the UT time, the position of the Sun relative to the Earth, and at the top right, the spectrum of three fringes; starting from the right, we see the intermediate, then the long, and finally the short baseline.

Programs were developed in MATLAB to read the data collected from the RAD files and compare them with the model developed. These programs are parameterized in a way so that when the user runs them, he specifies whether a sunspot should be applied (along with its amplitude), whether or not limb brightening should be added (and its amount), whether or not to use many rings or one outer ring for limb brightening, and whether or not a linear brightness gradient should be added (and its amount). The motivation for this is that using MATLAB, the user can quickly tweak the parameters and rerun the simulation to see the effect on the model's curves. More advanced programs that handle the least-square analysis task were also developed. Further information about these programs can be found in the "VSRT Software Documentation" document.

6 Results & Discussion

On July 8th, an intense sunspot appeared on the sun, and it is of interest to see how well we can detect that change on the sun's surface using the 3 baseline VSRT interferometer system. To do this, we utilize the concept of a closure phase. With three LNB's, the phase of the fringes can be added up by going around a triangle of baselines. We then get a "closure phase" when the local oscillator phases cancel. An interesting fact about the closure phase is that is independent of the station atmosphere and local oscillator phases. Since it's largely free from instrumental errors (slightly corrupted by measurement noise), it can be used to model source structure and remove the ambiguity in structure modeled with the visibility amplitudes alone.

Using this closure phase concept, for a completely uniform Sun, with no sunspot activity or limb brightening, we would expect a jump from 0 to ± 180° at around 16.25 hrs (UT) and another jump from -180 to 0° at about 17.5 hrs (UT), just like shown below in the simulation for day 192 (blue line):

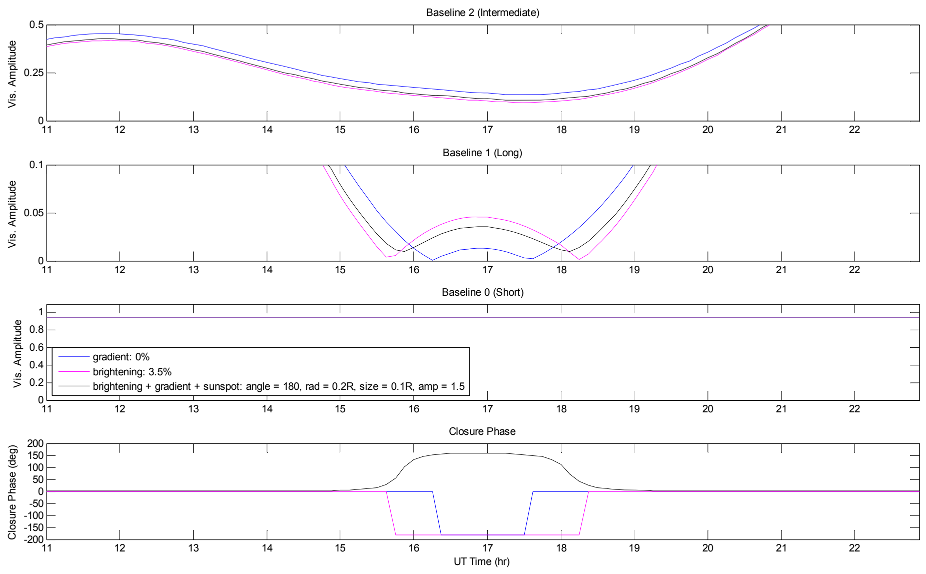


Figure 6.1: Simulation showing normalized visibility plots and closure phase

The data collected around day 190 (July 9th) suggest that there exist deviations from this ideal case. Indeed, when we run our simulation model with a certain sunspot of some intensity, we see that the model's curves tend to fit the actual data better. The parameters that enter the model include a surface gradient, limb brightening, the angular radius of the Sun, and sunspot attributes. The linear surface gradient is applied to account for pointing errors of the hardware. Limb brightening is applied to simulate the limb brightening of the sun's disk. Sunspot attributes include its position on the sun (set by its angle and distance away from the center of the disk), its relative flux and its intensity.

Although the observing frequency of the Nobeyama RadioHeliograph is at 17 GHz, we can use their daily images to evince the existence of solar activity for days 189, 190, 191, 192 respectively:

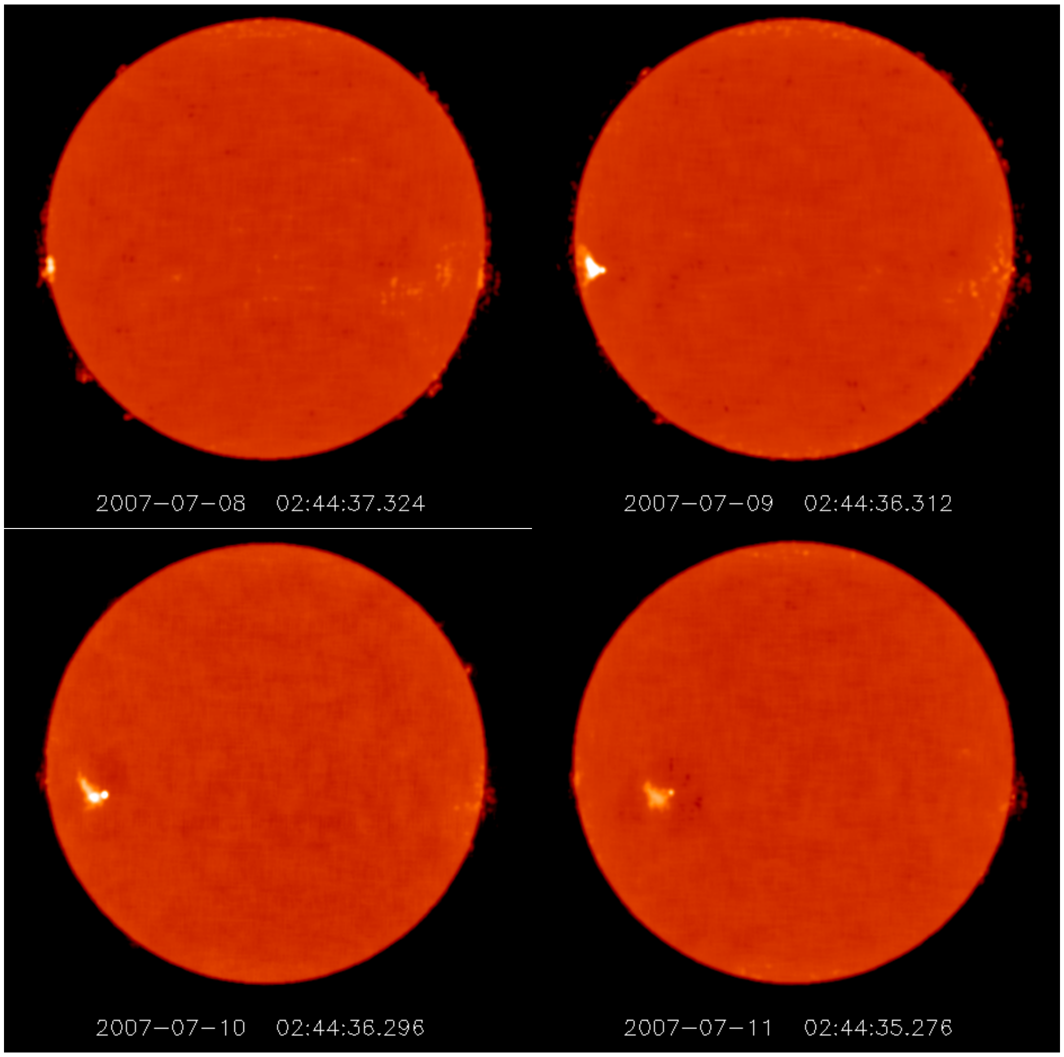


Figure 6.2: Nobeyama RadioHeliograph images at 3 UT

For these four days, according to the Nobeyama RadioHeliograph, the solar radius is displayed in the table below:

Table 6.1: Solar radius across a period of 4 days at a frequency of 17 GHz

Day of Year	Solar radius (arc seconds)	Solar radius (degrees)
189	955.676	0.2654656
190	955.690	0.2654694
191	955.709	0.2654747
192	955.732	0.2654811

Since our observing frequency is less than 17 GHz, we expect the sun to appear larger, thus, a larger angular radius. To run the models for these days, we need to determine a value for the solar radius. Selhorst has published some papers on radius variations over a solar cycle, and he found that the overall mean radius varied from 976.6 ± 1.5 arcsec to 974.8 ± 0.6. (0.2706 to 0.2717 degrees), using daily maps covering about one solar cycle (1992-2003). He states that solar radius measurements at radio frequencies are not a simple task, because the Sun lacks a clear quiet atmosphere. This atmosphere is filled with ever-changing small structures, such as sunspots, prominences, spicules, faculae, that are prominent in the observed radio Sun. These features influence the choice of where the solar radius is measured. If the atmosphere had a smooth profile, the brightest point at the solar limb would be close to unity at a certain frequency ($\tau = 1$). Selhorst et al. (2003), however, showed that the limb brightening intensity at 17 GHz is not uniformly distributed around the Sun, being larger near the polar region, indicating the influence of features such as spicules in the maximum brightening. This was accounted for in the model by adjusting the limb brightening profile, using exponential rising and decaying envelopes. After a lot of simulations and fitting throughout these days, we found that the data is best fit when R = 0.275°. In fact, we noticed that as the solar radius increased, the closure phase tends to become narrower; the transitions from around zero to ± 180°, and vice-versa. Thus, most models use this value for the solar radius.

Since our model contains quite a few parameters, which leads to extensive amounts of tweaking these parameters, a program was developed to construct a surface, plotting the sum-of-squares across a set of two parameters. Since our model seems to be quite sensitive to the angular radius of the Sun, I decided to vary the solar radius and the sunspot's intensity and compute the sum-of-squares based on these two variables. The resulting surface would show exactly how sensitive the model really is to the solar radius. Another reason why this approach was taken to solve the least squares problem is because the closure phase is nonlinear, so the best model has to be found by iterating through the parameter space to find the best fit; the fit that minimizes the sum of the squares of the residuals. The sum of squares depends only on the closure phase data collected, and not on any normalized visibilities of the baselines. The reason for that is, because the visibilities are scaled, and thus a scaling offset can throw off the best fit by a large amount, and fitting the baselines closely doesn't guarantee that the closure phase will be fit just as well; in fact, it may be way off. That is why the closure phase is the cardinal factor of this analysis. The goal is to be able to detect a sunspot's intensity to a discernable degree.

To demonstrate the usefulness of this program, here is the resulting sum-of-squares surface for day 189:

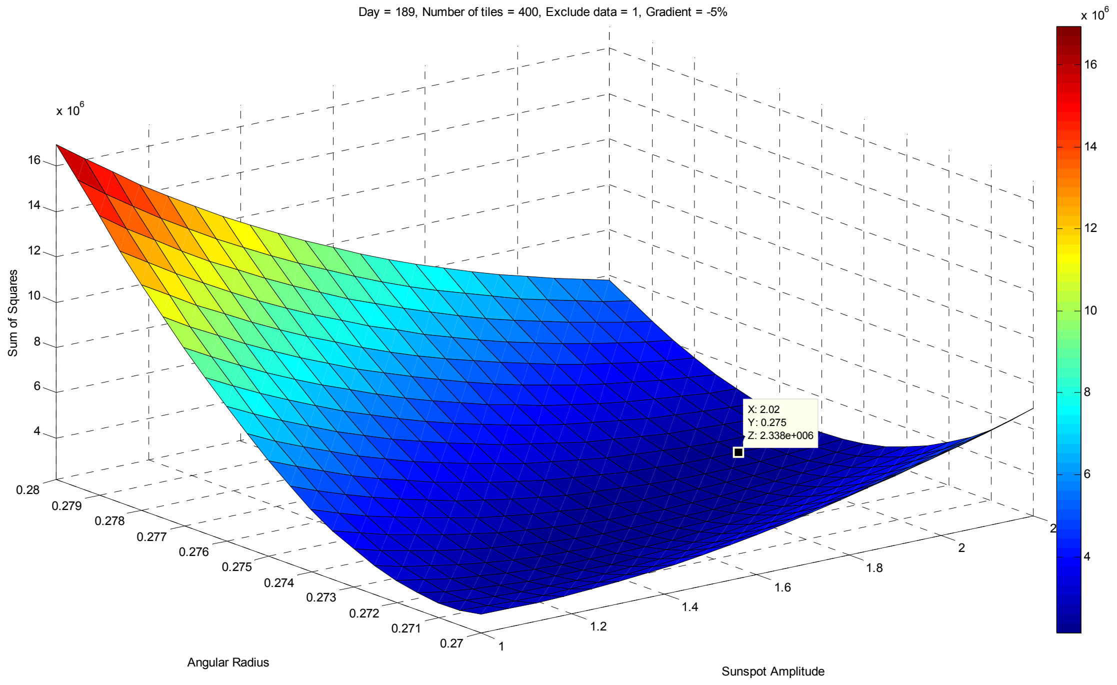


Figure 6.3: 3D surface plot for day 189 showing the minimum sum-of-squares point

Pinpointing at the minimum, we can simply read off the parameters that optimize the fitting. A model is computed at every vertex, and then, the sum of squares is computed. In a sense, the least-squares solution is found. Picking the optimum parameters in the same way, we obtain the following results for the closure phase for days 190, 191, and 192:

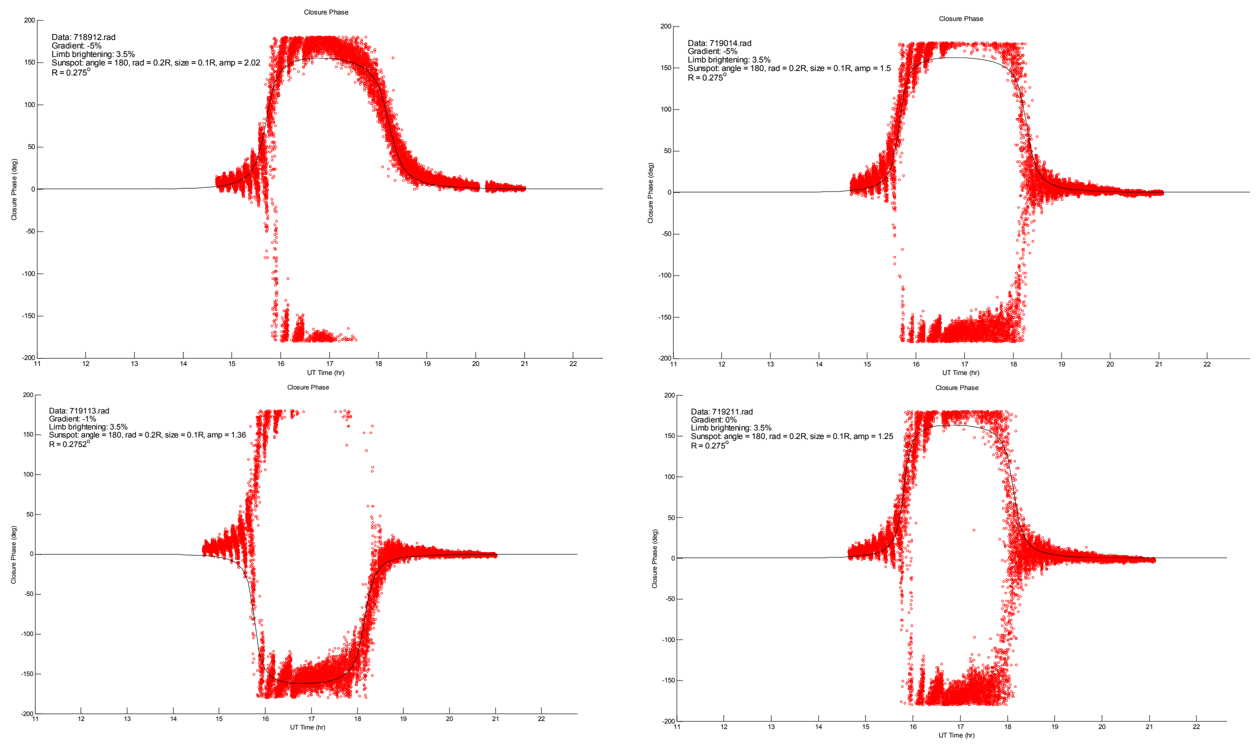


Figure 6.4: Actual closure phase data and model fit with optimum parameters

As shown, the models contain the parameters that fit the closure phase the best. Slight tweaking took place to account for outliers. To run the above simulations, we used the following parameters for each day:

Table 6.2: Parameters derived from the Sum-of-Squares surfaces

Day of year	**Solar radius (deg.)**	**Sunspot intensity**	**% Gradient**	**% Limb brightening**
189	0.275	2.02	5	3.5
190	0.275	1.50	5	3.5
191	0.2752	1.36	1	3.5
192	0.2749	1.54	0	3.5

Table 6.3: Final parameters used after tweaking

Day of year	**Solar radius (deg.)**	**Sunspot intensity**	**% Gradient**	**% Limb brightening**
189	0.275	2.02	5	3.5
190	0.275	1.50	5	3.5
191	0.2752	1.36	1	3.5
192	0.275	1.25	0	3.5

It's interesting to know what happens after day 192 to the sunspot's intensity. Although certain difficulties with taking data hindered our inference capabilities, the data taken on day 194 showed the apparent trend to continue. Running the model on day 194 and adjusting the parameters empirically, we observe the following closure phase fit:

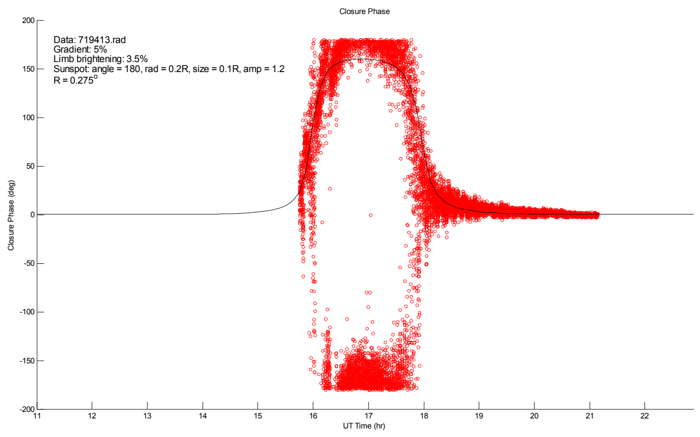


Figure 6.5: Closure phase fit for day 194

Thus, for day 194, the sunspot intensity was around 1.15; it's definitely less than 1.25. This is a good sign since on the NoRH image on July 13th (day 194), we see that the sunspot still exists and it's less intense than that of July 11th (day 192). Below are the NoRH images for days 193 and 194, respectively:

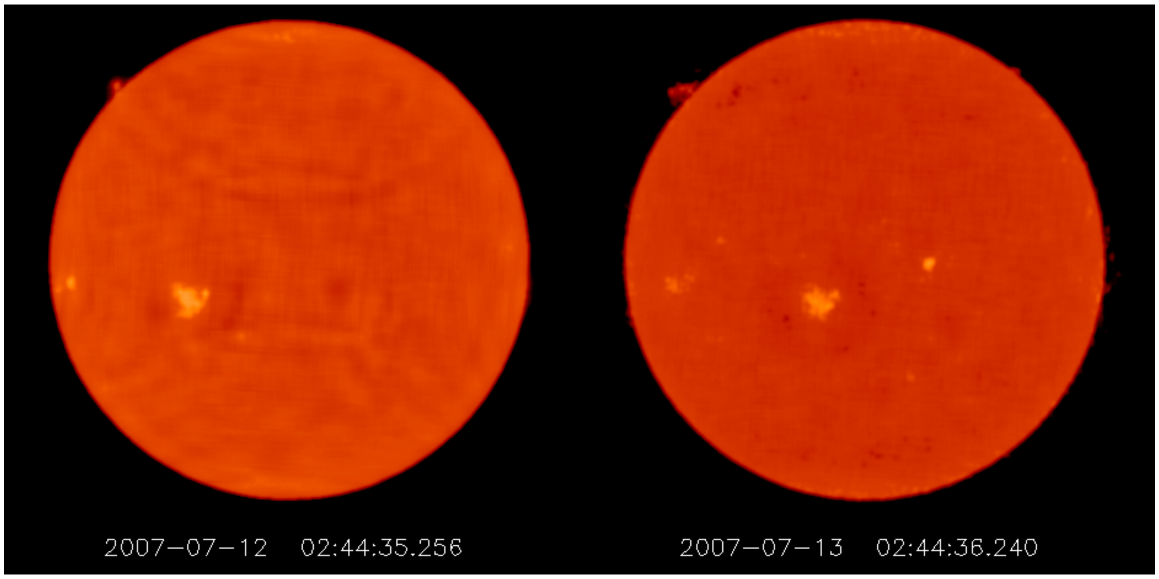


Figure 6.6: NoRH images of days 193 and 194

Now, we plot the sunspot trend observed:

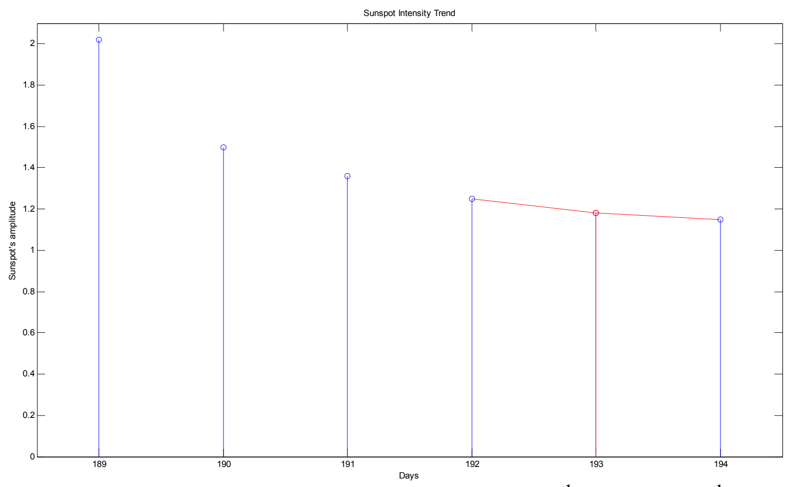


Figure 6.7: Sunspot trend from July 8th to July 13th

Data on day 193 was not collected and the point shown above in red is simply a guess of the sunspot's intensity. Below are plots that display the sum-of-squares vs. sunspot intensity:

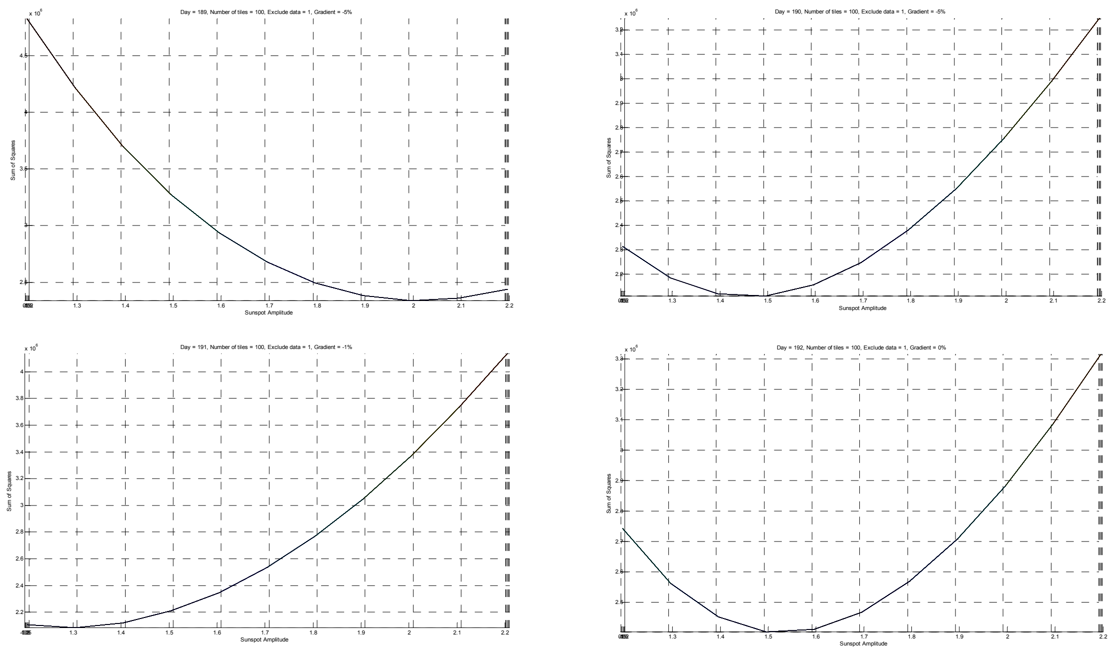


Figure 6.8: Series of plots of sum-of-squares vs. sunspot amplitude for four consecutive days at R = 0.275°

One interesting thing to notice from the 2D plots above is that the minimum is shifting to the left, moving toward lower intensity. However, the bottom right plot in figure 6.8 is a little off, displaying its minimum sum-of-squares at amplitude of 1.5, instead of 1.25, because in a non-weighted least squares sense, an amplitude of 1.5 will give you the best fit, but looking at the result graphically, we see that the upper data is not fit that well, so it's a more natural decision to fit the data with an intensity of 1.25 because on average, it fits the decay of the closure phase data slightly better, which is chiefly determined by the sunspot's amplitude; the smaller the amplitude, the sharper the decay. The results signify that throughout the period of these four days, the sunspot's intensity was decreasing. The minimum in the above plots represent the place where the sum of squares is minimized, and thus, get a better fit.

7 Conclusion

In summary, we documented the main hardware involved in the design of the VSRT's and showed an approach to analyzing data recorded from radio observations on the Sun. Moreover, we utilized the concept of least squares in order to infer information about the sunspot intensity and the solar radius by taking advantage of the closure phase. The main goal for this analysis tool is to be able to automatically determine the set of parameters {solar radius, sunspot intensity} that will yield the best fit. We were successfully able to detect the sunspot activity with an adequate degree of accuracy using the 3-element VSRT system.

Primary focus on future work on this project should include covering sensitive parts of the VSRT hardware, since it's heavily affected by rain and coming up with a more stable setup (maybe mounting the VSRT's to the wall) to minimize pointing errors. Finally, more complicated interferometer experiments should be carried out; possibly by adding another dish. Then, we would have four LNB's, six baselines, which indicate that there are three independent closure phase triangles.

References

[1] Rogers, Alan, E. E., "VSRT Memo Series", <http://www.haystack.mit.edu/edu/undergrad/VSRT/VSRT_Memos/memoindex.html>, July 13, 2007.
[2] Haystack Observatory, "Radio Astronomy Tutorial", <http://www.haystack.mit.edu/edu/undergrad/materials/RA_tutorial.html>, July 13, 2007.
[3] Baylin, Maddox, and McCormac, World Satellite TV and Scrambling Methods: The Technicians' Handbook, 2nd ed. Baylin Publications, 1991.
[4] Smerd, S. F., "Radio-Frequency Radiation from the Quiet Sun.", Division of Radiophysics, C.S.I.R.O., University Grounds, Sydney. (August 30, 1949).
[5] Wikipedia, "Black body", < http://en.wikipedia.org/wiki/Black_body>, July 11, 2007.
[6] Cillié, G. G., and Menzel, D. H., Harv. Coll. Obs. Circ. No. 410 (1935)
[7] Smerd, S. F., and Westfold, K. C., Phil. Mag. 40: 831 (1949)
[8] Kundu, Mukul R., Solar Radio Astronomy, The Radio Astronomy Observatory, The University of Michigan, Ann Arbor, Michigan, Interscience Publishers, 1965.

Appendix

Here is the ***simLimb.m*** Matlab code developed:

```
function simLimb(apply_gradient, apply_brightening, plot_spot, answer,
beta_input, F_input, input_string, spotamp_input, useManyRings, R_input)
% Function reads an input file and runs a simulation.
% Gradient: used to model pointing error effects on the closure phase, since
the closure phase is
% affected by pointing errors.
% Limb Brightening: more sophisticated limb brighteness profile (exponential)
%
% SYNTAX: simLimb(apply_gradient, apply_brightening, plot_spot,
plot_actual_data, beta_input,
%                  F_input, input_string, spotamp_input, useManyRings,
R_input)
%
% Example: simLimb(1, 1, 1, 1, 10, 3.5, '718912.rad', 1.6, 1, 0.275);
%          simulate a gradient, a sunspot and limb brightening,
%          read a data file, apply a 10% gradient and 3.5% limb
%          brightening, read data from '718912.rad', set sunspot's
%          amplitude to 1.6, use many rings for limb brightening, R = 0.275
%
% Author: Ted Tsiligaridis
% Last Edited: 2007/08/2, Version: 1.6

numargs = 10;
if nargin > numargs,
    error('Too many input arguments.');
elseif nargin < numargs,
    error('Need more input arguments.');
end

% Define global variables
global time_sim_hr;
global base1_sim base2_sim base3_sim closurePhase_sim;
global base1_sim2 base2_sim2 base3_sim2 closurePhase_sim2;
global base1_sim3 base2_sim3 base3_sim3 closurePhase_sim3;
global time_data_hr;
global base1_data base2_data base3_data closurePhase_data;
global input F beta;
global spotang spotsize spotamp spotrad;
global rmax;
global a1 b1 a2 b2;

% Program execution starts --------------------------------------------------
-----------------

index_sim = 0; % used for appending numbers in arrays
index_data = 0;

% parse input_string to get new_day
new_day_string = [input_string(2) input_string(3) input_string(4)];
new_day = str2double(new_day_string);
if isnumeric(new_day),
    disp(['New day = ' num2str(new_day)]);
```

```
else
    disp('Problem: new_day not read correctly.');
end

% Simulation
freq = 11.25e09 + 836e06; % 12.086 GHz observing frequency
R = num2str(R_input);
R = str2double(R);
rmax = R;
wavl = 299792458.0/freq;

% self-adjusting weighting functions
F = F_input/100.0;
offset_1 = 0.01; offset_2 = 0.02;
eq1 = [num2str(1+F), ' = a*exp(b*', num2str(R), ')'];
[a1, b1] = solve(['1.00 = a*exp(b*', num2str(R - offset_1), ')'], eq1);
a1 = max(double(real(vpa(a1))));
b1 = max(double(real(vpa(b1))));
eq2 = [num2str(1+F), ' = a*exp(-b*', num2str(R), ')'];
[a2, b2] = solve(eq2, ['0.0001 = a*exp(-b*', num2str(R + offset_2), ')']);
a2 = max(double(real(vpa(a2))));
b2 = max(double(real(vpa(b2))));

% define vectors
time_sim = 1:96; time_sim_hr = 1:96;
az_deg_sim = 1:96; el_deg_sim = 1:96;
base1_sim = 1:96; base2_sim = 1:96; base3_sim = 1:96; closurePhase_sim =
1:96;
base1_sim2 = 1:96; base2_sim2 = 1:96; base3_sim2 = 1:96; closurePhase_sim2 =
1:96;
base1_sim3 = 1:96; base2_sim3 = 1:96; base3_sim3 = 1:96; closurePhase_sim3 =
1:96;

% insert spot in the sun
spotang = 180;
spotrad = rmax*0.15;
spotsize = rmax*0.1;
spotamp = spotamp_input;

% initialize temporary arrays
baz = [0, 0, 0]; blen = [0, 0, 0]; belev = [0, 0, 0];
xx = [0, 0, 0]; yy = [0, 0, 0]; zz = [0, 0, 0];
bblen = [0, 0, 0]; delay = [0, 0, 0];
b_r = [0, 0, 0]; b_east = [0, 0, 0]; b_north = [0, 0, 0];
phas = [0, 0, 0]; visibility = [0, 0, 0];
phas2 = [0, 0, 0]; visibility2 = [0, 0, 0];
phas3 = [0, 0, 0]; visibility3 = [0, 0, 0];

disp('Simulation running...');
disp(['Angular radius of Sun = ', num2str(R)]);
numDays = 4;
totalTime = 24*numDays;
for i=0:totalTime-1,
    if mod(i, 24) == 0,
        percent = double(i/(1.0*totalTime)*100.0);
        disp([num2str(percent), '% complete.']);
    end
    lon = toRad(-71.5); % get latitude and longitude of Haystack
    lat = toRad(42.5);
    ttime = double(toSec(2007, new_day, 11, 0, 0)) + i*1800.0/4.0;
    timeStamp = toYrDay(ttime);
    date_tuple = strread(timeStamp, '%s', 'delimiter', ':');
    hr = str2double(date_tuple(2));

    % compute Sun's right ascension and declination
    [ra, dec] = get_sunra_and_dec(ttime);

    % compute azimuth and elevation angles
    ha = double(getGST(ttime) - ra + lon);
    [az, el] = get_radec_az_and_el(ha, dec, lat);

    index_sim = index_sim + 1;
    time_sim(index_sim) = ttime;
    az_deg_sim(index_sim) = az;
    el_deg_sim(index_sim) = el;

    % simulation hardware parameters
    feedoffset = toRad(11.0);
    faz = atan2(sin(feedoffset), cos(feedoffset)*cos(el));
    baz(0 + 1) = (az + faz + pi/2.0);
    baz(1 + 1) = (az + faz - pi/2.0);
    baz(2 + 1) = toRad(107.0);
    blen(0 + 1) = 0.5*21.25*2.54/100.0; % length between two dishes
    blen(1 + 1) = 0.5*21.25*2.54/100.0;
    blen(2 + 1) = (9*12.0 + 7.0 + 7.0/8.0)*2.54/100.0; % distance from the
center point of the two dishes to the third dish
    belev(0 + 1) = 0;
    belev(1 + 1) = 0; % lie in approximately the same plane
    el_ang = 1 + (2*57.3)/(9*12.0 + 7.0 + 7.0/8.0); % ~2 degrees
    elev_offset = -toRad(1)*sin(el);
    belev(2 + 1) = toRad(-el_ang) + elev_offset;

    for stat=0:2,
        lx = blen(stat + 1)*sin(baz(stat + 1));
        ly = blen(stat + 1)*cos(baz(stat + 1));
        lz = blen(stat + 1)*sin(belev(stat + 1));
        xx(stat + 1) = -lx*sin(lon) - ly*cos(lon)*sin(lat) +
lz*cos(lat)*cos(lon);
        yy(stat + 1) = lx*cos(lon) - ly*sin(lon)*sin(lat) +
lz*sin(lon)*cos(lat);
        zz(stat + 1) = ly*cos(lat) + lz*sin(lat);
    end

    for base=0:2,
        % get sx, sy, sz unit vectors in direction of the sun
        gha = double(getGST(ttime) - ra);
        sx = cos(dec)*cos(gha);
        sy = -cos(dec)*sin(gha);
        sz = sin(dec);
        if base < 2,
            bx = xx(base + 1 + 1) - xx(0 + 1);
            by = yy(base + 1 + 1) - yy(0 + 1);
            bz = zz(base + 1 + 1) - zz(0 + 1);
        else
```

```
            bx = xx(2 + 1) - xx(1 + 1);
            by = yy(2 + 1) - yy(1 + 1);
            bz = zz(2 + 1) - zz(1 + 1);
        end
        if hr == 16,
            bblen(base + 1) = sqrt(bx*bx + by*by + bz*bz);
        end

        % projected baseline components: u, v
        b_r(base + 1) = bx*cos(gha) - by*sin(gha);
        b_east(base + 1) = by*cos(gha) + bx*sin(gha);
        b_north(base + 1) = bz*cos(dec) - b_r(base + 1)*sin(dec);
        delay(base + 1) = bx*sx + by*sy + bz*sz;

        % apply a brightness gradient
        % compute Bessel function numerically: uniform disk + brightness
gradient
        sumr = 0.0; sumi = 0.0; r = 0;
        if apply_gradient,
            beta = -beta_input/100.0;
        else
            beta = 0.0;
        end
        increment = 0.00005;
        z_start = sqrt(b_east(base + 1)*b_east(base + 1) + b_north(base +
1)*b_north(base + 1));
        z = 2.0*pi*z_start/wavl;
        while r < R,
            xa = sqrt(R*R - r*r);
            xb = cos(r*z*pi/180.0);
            sumr = sumr + (xa*xb);
            if apply_gradient,
                xb = sin(r*z*pi/180.0);
            else
                xb = 0;
            end
            xc = beta*r/R;
            sumi = sumi + (xa*xb*xc);
            r = r + increment;
        end
        a = pi*R*R/4.0;
        sumr = sumr*increment/a;
        sumi = sumi*increment/a;

        phas(base + 1) = toDeg(atan2(sumi, sumr));
        visibility(base + 1) = sqrt(sumr*sumr + sumi*sumi);

        % use Matlab built-in Bessel function: uniform disk + limb
brightening
        if ~apply_brightening,
            F = 0.0;
        else
            F = F_input/100.0;
        end

        if useManyRings == 0,
            % disk + one outer ring
            y1 = 2*besselj(1, R*z*pi/180.0)/(R*z*pi/180.0);
            y2 = F*besselj(0, R*z*pi/180.0);
            y = (y1 + y2)/(1+F);
        else
            % disk + many weighted rings
            y = applyLimb(z, offset_1, offset_2); % y = Vuniform + Vlimb
        end

        phas2(base + 1) = toDeg(atan2(imag(y), real(y)));
        visibility2(base + 1) = abs(y);

        % use Matlab built-in Bessel function and approximate gradient
numerically
        % uniform disk + limb brightening + linear gradient
        r = 0; sum_g = 0.0;
        while r < R, % 0 < r < R (in degrees)
            xa = sqrt(R*R - r*r);
            if apply_gradient,
                xb = sin(r*z*pi/180.0);
            else
                xb = 0;
            end
            xc = beta*r/R; % linear in r
            sum_g = sum_g + (xa*xb*xc);
            r = r + increment; % increment
        end
        a = pi*R*R/4.0; % scale sum
        sum_g = sum_g*increment/a; % divide by pi*R^2/4
        v = y + sum_g; % limb + gradient

        sum2 = 1; sumi = 0;
        % insert sunspot
        if plot_spot,
            x = spotrad*cos(spotang*pi/180.0);
            y = spotrad*sin(spotang*pi/180.0);
            th = 2.0*pi*(b_east(base + 1)*(x*pi/180.0) + b_north(base +
1)*(y*pi/180.0))/wavl;
            a = (spotsize/rmax);
            v = v + spotamp*a*a*cos(th); % V = Vspot + Vsun
            sumi = spotamp*a*a*sin(th);
            sum2 = sum2 + spotamp*a*a;
        end
        phas3(base + 1) = toDeg(atan2(sumi, v));
        visibility3(base + 1) = sqrt(v*v + sumi*sumi)/sum2;
    end
    base1_sim(index_sim) = visibility(0 + 1);
    base2_sim(index_sim) = visibility(1 + 1);
    base3_sim(index_sim) = visibility(2 + 1);
    closurePhase_sim(index_sim) = phas(0 + 1) - phas(1 + 1) + phas(2 + 1);
    base1_sim2(index_sim) = visibility2(0 + 1);
    base2_sim2(index_sim) = visibility2(1 + 1);
    base3_sim2(index_sim) = visibility2(2 + 1);
    closurePhase_sim2(index_sim) = phas2(0 + 1) - phas2(1 + 1) + phas2(2 +
1);
    base1_sim3(index_sim) = visibility3(0 + 1);
    base2_sim3(index_sim) = visibility3(1 + 1);
    base3_sim3(index_sim) = visibility3(2 + 1);
    closurePhase_sim3(index_sim) = phas3(0 + 1) - phas3(1 + 1) + phas3(2 +
1);
end
fprintf(1, ['100%% complete.', '\n']);

% convert time vectors to hours before plotting
start_hr = 11;
for j=0:length(time_sim)-1,
    hour_sim = (time_sim(j + 1) - toSec(2007, new_day, 11, 0, 0))/3600.0;
    time_sim_hr(j + 1) = hour_sim + start_hr;
end


% Actual data
input = input_string;
time_data = [];
time_data_hr = [];
az_deg_data = [];
el_deg_data = [];
daz_deg_data = []; del_deg_data = [];
freq_data = [];
base1_data = []; base2_data = []; base3_data = []; closurePhase_data = [];
time_offset = 0;
a = 50; b = 50; c = 10; % used for scaling actual baseline data

if answer == 1,
    disp(['Reading: ', input]);
    fid = fopen(input, 'r');
    validLines = 0;
    totalLines = 0;
    while feof(fid) == 0,
        tline = fgetl(fid);
        line_tuple = strread(tline, '%s');
        if char(line_tuple(1)) == '*',
            continue;
        end

        if validLines == 1,
            timestamp = char(line_tuple(1));
            timestamp_tuple = strread(timestamp, '%s', 'delimiter', ':');
            new_day = int16(str2double(char(timestamp_tuple(2))));
            time_offset = toSec(int32(str2double(char(timestamp_tuple(1)))),
int32(str2double(char(timestamp_tuple(2)))), 11, 0, 0);
        end

        num = str2double(line_tuple(10));
        if num ~= 999,
            validLines = validLines + 1;
            index_data = index_data + 1;
            stamp = line_tuple(1);
            nexttime = getTime(stamp);

            time_data(index_data) = (double(nexttime) - double(time_offset) +
double(toSec(2007, new_day, 11, 0, 0)));
            hour_data = double((time_data(index_data) - double(toSec(2007,
new_day, 11, 0, 0)))/3600.0);
            time_data_hr(index_data) = hour_data + start_hr;
            if validLines == 1,
                time_data_hr(1) = min(time_data_hr); % avoid off-by-1 error
in the very beginning
            end
            az_deg_data(index_data) = str2double(char(line_tuple(2)));
            el_deg_data(index_data) = str2double(char(line_tuple(3)));
            daz_deg_data(index_data) = str2double(char(line_tuple(4)));
            del_deg_data(index_data) = str2double(char(line_tuple(5)));
            freq_data(index_data) = str2double(char(line_tuple(6)));

            % scale actual data
            base1_data(index_data) = str2double(char(line_tuple(7)))*6e-
2/sqrt(a*b); % intermediate
            base2_data(index_data) = str2double(char(line_tuple(8)))*6e-
2/sqrt(a*c); % long
            base3_data(index_data) = str2double(char(line_tuple(9)))*6e-
2/sqrt(b*c); % short
            closurePhase_data(index_data) = str2double(char(line_tuple(10)));
% closure phase
        end
        totalLines = totalLines + 1;
    end
    fclose(fid);
    fprintf(1, ['Text file successfully read. ', 'valid lines = ',
num2str(validLines), '\n']);
    fprintf(1, ['                             ', 'total lines = ',
num2str(totalLines), '\n']);
end

plotGraphs(plot_spot, answer);


% Inner functions -----------------------------------------------------------
------------------

% advanced limb brightening
function y = applyLimb(z, offset_1, offset_2)
    global rmax;
    global a1 b1 a2 b2;

    R = rmax;
    dr = 0.0005;
    r0 = R - offset_1;
    r1 = R + offset_2;
    sumF = 0.0;
    for r=r0:dr:r1,
        if r < R,
            weight = a1*exp(b1*r);
        else
            weight = a2*exp(-b2*r);
        end
        sumF = sumF + weight*2*pi*r;
    end
    sumF = sumF*dr/(pi*R*R);
    sumA = 0.0;
    for r=r0:dr:r1,
        if r < R,
            weight = a1*exp(b1*r);
        else
            weight = a2*exp(-b2*r);
        end
        sumA = sumA + weight*2*pi*r*besselj(0, toRad(R*z));
    end
    sumA = sumA*dr/(pi*R*R);
    y = (2*besselj(1, toRad(R*z))/(toRad(R*z)) + sumA)/(1 + sumF);
    return;

% input a string, output a float
function time = getTime(timeStamp)
    % do a delimiter read on the time stamp
    date_tuple = strread(char(timeStamp), '%s', 'delimiter', ':');
    year = int32(str2double(char(date_tuple(1)))); % need to convert strings
to doubles
    day = int32(str2double(char(date_tuple(2))));
    hour = int32(str2double(char(date_tuple(3))));
    min = int32(str2double(char(date_tuple(4))));
    sec = int32(str2double(char(date_tuple(5))));
    time = toSec(year, day, hour, min, sec);
    return; % time in seconds

% get time in seconds since 1970
function secs = toSec(yr, day, hr, min, sec)
    secs = (yr - 1970)*31536000.0 + (day - 1)*86400.0 + hr*3600.0 + min*60.0
+ sec;
    for i=1970:yr-1, % i++, 1970 <= k < yr
        if ((mod(i,4) == 0 && mod(i,100) ~= 0) || mod(i,400) == 0),
            secs = secs + 86400.0;
        end
    end
    if secs < 0.0, % year must be before 1970
        secs = 0.0;
    end
    return;

% plot baselines as a subplot
function plotGraphs(plot_spot, answer)
    global time_sim_hr;
    global base1_sim base2_sim base3_sim closurePhase_sim;
    global base1_sim2 base2_sim2 base3_sim2 closurePhase_sim2;
    global base1_sim3 base2_sim3 base3_sim3 closurePhase_sim3;
    global time_data_hr;
    global base1_data base2_data base3_data closurePhase_data;
    global input; global F beta;
    global spotang spotsize spotamp spotrad;
    global rmax;

    figure;

    % account for change in text file series of baselines: base3_data <->
base1_data
    subplot(4,1,1);
    if answer == 1,
        scatter(time_data_hr, base1_data, 5, 'r'); % intermediate baseline
        hold on;
    end
    plot(time_sim_hr, base3_sim, 'b', time_sim_hr, base3_sim2, 'm',
time_sim_hr, base3_sim3, 'k');
    axis([min(time_sim_hr), max(time_sim_hr), 0, 0.5]);
    set(gca, 'YTick', 0:0.25:0.51);
    set(gca, 'XTick', min(time_sim_hr):1:max(time_sim_hr)+1);
    title('Baseline 2 (Intermediate)');
    ylabel('Vis. Amplitude');

    subplot(4,1,2);
    if answer == 1,
        scatter(time_data_hr, base2_data, 5, 'r'); % long baseline
        hold on;
    end
    plot(time_sim_hr, base2_sim, 'b', time_sim_hr, base2_sim2, 'm',
time_sim_hr, base2_sim3, 'k');
    axis([min(time_sim_hr), max(time_sim_hr), 0, 0.1]);
    set(gca, 'YTick', 0:0.05:0.11);
    set(gca, 'XTick', min(time_sim_hr):1:max(time_sim_hr)+1);
    title('Baseline 1 (Long)');
    ylabel('Vis. Amplitude');

    subplot(4,1,3);
    if answer == 1,
        scatter(time_data_hr, base3_data, 5, 'r'); % short baseline
        hold on;
    end
    plot(time_sim_hr, base1_sim, 'b', time_sim_hr, base1_sim2, 'm',
time_sim_hr, base1_sim3, 'k');
    if plot_spot,
        sunspot_string = ['brightening + gradient + sunspot: angle = ',
num2str(spotang), ', rad = ', num2str(spotrad/rmax), 'R, size = ',
num2str(spotsize/rmax), 'R, amp = ', num2str(spotamp)];
    else
        sunspot_string = 'brightening + gradient (No sunspot)';
    end
    if answer,
        h = legend(['data: ', input], ['gradient: ', num2str(beta*100.0),
'%'], ['brightening: ', num2str(F*100), '%'], sunspot_string, 4);
    else
        h = legend(['gradient: ', num2str(beta*100.0), '%'], ['brightening:
', num2str(F*100), '%'], sunspot_string, 3);
    end
    set(h,'Interpreter','none');
    axis([min(time_sim_hr), max(time_sim_hr), 0, 1.1]);
    set(gca, 'YTick', 0:0.2:1.1);
    set(gca, 'XTick', min(time_sim_hr):1:max(time_sim_hr)+1);
    title('Baseline 0 (Short)');
    ylabel('Vis. Amplitude');

    subplot(4,1,4);
    if answer == 1,
        scatter(time_data_hr, closurePhase_data, 5, 'r'); % plot actual data
        hold on;
    end
    plot(time_sim_hr, closurePhase_sim, 'b', time_sim_hr, closurePhase_sim2,
'm', time_sim_hr, closurePhase_sim3, 'k');
```

```
    axis([min(time_sim_hr), max(time_sim_hr), -200, 200]);
    set(gca, 'YTick', -200:50:201);
    set(gca, 'XTick', min(time_sim_hr):1:max(time_sim_hr)+1);
    title('Closure Phase');
    xlabel('UT Time (hr)'); % time scale is the same for all plots
    ylabel('Closure Phase (deg)');

    return;

% calculate Sun ra and dec (approximate)
% see Astronomical Almanac page C24 Sun 1999
function [ra, dec] = get_sunra_and_dec(time) % input: float time in seconds
    n = -365.5 + double(time - toSec(1999, 1, 0, 0, 0)) / 86400.0;
    g = (357.528 + 0.9856003 * n) * pi / 180.0;
    lon = (280.460 + 0.9856474 * n + 1.915 * sin(g) + 0.02 * sin(2 * g)) * pi
/ 180.0;
    ecl = (23.439 - 0.0000004 * n) * pi / 180.0;
    ra = double(atan2(sin(lon) * cos(ecl), cos(lon))); % returns atan(y/x) in
radians (between -pi and pi)
    dec = double(asin(sin(lon) * sin(ecl)));
    return; % return a list of the two parameters

% convert from sky to antenna coords (azel mount)
% input: has,decs,lat
%   output: azs = azimuth of source
%           elev = elevation of source
function [azs, elev] = get_radec_az_and_el(has, decs, lat)
    p = sin(decs);
    w = sin(has) * cos(decs);
    r = cos(has) * cos(decs);
    zen = r * cos(lat) + p * sin(lat);
    north = -r * sin(lat) + p * cos(lat);
    elev = atan2(zen, sqrt(north * north + w * w));
    azs = atan2(-w, north);
    if azs < 0,
        azs = azs + pi * 2.0;
    end
    return;

% get Greenwich sidereal time, input: time in seconds since 1970
function d = getGST(ttime)
    secs = (1999-1970)*31536000.0 + 17.0*3600.0 + 16.0*60.0 + 20.1948;
    for i=1970:1999,
        if ((mod(i,4) == 0 && mod(i,100) ~= 0) || mod(i,400) == 0),
            secs = secs + 86400.0;
        end
    end
    % 17 16 20.1948 UT at 0hr newyear 1999
    remainder = rem((ttime-secs),86164.09053);
    decimal = remainder/86164.09053; % decimal part of division
    d = decimal*2.0*pi;
    return;

% convert seconds to Yr/Day/Hr/Min/Sec
function date = toYrDay(secs)
    i = 1970; % initialize 'i' before entering the while loop
    day = floor(secs/86400.0);
    sec = secs - day*86400.0;
    while day > 365,
        i = i + 1;
        temp = ((mod(i,4) == 0 & mod(i,100) ~= 0) | mod(i,400) == 0); %
boolean
        if temp == 1, days = 366;
        else days = 365; end
        day = day - days;
    end
    phr = int64(sec/3600.0); % casting issue with int64(), need to convert to
double
    sec = sec - double(phr)*3600;
    pmin = int64(sec/60.0);
    psec = sec - double(pmin)*60;
    pyear = i;
    day = day + 1;
    pday = day;
    if day == 366, % fix problem with day 366
        temp = ((mod(i,4) == 0 & mod(i,100) ~= 0) | mod(i,400) == 0); %
boolean
        if temp == 1, days = 366;
        else days = 365; end
        if days == 365,
            day = day - 365;
            pday = day;
            pyear = i+1;
        end
    end
    % concatenate strings together to return timestamp as a string
    date = [num2str(pyear), ':', num2str(int64(pday)), ':',
num2str(int64(phr)), ':', num2str(int64(pmin)), ':', num2str(int64(psec))];
    return;

% convert radians to degrees
function conversion1 = toDeg(radians)
    conversion1 = radians*180.0/pi;
    return;

% convert degrees to radians
function conversion2 = toRad(degrees)
    conversion2 = degrees*pi/180.0;
    return;
```

Below is the Python program ***simGradient.py*** developed:

```
# Script reads an input file and runs a simulation.
# Brightness Gradient: used to model pointing error effects on the closure phase, since the closure phase is
# affected by poiting errors.(Old limb brightening profile)
#
# Author: Ted Tsiligaridis
# Last Edited: 2007/07/06, Version: 1.1

from pylab import *; # import everything from pylab
from math import *;  # import math
from Tkinter import *; # for canvas
from scipy.special import *; # for Bessel functions

# input a string, output a float
def getTime(timeStamp):
    # do a delimiter read on the time stamp
    timevec = timeStamp.split(':');
    year = float(timevec[0]); # need to convert string to float
    day = float(timevec[1]);
    hour = float(timevec[2]);
    min = float(timevec[3]);
    sec = float(timevec[4]);
    time = toSec(year, day, hour, min, sec);
    return time; # time in seconds

# get time in seconds since 1970
def toSec(yr, day, hr, min, sec):
    secs = 0.0;
    secs = (yr - 1970)*31536000.0 + (day - 1)*86400.0 + hr*3600.0 + min*60.0 + sec;
    for i in range(1970, int(yr)): # i++, 1970 <= k < yr
        if ((i % 4 == 0 & i % 100 != 0) | i % 400 == 0):
            secs += 86400.0;
    if secs < 0.0: # year must be before 1970
        secs = 0.0;
    return secs;

# input a line as a string, output a tuple of the line
def getTuple(line):
    # do a delimiter read on this line
    return line.split(); # use for any amount of spaces

# turn list into a string
def getString(list):
    return ":".join(list); # put ":" in between strings

# interactive menu
def menu():
    ans = 0;
    while ans != 3:
        print "---- Plot Menu ----"
        print "1. Plot Results";
        print "2. Show Sun";
        print "3. Exit\n";
        ans = int(raw_input("Enter choice: "));
        checkAnswer(ans);
    # out of the loop
    print "Successfully exited menu.";

# check the user's response
def checkAnswer(ans):
    if ans==1:
        plotGraphs();
    elif ans==2:
        showSun();
    elif ans==3:
        print; # do nothing (print blank line)
    else: # default
        print "Please pick one of the choices.";

# plot baselines as a subplot
def plotGraphs():
    figure(1);

    # account for change in text file series of baselines: base3_data <-> base1_data
    subplot(411);
    scatter(time_data_hr, base1_data, s=5, c='r', edgecolor='r'); # intermediate
    plot(time_sim_hr, base3_sim, 'b', time_sim_hr, base3_sim2, 'm', time_sim_hr, base3_sim3, 'k');
    axis([min(time_sim_hr), max(time_sim_hr), 0, 0.5]);
    title('Baseline 2 (Intermediate)');
    yticks(arange(0, 0.51, 0.25));
    xticks(arange(min(time_sim_hr), max(time_sim_hr)+1)); # label x-axis using tick marks from 11 to 23
    ylabel('Vis. Amplitude');

    subplot(412);
    scatter(time_data_hr, base2_data, s=5, c='r', edgecolor='r'); # long
    plot(time_sim_hr, base2_sim, 'b', time_sim_hr, base2_sim2, 'm', time_sim_hr, base2_sim3, 'k');
    axis([min(time_sim_hr), max(time_sim_hr), 0, 0.1]);
    title('Baseline 1 (Long)');
    yticks(arange(0, 0.11, 0.05));
    xticks(arange(min(time_sim_hr), max(time_sim_hr)+1));
    ylabel('Vis. Amplitude');

    subplot(413);
    scatter(time_data_hr, base3_data, s=5, c='r', edgecolor='r'); # short
    plot(time_sim_hr, base1_sim, 'b', time_sim_hr, base1_sim2, 'm', time_sim_hr, base1_sim3, 'k');
    if plot_spot:
        sunspot_string = 'brightening + gradient + sunspot: angle = ' + str(spotang) + ', rad = ' + str(spotrad/rmax) + 'R,
size = ' + str(spotsize/rmax) + 'R, amp = ' + str(spotamp);
    else:
        sunspot_string = 'brightening + gradient (No sunspot)';
    h = legend(('gradient: ' + str(beta*100.0) + '%','brightening: ' + str(F*100) + '%',sunspot_string), loc=0,
shadow=1);
    axis([min(time_sim_hr), max(time_sim_hr), 0, 1.1]);
    title('Baseline 0 (Short)');
    yticks(arange(0, 1.1, 0.2));
    xticks(arange(min(time_sim_hr), max(time_sim_hr)+1));
    ylabel('Vis. Amplitude');

    subplot(414);
    scatter(time_data_hr, closurePhase_data, s=5, c='r', edgecolor='r'); # plot actual data
    plot(time_sim_hr, closurePhase_sim, 'b', time_sim_hr, closurePhase_sim2, 'm', time_sim_hr, closurePhase_sim3,
'k');
    axis([min(time_sim_hr), max(time_sim_hr), -200, 200]);
    yticks(arange(-200, 201, 50));
    title('Closure Phase');
    xlabel('UT Time (hr)');
    xticks(arange(min(time_sim_hr), max(time_sim_hr)+1));
    ylabel('Closure Phase (deg)');
    text(11.5, 100, 'R = ' + str(R), color='k');

    show();
```

```
# show the sun spot on the sun graphically
def showSun():
    canvas = Canvas(width=400, height=400, bg='black');
    canvas.pack(expand='yes', fill='both');
    main = canvas.create_oval(0, 0, 400, 400, width=1, tags='Default Sun', fill='orange');
    v_line = canvas.create_line(200, 0, 200, 400, fill='white');
    h_line = canvas.create_line(0, 200, 400, 200, fill='white');

    # spot's center coordinates
    x = spotrad*cos(spotang*pi/180.0);
    y = spotrad*sin(spotang*pi/180.0);
    center_x = int(200*(1.0-x/rmax));
    center_y = int(200*(1.0+y/rmax));

    # Assume spotamp = 1 (intensity)
    spot_size_pix = int((spotsize/rmax)*200);
    x0 = center_x - spot_size_pix;
    y0 = center_y - spot_size_pix;
    x1 = center_x + spot_size_pix;
    y1 = center_y + spot_size_pix;
    spot = canvas.create_oval(x0, y0, x1, y1, width=1, tags='Sun spot', fill='red');

    name = canvas.create_text(200, 50, text='Sun');
    if apply_gradient:
        gradient_text = canvas.create_text(200, 40, text=str(beta*100.0) + '% brightness gradient');
    if apply_brightening:
        brightening_text = canvas.create_text(200, 30, text=str(F*100.0) + '% limb brightening');
    angular_radius_text = canvas.create_text(30, 10, text='R = ' + str(R), fill='#fff');
    mainloop();

# calculate Sun ra and dec (approximate)
# see Astronomical Almanac page C24 Sun 1999
def get_sunra_and_dec(time): # input: float time in seconds
    n = -365.5 + (time - toSec(1999, 1, 0, 0, 0)) / 86400.0;
    g = (357.528 + 0.9856003 * n) * pi / 180.0;
    lon = (280.460 + 0.9856474 * n + 1.915 * sin(g) + 0.02 * sin(2 * g)) * pi / 180.0;
    ecl = (23.439 - 0.0000004 * n) * pi / 180.0;
    ra = atan2(sin(lon) * cos(ecl), cos(lon)); # returns atan(y/x) in radians (between -pi and pi)
    dec = asin(sin(lon) * sin(ecl));
    return [ra, dec]; # return a list of the two parameters

# convert from sky to antenna coords (azel mount)
# input: has,decs,lat
#   output: azs = azimuth of source
#           elev = elevation of source
def get_radec_az_and_el(has, decs, lat):
    p = sin(decs);
    w = sin(has) * cos(decs);
    r = cos(has) * cos(decs);
    zen = r * cos(lat) + p * sin(lat);
    north = -r * sin(lat) + p * cos(lat);
    elev = atan2(zen, sqrt(north * north + w * w));
    azs = atan2(-w, north);
    if azs < 0:
        azs = azs + pi * 2.0;
    return [azs, elev];
```

```
# get Greenwich sidereal time, input: time in seconds since 1970
def getGST(ttime):
    secs = (1999-1970)*31536000.0 + 17.0*3600.0 + 16.0*60.0 + 20.1948;
    for i in range(1970, 1999):
        if ((i % 4 == 0 & i % 100 != 0) | i % 400 == 0):
            secs += 86400.0;
    # 17 16 20.1948 UT at 0hr newyear 1999
    [decimal, integer] = modf((ttime-secs)/86164.09053);
    return (decimal * 2.0*pi);

# convert seconds to Yr/Day/Hr/Min/Sec
def toYrDay(secs):
    i = 1970; # initialize 'i' before entering the while loop
    day = floor(secs/86400.0);
    sec = secs - day*86400.0;
    while day > 365:
        i += 1;
        temp = ((i % 4 == 0 & i % 100 != 0) | i % 400 == 0); # boolean
        if temp == 1: days = 366;
        else: days = 365;
        day -= days;
    phr = int(sec/3600.0);
    sec -= phr*3600.0;
    pmin = int(sec/60.0);
    psec = sec - pmin*60;
    pyear = i;
    day += 1;
    pday = day;
    if day == 366: # fix problem with day 366
        temp = ((i % 4 == 0 & i % 100 != 0) | i % 400 == 0); # boolean
        if temp == 1: days = 366;
        else: days = 365;
        if days == 365:
            day -= 365;
            pday = day;
            pyear = i+1;
    return getString([str(pyear), str(int(pday)), str(int(phr)), str(int(pmin)), str(int(psec))]); # return timestamp as a
string

# convert radians to degrees
def toDeg(radians):
    rad_to_deg = 180.0/pi;
    return radians*rad_to_deg;

# convert degrees to radians
def toRad(degrees):
    deg_to_rad = pi/180.0;
    return degrees*deg_to_rad;

# Program execution starts -----------------------------------------------------------------

# Actual data

#input = raw_input("Enter input filename: "); # allow user to enter filename
```

```
input = '718614.rad'; print "Reading: " + input;
file = open(input, 'r'); # open for read

# read first two lines
firstline = file.readline();  # read first line
secondline = file.readline(); # ignore second line

value = firstline.split(' '); # delimiter: space; tuple (ordered list)
print value[1] + ": Latitude = " + value[4] + ", Longitude West = " + value[8];

allLines = file.readlines();
print "Number of samples = " + str(len(allLines)); # length of file
print "Reading file...";

# define vectors
timeStamp_data = []; # holds Strings
time_data = []; # holds computed time in seconds
time_data_hr = [];
az_deg_data = []; # holds azimuth angle as a float
el_deg_data = []; # holds elevation angle as a float
daz_deg_data = [];
del_deg_data = [];
freq_data = [];
base1_data = [];
base2_data = [];
base3_data = [];
closurePhase_data = [];
a = 50; b = 50; c = 10; # used for scaling actual baseline data

thirdline = getTuple(allLines[0]); # read first line of timestamped lines
time_stmp = thirdline[0]; # string
time_tuple = time_stmp.split(':');
new_day = int(time_tuple[1]);
time_offset = toSec(int(time_tuple[0]), int(time_tuple[1]), 11, 0, 0);

# loop to read every line of the file
for i in range(len(allLines)): # i = index of sample
  nextline = getTuple(allLines[i]); # break down line
  if nextline[0] == '*': # ignore lines which start with a star
    continue;
  # get data points and add values in vectors(lists) using append
  if int(nextline[9]) != 999: # ignore line if closure phase = 999
    stamp = nextline[0];
    timeStamp_data.append(stamp); # time stamp vector
    nexttime = getTime(stamp);
    time_data.append(nexttime - time_offset + toSec(2007, new_day, 11, 0, 0)); # account for the starting time of
the simulation
    az_deg_data.append(float(nextline[1]));
    el_deg_data.append(float(nextline[2]));
    daz_deg_data.append(float(nextline[3]));
    del_deg_data.append(float(nextline[4]));
    freq_data.append(float(nextline[5]));

    # scale actual data
    base1_data.append(float(nextline[6])*6e-2/sqrt(a*b)); # intermediate
    base2_data.append(float(nextline[7])*6e-2/sqrt(a*c)); # long
```

```
        base3_data.append(float(nextline[8])*6e-2/sqrt(b*c)); # short
        closurePhase_data.append(float(nextline[9])); # closure phase

# Close file
file.close();
print "Text file successfully read.\n";


# Simulation
freq = 11.25e09 + 836e06;
R = raw_input("Enter angular radius of the Sun: ");
rmax = R = float(R); # in degrees
wavl = 299792458.0/freq; # meters

# define vectors
time_sim = []; # holds computed time in seconds
time_sim_hr = [];
az_deg_sim = [];
el_deg_sim = [];
base1_sim = []; base2_sim = []; base3_sim = []; closurePhase_sim = [];
base1_sim2 = []; base2_sim2 = []; base3_sim2 = []; closurePhase_sim2 = [];
base1_sim3 = []; base2_sim3 = []; base3_sim3 = []; closurePhase_sim3 = [];

# insert spot in the sun
spotang = 0;
spotrad = rmax*0.2;
spotsize = rmax*0.1;
spotamp = 1;

# initialize temporary arrays
baz = [0, 0, 0]; blen = [0, 0, 0]; belev = [0, 0, 0];
xx = [0, 0, 0]; yy = [0, 0, 0]; zz = [0, 0, 0];
bblen = [0, 0, 0]; delay = [0, 0, 0];
b_r = [0, 0, 0]; b_east = [0, 0, 0]; b_north = [0, 0, 0];
crr = [0, 0, 0]; phas = [0, 0, 0]; visibility = [0, 0, 0]; # used in plotting
crr2 = [0, 0, 0]; phas2 = [0, 0, 0]; visibility2 = [0, 0, 0];
crr3 = [0, 0, 0]; phas3 = [0, 0, 0]; visibility3 = [0, 0, 0];

# simulation loop
print "Simulation running...";
print "Angular radius of Sun = " + str(R); # display R
numDays = 4;
totalTime = 24*numDays;
for i in range(totalTime): # time loop
    if i % 24 == 0:
        percent = int(i/float(totalTime)*100.0);
        print str(percent) + "% complete.";
    lon = toRad(-71.5); # get latitude and longitude of Haystack
    lat = toRad(42.5);
    hgt = 0.0;
    ttime = toSec(2007, new_day, 11, 0, 0) + i*1800.0/4.0;
    timeStamp = toYrDay(ttime);
    delim = timeStamp.split(':');
    hr = delim[2];

    # compute Sun right ascension and declination
```

```
[ra, dec] = get_sunra_and_dec(ttime); # output: radians

# compute azimuth and elevation angles
ha = getGST(ttime) - ra + lon; # radians
[az, el] = get_radec_az_and_el(ha, dec, lat); # output: radians

time_sim.append(ttime); # record time
az_deg_sim.append(az);
el_deg_sim.append(el);
rap = toRad(286.11); # radians
decp = toRad(63.85);
x = cos(decp)*cos(rap - ra);
y = cos(decp)*sin(rap - ra);
z = sin(decp);
tilt = atan2(y, z*cos(dec)); # PA of sun's rotation axis relation "up" of the disk

# simulation parameters
feedoffset = toRad(11.0);
faz = atan2(sin(feedoffset), cos(feedoffset)*cos(el));
baz[0] = (az + faz + pi/2.0); # azimuth angle
baz[1] = (az + faz - pi/2.0);
baz[2] = toRad(107.0); # radians
blen[0] = 0.5*21.25*2.54/100.0; # length between
blen[1] = 0.5*21.25*2.54/100.0;
blen[2] = (9*12.0 + 7.0 + 7.0/8.0)*2.54/100.0; # 9 ft and 7+7/8 inches --> convert to meters (in. -> m.)
belev[0] = 0;
belev[1] = 0; # lie in approximately the same plane
el_ang = 1 + (2*57.3)/(9*12.0 + 7.0 + 7.0/8.0); # ~2 degrees
elev_offset = -toRad(1)*sin(el);
belev[2] = toRad(-el_ang) + elev_offset;

for stat in range(3):
   lx = blen[stat]*sin(baz[stat]); # local x-y-z
   ly = blen[stat]*cos(baz[stat]);
   lz = blen[stat]*sin(belev[stat]);
   x = -lx*sin(lon) - ly*cos(lon)*sin(lat) + lz*cos(lat)*cos(lon);
   y = lx*cos(lon) - ly*sin(lon)*sin(lat) + lz*sin(lon)*cos(lat);
   z = ly*cos(lat) + lz*sin(lat);
   xx[stat] = x; # store coordinates
   yy[stat] = y;
   zz[stat] = z;

for base in range(3):
   # get sx, sy, sz unit vectors in direction of the sun
   gha = getGST(ttime) - ra; # Greenwich hour angle
   sx = cos(dec)*cos(gha);
   sy = -cos(dec)*sin(gha);
   sz = sin(dec);
   if base < 2:
      bx = xx[base + 1] - xx[0];
      by = yy[base + 1] - yy[0];
      bz = zz[base + 1] - zz[0];
   else:
      bx = xx[2] - xx[1];
      by = yy[2] - yy[1];
      bz = zz[2] - zz[1];
```

```
if hr == 16:
    bblen[base] = sqrt(bx*bx + by*by + bz*bz);

# projected baselines: u, v
b_r[base] = bx*cos(gha) - by*sin(gha);
b_east[base] = by*cos(gha) + bx*sin(gha); # u
b_north[base] = bz*cos(dec) - b_r[base]*sin(dec); # v
delay[base] = bx*sx + by*sy + bz*sz;

# apply a brightness gradient
apply_gradient = 1; # boolean to control application of gradient

# compute Bessel function numerically: uniform disk + linear brigtness gradient
# sun with gradient from antenna mispointing: single integral
sumr = 0.0; sumi = 0.0; r = 0;
#beta = -(0.203971 + 0.239426)/2; % determine this factor to get the correct beamwidth at half power
if apply_gradient:
    beta = -0.1;
else:
    beta = 0.0;
increment = 0.00005;
z_start = sqrt(b_east[base]*b_east[base] + b_north[base]*b_north[base]);
z = 2.0*pi*z_start/wavl; # depends on the angles: ra, dec, and ttime
while r < R: # 0 < r < R (in degrees)
    xa = sqrt(R*R - r*r);
    xb = cos(r*z*pi/180.0);
    sumr = sumr + (xa*xb);
    if apply_gradient:
        xb = sin(r*z*pi/180.0);
    else:
        xb = 0;
    xc = beta*r/R;
    sumi = sumi + (xa*xb*xc);
    r = r + increment; # increment

# scale sum
a = pi*R*R/4.0;
sumr = sumr*increment/a; # divide by pi*R^2/4
sumi = sumi*increment/a;

crr[base] = sqrt(sumr*sumr + sumi*sumi);
phas[base] = toDeg(atan2(sumi, sumr));
visibility[base] = crr[base];

# use Python built-in Bessel function: uniform disk + limb brightening
apply_brightening = 1;
if apply_brightening == 0:
    F = 0.0;
else:
    F = 0.035; # fraction of the Sun's radio output in the enhanced brightness of the limb
y1 = 2*jn(1, R*z*pi/180.0)/(R*z*pi/180.0); # uniform disk
y2 = F*jn(0, R*z*pi/180.0); # limb brightening
y = (y1 + y2)/(1 + F);
crr2[base] = abs(y);
phas2[base] = toDeg(atan2(imag(y), real(y)));
visibility2[base] = crr2[base];
```

```
        # approximate gradient numerically and use previous result
        # uniform disk + limb brightening + linear gradient
        r = 0; sum_g = 0.0;
        while r < R: # 0 < r < R (in degrees)
            xa = sqrt(R*R - r*r);
            if apply_gradient:
                xb = sin(r*z*pi/180.0);
            else:
                xb = 0;
            xc = beta*r/R;
            sum_g = sum_g + (xa*xb*xc);
            r = r + increment; # increment
        a = pi*R*R/4.0; # scale sum
        sum_g = sum_g*increment/a; # divide by pi*R^2/4

        v = y + sum_g; # limb + gradient

        sum2 = 1;
        sumi = 0;
        # sun spot
        plot_spot = 1;
        if plot_spot:
            x = spotrad*cos(spotang*pi/180.0);
            y = spotrad*sin(spotang*pi/180.0);
            th = 2.0*pi*(b_east[base]*(x*pi/180.0) + b_north[base]*(y*pi/180.0))/wavl;
            a = (spotsize/rmax); # a^2: fraction of flux in spot
            v = v + spotamp*a*a*cos(th); # use superposition to add the sums, V = Vspot + Vsun
            sumi = spotamp*a*a*sin(th);
            sum2 = sum2 + spotamp*a*a; # sum used for normalizing

        crr3[base] = sqrt(v*v + sumi*sumi)/sum2;
        phas3[base] = toDeg(atan2(sumi, v));
        visibility3[base] = crr3[base];

    # outside of the 'base' loop
    base1_sim.append(visibility[0]);
    base2_sim.append(visibility[1]);
    base3_sim.append(visibility[2]);
    closurePhase_sim.append(phas[0] - phas[1] + phas[2]);
    base1_sim2.append(visibility2[0]);
    base2_sim2.append(visibility2[1]);
    base3_sim2.append(visibility2[2]);
    closurePhase_sim2.append(phas2[0] - phas2[1] + phas2[2]);
    base1_sim3.append(visibility3[0]);
    base2_sim3.append(visibility3[1]);
    base3_sim3.append(visibility3[2]);
    closurePhase_sim3.append(phas3[0] - phas3[1] + phas3[2]);

print "100% complete.\n";

# convert time vectors to hours before plotting
start_hr = 11;
for j in range(len(time_sim)): # rescale time_sim vector
    hour_sim = (time_sim[j] - toSec(2007, new_day, 11, 0, 0))/3600.0; # account for sim. start time
    time_sim_hr.append(hour_sim + start_hr); # start at 11 hours
```

```
for k in range(len(time_data)): # rescale time_data vector
    hour_data = (time_data[k] - toSec(2007, new_day, 11, 0, 0))/3600.0;
    time_data_hr.append(hour_data + start_hr);

# interactive menu
menu();
```