Determination of Sun Effective Temperature Through Analyzing Center-to-Limb Darkening

Maksym Brilenkov[*], Ruslan Brilenkov[†], and Oleg Kalinin[‡]

University of Rome 'Tor Vergata', Rome, Italy

October 18, 2021

Abstract

This work was dedicated to the determination of the solar effective temperature via center-to-limb darkening theory. Using the CCD camera ATIK CAMERAS ATIK-320E and the mechanical setup, we measured the focal distance of our setup to be $f = (45.65 \pm 0.057)$cm and acquired Sun images in different waveband filters, with two different camera supplies/charges. After that, we wrote the self-consistent GUI application in Python 2.7 with the implementation of wxWidget libraries to enable solving similar tasks easier in future. By calculating $I_\lambda(0, \tau_\lambda)$ and $T(\tau_\lambda)$ for different filters (and different power supplies), we found effective temperature to be: $T_{eff}^{(new)} = (6055 \pm 189)\, K$ and $T_{eff}^{(old)} = (6175 \pm 201)\, K$.

1 Introduction

The goal of our work was the determination of Sun effective temperature; thus, we applied the center-to-limb darkening theory developed by [2]. As the detailed analysis of the CCD camera ATIK CAMERAS ATIK-320E (1618x1219) was done by [1] via Photon Transfer Curve method, our efforts was concentrated mostly on programming part, i.e. we developed the self-consistent GUI application which will help to solve similar tasks in future.

We used the same optical setup as [1] and [2]. By applying several measurements we found the focal distance of our system and also obtained Sun images for UBVRI filters for two different power-supplies (we refer to them as *new* and *old* power supplies). This enables us to calculate the so-called *blooming effect*. As the authors of [1] stated that the response of the sensor was different for two perpendicular directions, we also applied corrections to our images.

This report organized as follows. In section (2), we state the theoretical background of Sun photosphere temperature. In sections (3), (4) we discuss the observational/experimental part of the work summarizing the theoretical background and characterization of CCD as well as optical system used. In section

[*]Maksym.Brilenkov@student.uibk.ac.at
[†]Ruslan.Brilenkov@student.uibk.ac.at
[‡]Oleg.Kalinin@student.uibk.ac.at

(6) and 6 we remind reader theoretical background and data analysis techniques. In section (7) we discuss the advantages of our GUI program. In section (8) we give suggestions for future improvements. Results were summarized in the conclusion section (9). The code can be found in the Appendix.

2 Theoretical background: Theory for Sun photosphere temperature

Let us establish the theoretical background (concepts and definitions) of the work. In particular, we will briefly consider radiative transfer and approximations like plane-parallel and Eddington. (see also [2] for similar description).

2.1 Definitions and Radiative transfer equation

Effective temperature gives us the temperature which a black body would need to have if it were to radiate the same amount of energy per cm^{2}s as the star. (from [3] p.23)

Lets consider a surface area of size $d\sigma$ and light passing through it perpendicularly in a narrow cone of opening $d\omega$ (see [3] from page 26 and further). The amount of energy E_λ going per second through the area $d\sigma$ is proportional to $d\sigma$, and to the opening of the cone $d\omega$. the energy E_λ passing through this area per second is given by

$$E_\lambda = I_\lambda d\omega d\sigma d\lambda, \quad \rightarrow \quad I_\lambda = \frac{E_\lambda}{d\omega d\sigma d\lambda}\,, \tag{1}$$

i.e., it increases proportionately to the opening of the cone, the size of the area, and the width of the wavelength band $d\lambda$. According to the definition (1), the intensity I_λ is then the energy which passes per second through n area of 1 cm^2 into a solid angle $\delta\omega = 1$ in a wavelength band $\delta\lambda = 1$.

Let us imagine the beam of light passing through the volume of gas, which has a rectangular shape with the σ being its cross-section and s being its length, which corresponds to an optical depth $\tau_\lambda = \int_0^s \kappa_\lambda ds$ (i.e. the specific level of transparency of the material). Initially, the beam has the energy $E_{\lambda 0}$. When the light beam passes along the path element ds there is some absorption (represented by absorption coefficient κ_λ) and the energy E_λ is reduced by the amount

$$dE_\lambda = -\kappa_\lambda E_\lambda ds = -\kappa_\lambda ds I_\lambda d\omega d\lambda d\sigma\,. \tag{2}$$

There is also an emission from the volume $dV = d\sigma ds$ given by

$$dE_\lambda = \epsilon ds d\omega d\lambda d\sigma\,. \tag{3}$$

where ϵ_λ now is the amount of energy[1] emitted each second per unit volume into $\delta\omega = 1$ per wavelength band $\delta\lambda = 1$. Absorption and emission together give us

[1]It is worth to mention, despite the name "coefficient", there is big difference between emission and absorption coefficients, such as the first one is the energy, while the second is just the coefficient

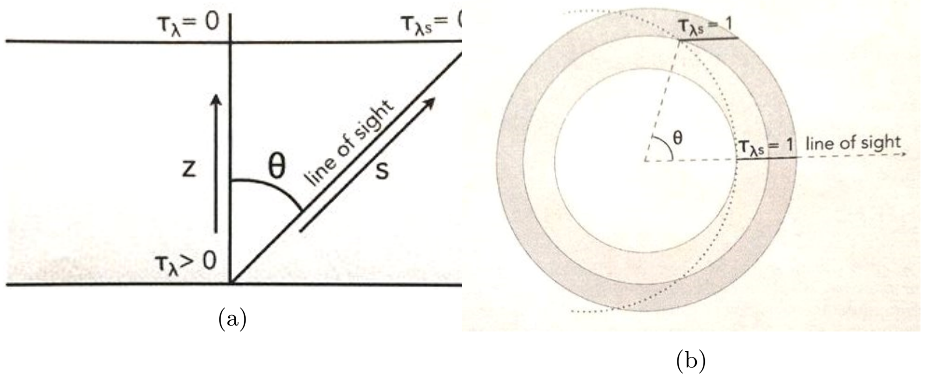


Figure 1: (a) Visual representation of the plane parallel approximation (credits to [2]); (b) Result of the Eddington approximation. Dashed line stands for the ideal surface on which $\tau_{\lambda s} = 1$ (credits to [2])

$$dE_\lambda = dI_\lambda d\omega d\lambda d\sigma = -\kappa_\lambda I_\lambda d\omega d\lambda d\sigma ds + \epsilon_\lambda ds d\omega d\lambda d\sigma \,. \tag{4}$$

Dividing by $d\omega d\lambda ds$, we obtain in the limit $ds \to 0$

$$\frac{dI_\lambda}{ds} = -\kappa_\lambda I_\lambda + \epsilon_\lambda \,. \tag{5}$$

This gives

$$\frac{dI_\lambda}{ds} > 0 \quad \text{if} \quad \epsilon_\lambda > \kappa_\lambda T_\lambda \,, \quad \text{and} \quad \frac{dI_\lambda}{ds} < 0 \quad \text{if} \quad \epsilon_\lambda < \kappa_\lambda T_\lambda \quad . \tag{6}$$

If we divide equation (5) by κ_λ, we obtain

$$\frac{dI_\lambda}{\kappa_\lambda ds} = \frac{dI_\lambda}{d\tau_{\lambda s}} = -I_\lambda + \frac{\epsilon_\lambda}{\kappa_\lambda} = -I_\lambda + S_\lambda \,, \tag{7}$$

where the new symbol was introduced

$$S_\lambda = \frac{\epsilon_\lambda}{\kappa_\lambda} \quad \to \quad \epsilon_\lambda = S_\lambda \kappa_\lambda \,, \tag{8}$$

Equation (7) is the *radiative transfer equation* in the plane parallel case. S_λ is generally called the *source function.*

2.2 Plane-parallel approximation

In general, solving the radiative transfer equation is a tedious work. Luckily, some simplifications can be made in study of stellar atmospheres. Considering the outer layers of a star, which are relatively low in density (in comparison with the star itself), we can neglect convective heat transport, so radiation is the only significant method of transporting energy. Furthermore, gradients in specific intensity and temperature are much greater in the radial directions than in the transverse ones. Thus, a stellar atmosphere can usually be well approximated as a plane-parallel system, in which properties of the atmosphere, such as T, σ_λ and ϵ_λ depend only on a vertical coordinate z (see figure 1a).

In the plane parallel approximation, the specific intensity I_λ depends only on z and on the angle θ between the direction of the light ray (or the line of sight, l.o.s.) and the z axis: when the light travels straight upward (in the same direction that l.o.s.) that z increases, $\theta = 0$; when the light travels straight down (in the opposite way to l.o.s.), $\theta = \pi$. In traversing a thin layer of the atmosphere, of vertical thickness dz, the distance traveled by light is

$$ds = \frac{dz}{\cos\theta}\,, \tag{9}$$

implying $(ds)^2 \gg (dz)^2$. The radiative transfer equation in a plane parallel atmosphere thus takes the form

$$\cos\theta \frac{\partial I_\lambda}{\partial z} = -\kappa_\lambda \left(I_\lambda - S_\lambda\right)\,, \tag{10}$$

where we assumed the emission being thermal. The above equation can be rewritten as

$$I_\lambda = S_\lambda - \frac{\cos\theta}{\kappa_\lambda}\frac{\partial I_\lambda}{\partial z}\,, \tag{11}$$

Note that when $\cos\theta = 0$, indicating that a light ray is running parallel to the star's surface, the specific intensity I_λ is equal to the source function S_λ. When $\cos\theta \neq 0$, there is a correction term proportional to the gradient of I_λ in the z direction.

How can we solve equation (11) for light traveling at an arbitrary angle θ in a real star's atmosphere? Let us make the assumption that the atmosphere is homogeneous and isotropic; this will be our zeroth order approximation

$$I_\lambda^{(0)} = S_\lambda^{(0)}\,, \tag{12}$$

However, employing the definition of source function (in form of $\kappa_\lambda S_\lambda = \kappa_\lambda B_\lambda$) we have

$$I_\lambda^0 = B_\lambda\,. \tag{13}$$

Thus, a homogeneous isotropic atmosphere in which the emission is thermal produces a blackbody spectrum. Now let's take into account the fact that the specific intensity is not perfectly homogeneous, but has a gradient in the z direction. By plugging our zeroth order approximation, $I_\lambda^{(0)} = B_\lambda$, back into equation (11), we find that

$$I_\lambda^{(1)} = B_\lambda - \frac{\cos\theta}{\kappa_\lambda}\frac{\partial B_\lambda}{\partial z}\,, \tag{14}$$

Note that if the temperature T decreases as you go upward in the atmosphere, then the upward specific intensity ($\cos\theta = 1$) is slightly greater than it would be in a homogeneous atmosphere, and the downward specific intensity ($\cos\theta = -1$) is slightly less. This means that there is a net upward flux of light energy in this case.[2]

[2]In other words, energy flows from regions of higher temperature to regions of lower temperature. This is reassuringly consistent with the second law of thermodynamics.

Once more, what have we done till now? We defined the vertical optical depth measured along orthogonal direction to parallel planes, in other words, along the l.o.s. (see figure 1a): $d\tau_\lambda = \cos\theta d\tau_{\lambda s}$. As we measure the intensity of the solar radiation which is leaving the photosphere (in the analysis of solar images), $d\tau_{\lambda s} = 0$. We want to get an expression of $I_\lambda\,(0, \mu = \cos\theta)$ in terms of source function $S_\lambda(\tau_\lambda)$ (equation (11)). In order to do that (see [3], chapter "Radiative transfer in stellar atmospheres", p.39), we assume the source function S_λ being the product of the power series in τ_λ of order m.

$$S_\lambda = I_\lambda\,(0,1) \sum_{n=0}^{m} a_{\lambda n} \tau_\lambda^n, \quad I_\lambda(0,\mu) = I_\lambda(0,1) = \sum_{n=0}^{m} a_{\lambda n} n! \mu^n\,, \tag{15}$$

where $I_\lambda\,(0, \mu = 1)$.

2.3 Eddington and Two-Stream Approximation

Starting from the Rosseland approximation, which basic idea is that the intensities approach the Planck function at large effective depths in the medium (see [4] p.42), we move to Eddington approximation. Which tells that the intensities approach isotropy, and not necessarily their thermal values. Because thermal emission (and, in general, scattering) are isotropic, one expects isotropy of the intensities to occur at depths of order of an ordinary mean free path; thus the region of applicability of the Eddington approximation is potentially much larger than the Rosseland approximation, the latter requiring depths of the order of the effective free path. With the use of appropriate boundary conditions (here introduced through the two-stream approximation) one can obtain solutions to scattering problems of reasonable accuracy at all depths.

The assumption of near isotropy is introduced by considering that the intensity is a power series in μ, with terms only up to linear:

$$I_\lambda(\tau_\lambda, \mu) = a_\lambda(\tau) + b_\lambda(\tau)\mu\,. \tag{16}$$

Another approach to the problem was given by [5], where the author also started from the special case of *two-stream approximation*[3], (*Eddington approximation*), but considered source function being linear in optical depth τ, instead of intensity being linear in distance parameter μ (both are in the first order approximation):

$$S_\lambda(\tau_\lambda) = a_\lambda + b_\lambda \tau_\lambda\,, \tag{17}$$

By some passages and considerations given by [2], we are lead to the conclusion that if we assume plane parallel approximation to be valid, for every observation angle we see into Sun's atmosphere down to an optical depth $\tau_{\lambda s} = 1$. This means that we see up to different physical depths in function of the distance from the center.

Assuming Eddington approximation with the local thermodynamic equilibrium, from (15) we get

$$T^4(\tau) = \frac{3}{4} T_{eff}^4 \left(\tau + \frac{2}{3}\right)\,. \tag{18}$$

[3]an approximation in which radiation is propagating in only two discrete directions (It was first used by Arthur Schuster in 1905; see [5] p. 106)

Since photosphere is defined as the layer on which effective temperature T_{eff} is equivalent to real temperature T, from (4) we get

$$T_{eff}^{phot} = T\,(\tau = 2/3)\ . \tag{19}$$

Even if photosphere doesn't exhibit thermal equilibrium, in order to find a relation between temperature and intensity of radiation we can suppose to have local thermal equilibrium.

Starting from the Planck's law expressed in terms of wavelength λ

$$B_\lambda(\lambda, T) = \frac{2hc^2}{\lambda^5}\frac{1}{e^{\frac{hc}{\lambda k_B T}} - 1}\ , \tag{20}$$

after several algebraic operations we get

$$T(\tau_\lambda) = \frac{hc}{k\lambda}\frac{1}{\log\left(1 + \frac{2hc^2}{\lambda^5 B_\lambda(\lambda,T)}\right)}\ . \tag{21}$$

Supposing the atmosphere having local thermal equilibrium and appealing to the equations (8), (12) and (13) leads

$$T(\tau_\lambda) = \frac{hc}{k\lambda}\frac{1}{\log\left(1 + \frac{2hc^2}{\lambda^5 S_\lambda(\tau_\lambda)}\right)}\ . \tag{22}$$

In the next section we briefly describe our experimental base together with our detailed preparation and measurements.

3 CCD

In this section we briefly describe the theoretical background for the *Charge-Coupled Device* (CCD) and also give the characteristics of the camera we used (see [1], [6] for more details)

3.1 Characteristics of the CCD

3.1.1 Operation and performance

The CCD is memory device, where the physical quantity, which represents a bit of information (in order to function as memory) by means of recognition the presence or absence of the bit (reading) and by means of creation or destruction of the bit (writing or erasing), is usually represented as the packet of charges - electrons or holes. These charges are stored in the depletion region of a *metal-oxide-semiconductor* (MOS) capacitor (see figure (2)). Charges are moved about in the CCD circuit by placing the MOS capacitors very close to one another and manipulating the voltages on the gates of the capacitors, so as to allow the charge to spill from one capacitor to the next. Thus the name "charge-coupled" device. A charge detection amplifier detects the presence of the charge packet, providing an output voltage that can be processed. Charge packets can be created by injecting charge that is from an input diode next to a CCD gate or introduced optically. Like the magnetic bubble device, the CCD is a serial device where charge packets are read one at a time.

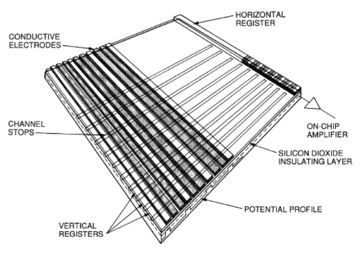


Figure 2: Primary components that make up a three-phase CCD (credits to [6])

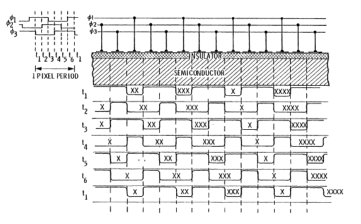


Figure 3: Three-phase potential well timing diagram showing signal charge being transferred from left to right (credits to [6])

The simplest way MOS capacitor is arranged to form a CCD image array, is a *three-phase device*. So, the gates are arranged in parallel with every third gate connected to the same clock driver. The basic cell in the CCD, which corresponds to one *pixel*, consists of a triplet of these gates, each separately connected to phase-1, -2 and -3 clocks making up a pixel register. Figure (3) shows the layout of a three-phase CCD area array and how charge is transferred in a three-phase CCD register. The image-forming section of the CCD is covered with closely spaced vertical *parallel registers*. During integration of charge, two vertical phases are biased high (e.g., phase-1 and -2), which results in a high field region that has a higher electrostatic potential relative to the lower biased neighboring gate (phase-3).The dark phases shown represent this bias condition, and it is under these gates where signal electrons collect in a pixel when the CCD is exposed to light. These phases are called *collecting phases*. The phase that is biased at lower potential is called the *barrier phase* since it separates charge being collected in the pixels on either side of this phase.

Vertical registers are separated by potential barriers called *channel stops* which prevent the spread of the signal charge from one column into another. A CCD image is read out by a succession of vertical shifts through the vertical registers. For each vertical shift a line of pixels is transferred into a horizontal register (also called a *serial register*) which is oriented at right angles to the vertical registers. Before the next line is shifted the charge in the horizontal register is transferred to the output amplifier. The amplifier converts the charge contained in each pixel to a voltage. The device is serially read out line-by-line, pixel-by-pixel, representing the scene of photons incident on the device.

3.1.2 Performance functions

The CCD must perform four primary tasks in generating an image, i.e. the performance functions, which are called *charge generation*, *charge collection*, *charge transfer* and *charge measurement*.

- ***Charge generation*** is the ability of a CCD to intercept an incoming photon and generate an electric charge. *Charge generation efficiency* (CGE) is described by a function called *quantum efficiency* (QE), which is the fraction of incident photons that produces a useful charge in the silicon chip (i.e. it gives a measure of how many incident photons get converted into electrical charge). It's defined as

$$QE = \eta_i \frac{P_i}{P}\,, \tag{23}$$

where η_i is the quantum yield gain (in e^-/interacting photon), P is the average number of incident photons per pixel and P_i is the average number of interacting photons per pixels.

An ideal CCD would have 100% QE at all wavelengths (i.e., all the *interacting photos* will be transformed into electric signals due to *photoelectric effect*, which was theorized by Albert Einstein, for which he was awarded the 1921 year's Nobel Prize), but in reality QE is highly dependent on wavelength, falling off steeply at the near-infrared (NIR) and blue wavelengths.

- ***Charge collection*** is the ability of the CCD to accurately reproduce an image from the electrons generated. Three parameters are included to describe this process: (1) the area or number of pixels contained on the chip, (2) the number of signal electrons that a pixel can hold (i.e., charge capacity of a pixel), and (3) the ability of the "target pixel" to efficiently collect electrons when they are generated. The last parameter is most critical because it defines the spatial resolution performance for the CCD.

  Full-well capacity is a measure of how many electrons a pixel can hold. The bigger is better, since the greater the *full-well capacity*, the greater a CCD's dynamic range and signal-to-noise. *Dynamic range* is the ratio between the brightest and faintest objects that can be simultaneously discriminated (or, putting this differently, it gives a measure of the maximum signal variation over which the CCD still behaves linearly). It is defined as the

$$DR = 20 \log \frac{FWC}{\sigma_{tot}}\,, \tag{24}$$

  where σ_{tot} is the camera noise.

  Large pixels can hold more charge but require more silicon, which in turn degrades the production and increases the price. The smallest pixel that can be manufactured is only a couple of microns in extent, set by lithography design and process rules. Also, charge capacity is an important consideration for very small pixels because full well decreases by pixel area. The advantage of small pixels is higher yield with fewer shorted devices. Production yield is roughly inversely proportional to the area used on the wafer by the CCD. Therefore, yield will be approximately four times greater for a 12-m pixel than a 24-m pixel, assuming the same pixel count.

- ***Charge transfer*** is described by *Charge transfer efficiency* (CTE) (i.e. the fraction of electrons successfully transfered per pixel transfer) performance to the limit. Charge is transferred due to fringing fields between phases, thermal diffusion, and self-induced drift. The relative importance of each effect is dependent primarily on how much charge is being transferred.

- ***Charge Measurement*** is the detection and measurement of the charge collected in each pixel. This is accomplished by dumping the charge onto a small capacitor connected to an output Metal Oxide Semiconductor Field Effect Transistor (MOSFET) amplifier, also called the "sense node" or "output diode." The output amplifier generates a voltage for each pixel proportional to the signal charge transferred.

Additional important performance factors are (for more details refer to [1]):

- ***Linearity*** describes the CCD in standard conditions as a linear function over a wide dynamic range (in other words, output voltage signal is proportional to the amount of incoming light in a wide range or the signal is proportional to the exposure time).

- ***Gain***, g, is the conversion between the number of electrons recorded by CCD and the number of digital units (ADU) contained in the image, so it's a measure of how many electrons correspond to an ADU.

3.1.3 CCD camera noises

- ***Photon shot noise***

  Due to the quantum nature of light, *shot noise* originates from the random arrival of photons. While an effect of other noises can be partially removed, shot noise will always be present since each photon is an independent event; thus, the arrival of any given photon cannot be precisely predicted. The probability of its arrival in a given period of time is governed by a Poisson distribution. Taking into account that the average number of photo-electrons S created per pixel is proportional to the number of incident photons P_l through the quantum yield gain η_i, we get

  $$\sigma_{phot}(P_l) = \sqrt{P_l} \propto \sqrt{S}\,, \tag{25}$$

- ***Fixed pattern noise (FPN)***

  Although modern CCDs are build to match all standards, they still are not perfect, which means that each pixel has a slightly different sensitivity to light (1% to 2% of the average signal). FPN originates from this fact and it will be the same (fixed) for any image taken in the same conditions (same signal and exposure time). This effect can be reduced by calibrating an image with a *flat-field image.*

  FPN is defined as

  $$\sigma_{fpn} = P_{fpn} S\,. \tag{26}$$

  where P_{fpn} is the FPN quality factor (usually ~ 0.01).

- ***Dark current***

  Whether the CCD is being exposed to light or not, they will develop so-called *dark current*, which is caused by thermally generated electron-hole pairs build up in the pixels of all CCDs. Its accumulation rate depends on the temperature of the CCD, but will eventually completely fill every pixel. The pixels in a CCD are cleared before beginning an exposure, but dark current starts accumulating again immediately. Therefore, this effect is very important for the observations which require long exposure times. However, the cooling of the CCD camera can reduce this effect by a factor of 100 or more. The remaining dark current is subtracted from an image using dark frames.

- ***Read-out noise (RON)***

  Read-out noise consists of two consequent operations. One is the result of the conversion of the analog signal (voltage) into a digital signal through the Analog-to-Digital Converter (ADC) (typically described by a Gaussian distribution), while another is due to the electronic components, which introduce additional electrons.

Supposing the three noises are independent one from the other and neglecting dark noise, it is possible to get an expression for the camera total noise as:

$$\sigma_{TOT} = \sqrt{\sigma^2_{RON} + \sigma^2_{shot} + \sigma^2_{fpn}}\,. \tag{27}$$

Furthermore, in practice it is possible eliminate the FPN and find the relation between σ_{tot}, S and σ_{RON} in the linear regime as:

$$\sigma^2_{tot} = \frac{S}{g} + \left(\frac{\sigma_{RON}}{g}\right)^2, \tag{28}$$

This equation was used by our predecessors (see [1]) to calibrate the CCD camera.

3.1.4 Parameters of the CCD

Sensor Type	CCD - Sony ICX274
Resolution	1620x1220
Pixel Size (μm)	4.4 x 4.4
ADC	16bit
Readout Noise	$3e^-$
Power	12v DC 0.8A
Backfocus distance	13mm
Cooling Delta	-25
PC	Pentium II PC with 64MB RAM
ODD	CD-ROM drive
OS requirements	Windows 98/XP/Vista
Interface	USB 2.0 High Speed
Weight	350 g /12.34 ounces

Table 1: Technical specification of the camera.

In this part we list the specifications of the ATIK 320E camera used in our experimental setup (see table (1)). Its accurate study and calibration was done by [1] (see section 4 in the work). We summarize the results of their work in table (2).

Name	Symbol	Value
Gain	g	$(0.1285 \pm 0.0003)\, e^-/ADU$
Read-out noise (RON)	σ_{RON}	$(8.0 \pm 0.4)\, e^-$
Maximum number of ADU	S_M	$(64270 \pm 140)\, ADU$
Full-well capacity (FWC)	$g \times S_M$	$(8260 \pm 40)\, e^-$
Dynamic Range (DR)	$20 \times \ln\left(\frac{FWC}{\sigma_{RON}}\right)$	$(60 \pm 3)\, dB$

Table 2: Summary of parameters estimated in [1], which describes CCD camera we used.

4 Instrumentation

We used the same optical system as [2], but upgraded with the mechanical focusing system by [1] in order to able to determine exact focal length (for details see section (4.2)). The focal length in our experiment was $f = (45,65 \pm 0,04)$.

An optical system included the telescope (see figure (5)), filter wheel ("Starlight Express") with U, B, V, R, I bands, CCD camera "ATIK CAMERAS ATIL-320E" (for detailed description of the CCD see section (3)) and Thousand Oak Black Polymer Filter (for details see [1]). Also, an equatorial mount and important software ("Artemis ATK Capture" (Version 3,09) and "SX Filter Wheel") in order to maintain the camera and filter wheel during the experiment. Moreover, we used two different power supplies for the camera (the reasons are discussed in section (6.5)).

4.1 Preparations and Calibration

First of all, the parameters and working conditions of our equipment were determined . Furthermore, the usage of the wrong power supply for the camera leads to some particular distortions of the image (see also [1] and [2] for discussion). Possible solution of the problem was proposed by Prof. Giovannelli and discussed in subsection (6.5)).

Secondly, we assemble and set up the telescope system to conduct the observations (see section (5)). The obtained data was analyzed with an algorithm presented in section (6).

In order to assemble the system, simply speaking, we should know how to do this, and where to place each element. One of the main parameters of the telescope is the focal distance (more precisely, the point where the light beams converge, which should coincide with the camera sensor, otherwise there will not be the desirable picture). We had to do the calibration of the telescope, in other words, appealing to geometrical Optics (equation (23)), we determine all the both the geometrical and the focal lengths.

4.2 Focal length

To obtain better focal length, we needed to conduct several measurements of it. We mounted the telescope, the plane-light source (consisting of LED lamp and transparent plane) and the screen on the top of the "SMART" table (the tool isolated from the ground noises) provided by the UoRTV Solar Laboratory (see the figures (4) and (5)).

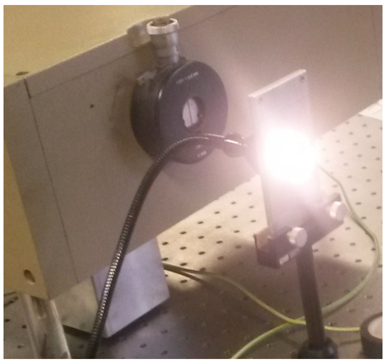
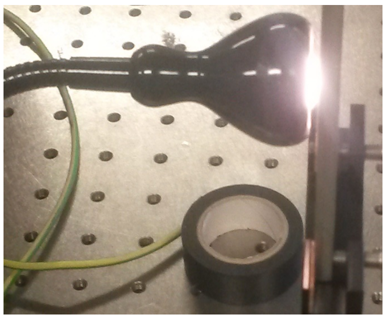



Figure 4: LED lamp and the plane in the measurement of the focal distance.

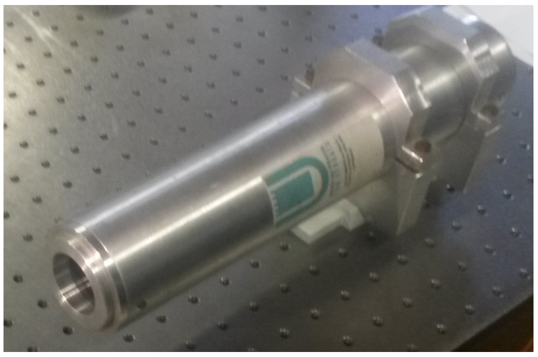


Figure 5: The telescope during the measurements of the focal distance.

The fundamental equation (which relates p, q and f, the distances between the object and the lens, the lens and the image and the focal distance) for thin lens

$$\frac{1}{p} = \frac{1}{f} - \frac{1}{q}\,, \tag{29}$$

allows us to determine the focal distance. In our case, the source is the LED lamp, the lens is the telescope lens and the image of the source is the screen. Keeping in mind that we cannot change the position of the Astronomical source, like our Sun, we fixed the position of the lamp and moved the telescope. With the help of the screen we determined p, q and f employing an equation (29). The results are listed in the table (3).

Measure	p, cm	q, cm	f, cm
1	$104,4 \pm 0,03$	$81,5 \pm 0,05$	$45,5 \pm 0,04$
2	$93 \pm 0,03$	$89,1 \pm 0,05$	$45,8 \pm 0,04$
Average	$98,7 \pm 0,03$	$85,3 \pm 0,05$	$45,65 \pm 0,04$

Table 3: Geometrical scales and focal distance of the telescope we used.

Note: we assumed any ray parallel to the axis on one side from the lens, converges to the focal point on the other side (and vice versa, any ray coming from the focal point, exits parallel to the axis on the other side), also that any ray does not change its direction if it passes thought the center of the lens. As seen in the table (3), our focal distance is $f = 45,65 \pm 0,04cm$ in comparison with [1] ($44,7 \pm 0,3cm$)

4.3 Telescope system and observations

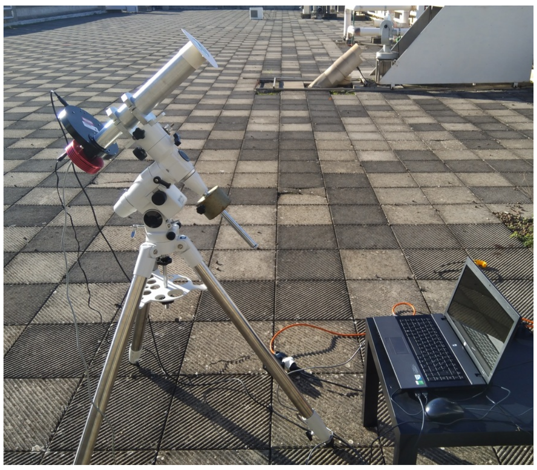


Figure 6: Instrument system during the observations.

Before conducting the observations we calibrated an assembled telescope system (depicted at picture (6)) for optimal use.

5 Observations

We used a motor-powered equatorial mount in the same way of [1] and [2] and acquired one data set (in contrast with 2 data sets of [1] who did not get all the data in one day), but with the same experimental setup (picture (6)).

It is worth to mention, that equatorial mount (figure (6)) works as following: while pointing the North Pole (Northern Star), it is able to track the declination and right ascension, simply speaking it is able to track the Sun going on the ecliptic plane. But if it points to any other direction, then it is not able to follow the Sun correctly, which we experienced during our observations.

5.1 Dataset: December 5, 2017

We observed Sun in 4 different positions of filter wheel (4 filters and one without filter), made 20 snapshots per filter and repeated twice for two different adapters (power supply for our CCD camera). Due to mechanical focusing system (for details see [1]), we optimized the focusing for each single filter by changing the length of the focusing system itself. This procedure enables us to minimize the effect of spherical aberration, which relates to the focusing differences in different wavelengths. This procedures were repeated twice for two different power supplies (adapters) of the CCD camera. Also, the snapshots of flat-field (in other words, background or sky) for each filter were made.

Due to the differences in the transmissivity profiles of each waveband (see figure (7)), we had the different exposure time, summarized in the table (4). It is clearly seen, that the U-band filter has a very long exposure time (7 sec.) in comparison to the other ones. It means, that in order to get enough light from the Sun through the U-filter, we have to keep our telescope pointing the Sun for 7 seconds, which was impossible due to the reason given at the beginning of this section.

Filter	Exposure time (sec)
I	7×10^{-2}
R	7×10^{-2}
V	7×10^{-1}
B	7×10^{-1}
U	7×10^{0}

Table 4: The table shows the set of filters with the exposure times for each.

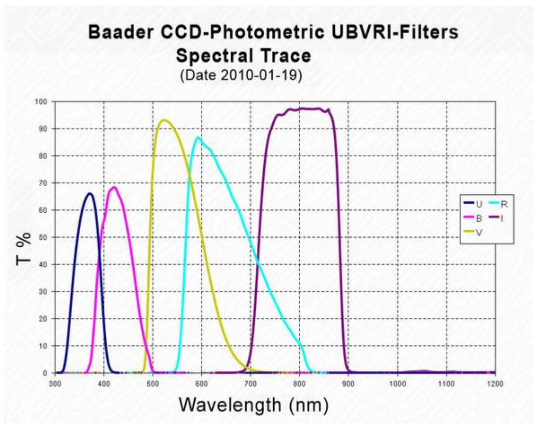


Figure 7: Transmissivity profiles of UBVRI-filters.

As was described above, without precise adjustment of equatorial mount and ecliptic, our telescope was not able to observe Sun without changing its position. Hence, having very long exposure time (which is required for smaller wavelength, say U,B in comparison with R,I), Sun moves with respect to the camera and the image becomes blurred, elongated and not readable for the future analysis).

6 Data analysis

6.1 Definitions and algorithm

In the next section we will closely refer to the techniques and the notations described in [2]. To describe the distance from the center of the Sun, we use the distance parameter μ:

$$\mu = \cos\theta = \sqrt{1 - \left(\frac{r}{R}\right)^2} \tag{30}$$

where θ is the heliocentric angle, R is the Sun's radius and r is the current distance to the center of the disk. It is seen that distance parameter decreases with an increasing radius (being maximum in the center).

For each single snapshot, we divide the Sun for 9 concentric rings with the center in the solar center, later on, we consider only the horizontal diameter (note that the regions with the same distance by the center in the two semi-diameters are joined, are considered together). Every region has $N \approx 100$ points (see [2] for similar reasoning).

We number the regions with the symbol k. The k-th region has a number N_k of pixels and extends from the minimum distance (from the center) r_k^{in} to the maximum r_k^{out}. The mean distance r_k from the center and its associated error δr_k (similar to the average and the semi dispersion on r_k^{in} and r_k^{out}) are defined as:

$$r_k = \frac{r_k^{in} + r_k^{out}}{2}, \quad \delta r_k = \frac{r_k^{in} - r_k^{out}}{2}\,. \tag{31}$$

To get more information from our data, we firstly find an intensity of each pixel I_k^n in the k-th region and then get an average intensity I_k in this ring. as error we take the standard deviation over the I_k^n. We also define the distance parameter μ_k of the k-th region

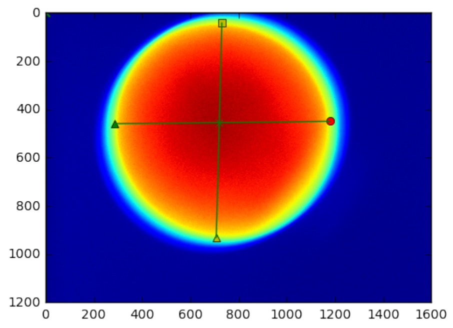
$$I_k = \frac{1}{N_k}\sum_{n=1}^{N_k} I_k^n \pm \Delta_k I_k, \quad \mu_k = \sqrt{1 - \left(\frac{r_k}{R}\right)^2} \pm \frac{r_k}{\sqrt{1 - \left(\frac{r_k}{R}\right)^2}}\delta r_k\,. \tag{32}$$

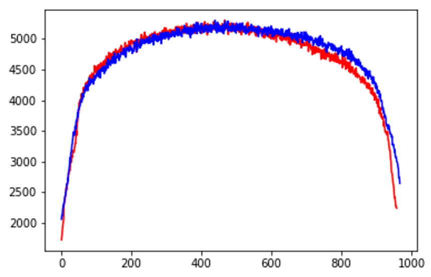
Note that $\delta\mu_k$ is not a measure error, but depends on the way in which we define the regions on the diameter.

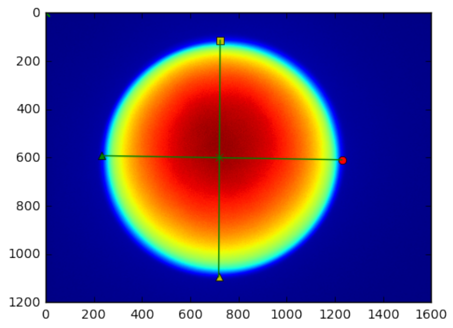
During our observations we have collected 20 snapshots of the Sun for 4 positions of the filter wheel (see details in section (4)). In order to analyze them, we developed an algorithm in the Python environment. The program goes filter by filter, repeated over all 20 snapshots (collected in our experiment) and finds the coordinates of the center, as well as four recognizable points as (Left, Right, Top and Bottom of the solar snapshot, from the point of view of our CCD camera, see left figures (8)), defines the regions on the diameters and calculates all the quantities from above. At last, the final intensity I_k^{FIN} and the distance parameters μ_k^{FIN} are obtained (for the given filter) through the weighted average over all twenty images.

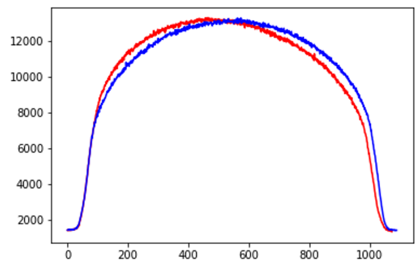
6.2 Second order fit

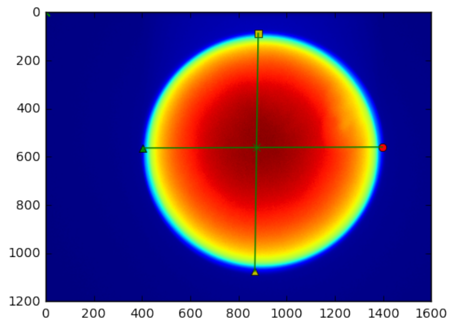
In section (2) we mentioned about the possibility to express the center-to-limb intensity and the source function ($I_\lambda(0,\mu)$ and $S_\lambda(\tau_\lambda)$) in terms of the power series in distance parameter and optical depth (μ and τ_λ), respectively. Moreover, looking at the Sun, we gather more atmosphere closer to the edge (limb) than to the center, while as τ is bigger at the edge (limb) and smaller in the center. However, distance parameter μ behaves in opposite way (maximum in


(a) Snapshot for B filter


(b) Intensity plot for the B filter


(c) Snapshot for V filter


(d) Intensity plot for the V filter


(e) Snapshot for R filter

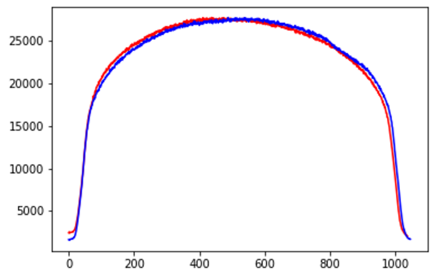
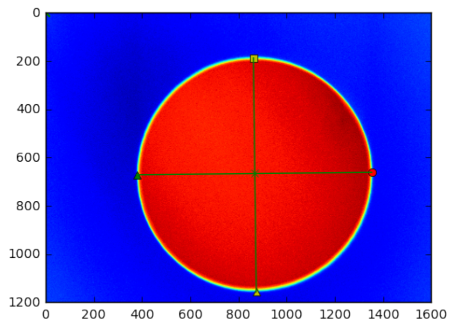

(f) Intensity plot for the R filter

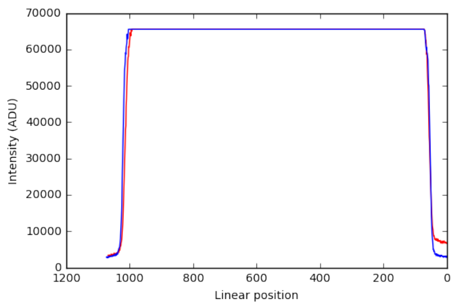
Figure 8: *Left*: Snapshots for different filters (for *old* power supply). The reading direction is aligned with the vertical one. The horizontal and vertical diameters are outlined. *Right*: the intensity profiles along the vertical (red) and horizontal (blue) diameter.

the center and minimum at the edge), we can relate these two quantities. Additionally, referring to the central intensity $I_\lambda\,(0,1)$ and restricting ourselves to a second order polynomial fit, we can write

$$\frac{I_\lambda(0,\mu)}{I_\lambda(0,1)} = a_0 + a_1\mu + 2a_2\mu^2, \quad \frac{S_\lambda(\tau_\lambda)}{I_\lambda(0,1)} = a_0 + a_1\tau + a_2\tau_\lambda^2\,, \tag{33}$$

In order to avoid non-physical behaviors, we limit ourselves with the second


(a) Snapshot for I filter



(b) Intensity plot for the I filter

Figure 9: The snapshot (*left*) and the intensity profile (*right*) of the averaged Sun image for the I filter. Clearly, the dataset is saturated at the sun center, which will give us the wrong estimates for the temperature.

order expansion (as it was done in [1]). We are interested in relative intensities $I_\lambda(0,\mu)/I_\lambda(0,1)$ and the least square polynomial fit allows us to find the fit coefficients (a_0, a_1 and a_2).

We plot the relative intensities $I_\lambda(0,\mu)/I_\lambda(0,1)$ in the figure (10) referring with the different symbols to the various filters' measures. For the fit, we reject the data relative to the four most internal regions (nearest to the center) because they are more affected by the blooming effect (called ghost effect by [1]), as explained before. The determined coefficients are listed in tables 5 and (6).

Band	a_0	a_1	a_2	R^2 test
B	0.58 $\pm$ 0.09	0.50 $\pm$ 0.33	-0.04 $\pm$ 0.14	0.98
V	-0.04 $\pm$ 0.07	1.58 $\pm$ 0.26	-0.27 $\pm$ 0.11	1.00
R	0.29 $\pm$ 0.07	0.98 $\pm$ 0.24	-0.15 $\pm$ 0.10	1.00
I	-1,999 $\pm$ 0,471	8,558 $\pm$ 1,706	-5,951 $\pm$ 1,424	0, 70

Table 5: Second order fit coefficients and R-square test for *old* power supply

Band	a_0	a_1	a_2	R^2 test
B	0.86 $\pm$ 0.16	0.31 $\pm$ 0.6	-0.16 $\pm$ 0.5	0.99
V	-0.16 $\pm$ 0.03	2.07 $\pm$ 0.11	-0.51 $\pm$ 0.04	1.00
R	0.48 $\pm$ 0.06	0.44 $\pm$ 0.21	0.07 $\pm$ 0.09	1.00

Table 6: Second order fit coefficients and R-square test *new* power supply

In order to see how good the polynomial fit describes the dependence of solar intensity from the distance parameter we performed the R-square test (which indicates how the model explains the variability of the response data around its mean). In our case, the model is our polynomial fit. If R-square test is 0 then the model does not explain the variability of the data response, in contrast if R-square is 1, then all the variability is explained. In the table 5 we also showed R-square test for each polynomial fit. It is seen, that only one polynomial fit (for I-band filter) does not fit the data as well (the same applied for the data obtained with the *new* power supply). The reason is because the dataset we are using for I-band is saturated at the sun center, which will also give us wrong estimates for the temperature (see figure (9)). Therefore, we excluded this filter from the analysis.

6.3 Physical central intensity

In the whole discussion we refer to relative intensities. In particular all the intensity values we treat are expressed in digital units and we usually normalize all the values relatively to the center of the disk. But, to obtain the source function we need physical central intensity $I_\lambda(0,1)$ expressed in $Wm^{-3}sr^{-1}$ units. Following the reasoning of [1], we could not measure them, so we can use a set of tabulated intensities determined for some specific wavelengths (see table 7).

Band	Wavelength (nm)	$I_\lambda(0,1)$ $(Wm^{-3}sr^{-1})$
B	420	$(4,5 \pm 0,6) \times 10^{13}$
V	547	$(3,6 \pm 0,2) \times 10^{13}$
R	648	$(2,8 \pm 0,3) \times 10^{13}$
I	871	$(1,6 \pm 0,5) \times 10^{13}$

Table 7: Interpolated physical central intensities $I_\lambda(0,1)$ for different wavelengths λ using the tabulated values from [3], see also [1]

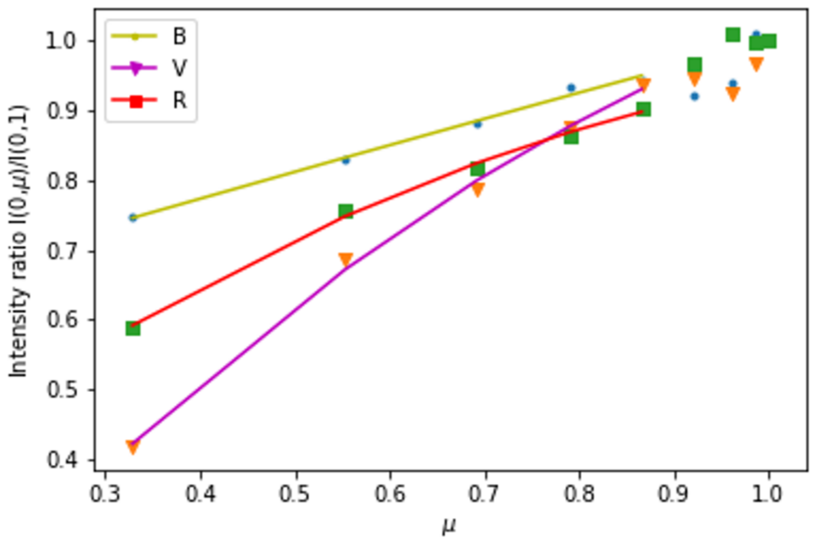


Figure 10: Relative intensities profiles for the used filters. The solid lines correspond to the polynomial fits (excluding the four central regions of the disk).

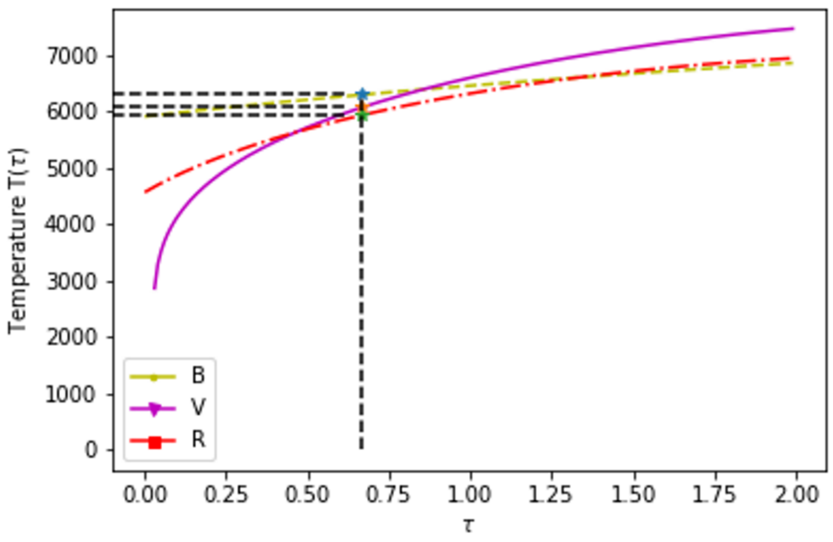


Figure 11: Temperature, $T(\tau_\lambda)$, for the considered filters. The effective temperature ($T(\tau = 2/3)$) for each filter is signed by dotted lines.

6.4 Effective temperature

Till now, we described the relative intensity and the source function using a second order fit. It is exactly the right time to apply the Eddington approximation, which is a set of assumptions for radiative transfer (which was described in theoretical part). In particular, that atmosphere is gray[4], is in thermal radiative equilibrium and that geometry of the atmosphere is plane-parallel. Together with an additional hypothesis that the intensity is a linear function of μ. Applying all of these we can directly equalize the source function and Blackbody Planck function of the temperature at the same optical depth. So, the effective temperature T_{eff} is equal to the real temperature $T(\tau)$ for $\tau = 2/3$. .

In figure (11) we show the temperature as a function of the optical depth for each filter. To obtain these profiles, we used the source function calculated through the parameters of the polynomial fits and the interpolated central intensities putting then into equation (6).

Then we extrapolated the temperatures for $\tau = 2/3$ for each filter and found a values for the effective temperatures: we list them in the table 8 [5]. We applied the uncertainty propagation theory for calculation of the associated errors on the wavelength errors $\delta\lambda = FWHM/2$ (the FWHM of the filter transparency profile), the central intensities δI_λ and the fit coefficients δa_i ($i = 0, 1, 2$).

In the end, we used a weighted mean for B,V and R (excluding I) band

[4]Based on the simplification that the absorption coefficient of matter and optical depth within the atmosphere is constant for all frequencies of incident radiation. Which means that being in gray atmosphere approximation we can eliminate the dependence on λ

[5]see next section (6.5) for details on the power supplies (adapters).

Band	T_{eff} K (New adapter)	T_{eff} K (Old adapter)
B	6044.66 ± 34.01	6298.42 ± 74.26
V	6218.47 ± 90.12	6071.16 ± 90.13
R	5769.23 ± 157.01	5934.25 ± 157.00

Table 8: Effective temperature for the filters we used for two different power supplies (adapters). The voltages of the new and old adapters are $V_{new} = 20,07V$ and $V_{old} = 12,57V$, respectively, for more details see the next section

filters as

$$T_{eff} = \frac{\sum_i T_{eff}^{(i)} \sigma_i^{-2}}{\sum_i \sigma_i^{-2}}, \tag{34}$$

together with associated uncertainties σ_i, supposing them being small, i.e., neglecting the higher orders of smallness (for example, $\sigma_i \sigma_j \ll \sigma_i$)

$$\sigma_i = T_{eff} \left[\sum_i \left(\frac{\sigma_i}{T_{eff}^{(i)}} \right)^2 \right]^{-1/2}, \tag{35}$$

to obtain the final effective temperature

$$T_{eff} = T\left(\tau = \frac{2}{3}\right). \tag{36}$$

and the averaged values with the associated errors for the new and the old adapters, respectively, are:

$$T_{eff} = (6055 \pm 189)\,K \quad \text{and} \quad T_{eff} = (6175 \pm 201)\,K\,, \tag{37}$$

Although, this result is higher than obtained by [1] and [2]

$$T_{eff} = (5996 \pm 116)\,K \quad \text{and} \quad T_{eff} = (6066 \pm 142)\,K\,, \tag{38}$$

respectively, it is comparable and the averaged value from the new adapter lies between both of them. One of the main reasons we got the exceeding value is that we excluded the data from one filter due to saturation discussed above, and another reason is because of the *threshold* we have chosen (this specific feature of the program is discussed in subsection (7.2)).

6.5 Blooming effect VS. Dark current

While the detailed description of CCD working principle was described in section (3), here we limit ourselves to a very basic understanding of this process. The group of photons is coming to the sensor, transformed into the electric signal (by photoelectric effect) then is read out to be recorded; after the next group of photons arrives and the process is repeated. If CCD collects more photos than it can read out per cycle, there is a "leftover" of photons, i.e., as extra light, which in our case was observed. As the result CCD is saturated and the blooming appears.

The results of our observations showed there is a biased increase of intensity, which does not depend on the position of the camera, but imprinted in the CCD itself (see [1] for more details). It is naturally to suppose, this a clear sign of the blooming effect and it was our initial assumption. In this work we answered the question whether it is blooming effect or not, and propose our hypothesis for the observed effect.

Simply speaking, higher power supply to CCD will increase its efficiency, i.e., read out ability and will decrease this annoying effect. In order to check this idea, we acted directly on the CCD by using two different power supplies, say Old and New ones ($12,57V$, $11,21A$; and $20,07V$, $5,37A$ of voltage and current, respectively) to see the difference in the outcome (see figures (12)).

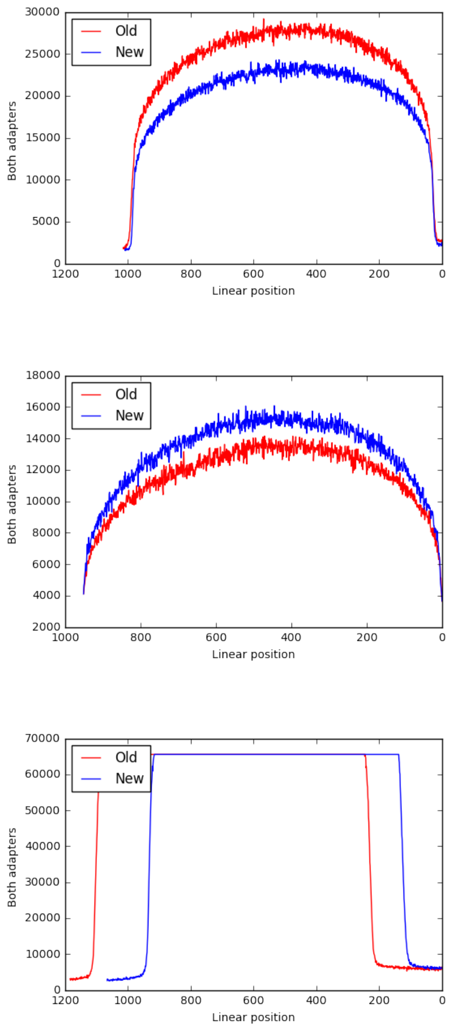


Figure 12: Observed solar center-to-limb intensities for two different adapters.

The results confirmed this hypothesis (except of the V-band filter) and are summarized in the table (9). Here, the ratio I_{new}/I_{old} is between the central intensities obtained with the old and the new adapters, respectively. It represents how many times CCD camera works more efficiently than before: Ratio $\propto$ Efficiency, while the ratio of the voltages of these two adapters is $V_{new}/V_{old} = 1,87$

Band	B	V	R
New	5034	14970	22933
Old	5113	13520	27694
Ratio	1,02	0,9	1,21

Table 9: Blooming experiment results. We show the central intensities for each filter for two power supplies (Old having $V_{old} = 12,57V$, and New having $V_{new} = 20,07V$) and their ration (I_{new}/I_{old})

Besides, the discarded saturated data taken with the I-band filter, only V-band filter showed lowering in efficiency of the CCD sensitivity, which may be explained due to the dark current (see section (3)), thermal noise rise because of the Joule–Lenz law: $P = IV \propto Q$. Where P is the power, V is the voltage, I is the current and Q is Joule–Lenz heat of the electric circuit (CCD sensor in our case).

Also, we supposed the ratios between voltages and intensities being inversely proportional to each other, whereas the exact relation is beyond the scope of our work. It will be interesting to investigate the dependence of the blooming effect (as well as dark current) on the applied power supply in future.

7 Program: GUI application

The main aim of this work was the determination of the Sun effective temperature and for doing this, we made observations and performed data analysis. However, the same steps were already undertaken by our predecessors, that is why in the current work we concentrated mostly on the software improvement and created a self-consistent GUI application which will help to easily solve similar tasks in future. In this section we discuss this in more detail.

7.1 Description of used libraries

To write the GUI application we were using python version 2.7 with the implementations of wxWidget Library . Below you can find some general information on them:

1. wxWidgets (see [7]) gives you a single, easy-to-use API for writing GUI applications on multiple platforms that still utilize the native platform's controls and utilities. Link with the appropriate library for your platform and compiler, and your application will adopt the look and feel appropriate to that platform. On top of great GUI functionality, wxWidgets gives you: online help, network programming, streams, clipboard and drag and drop, multithreading, image loading and saving in a variety of popular formats, HTML viewing and printing, and much more.

   Although wxWidgets is written in C++, you can use it with a variety of languages including Python, Perl, and CSharp. If using wxWidgets with

C++, you will link your code to a different version of the library on each platform. Since the wxWidgets libraries are built and compiled in C++ rather than a language like Java, they are high-performance and nearly as fast as using the native toolkits themselves.

One of the main advantages of wxWidget is its compatibility with various platforms, such as:

- wxGTK: The recommended port for Linux and other Unix variants, using GTK+ version 2.6 or higher.
- wxMSW: The port for 32-bit and 64-bit Windows variants including Windows XP, Vista, 7, 8 and 10.
- wxMac: For delivering Carbon applications on Mac OS X 10.2 through 10.6.
- wxOSX/Carbon: For delivering 32-bit Carbon-based applications on Mac OS X 10.5 and above.
- wxOSX/Cocoa: For delivering 32-bit and 64-bit Cocoa-based applications on Mac OS X 10.5 and above.
- wxX11: A port for Linux and Unix variants targetting X11 displays using a generic widget set.
- wxMotif: A port for Linux and Unix variants using OpenMotif or Lesstif widget sets.,

2. SciPy (see [8]) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages: Numpy, SciPy library, Matplotlib, IPython, Sympy and Pandas

3. NumPy (see [9]) is the fundamental package for scientific computing with Python. It contains among other things:

   - a powerful N-dimensional array object
   - sophisticated (broadcasting) functions
   - tools for integrating C/C++ and Fortran code
   - useful linear algebra, Fourier transform, and random number capabilities

     Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

   NumPy is licensed under the BSD license, enabling reuse with few restrictions.

4. Matplotlib (see [10]) is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits.

5. AUI (see [11]) is an Advanced User Interface library that aims to implement "cutting-edge" interface usability and design features so developers can quickly and easily create beautiful and usable application interfaces.

   AUI attempts to encapsulate the following aspects of the user interface:

   - **Frame Management** provides the means to open, move and hide common controls that are needed to interact with the document, and allow these configurations to be saved into different perspectives and loaded at a later time.
   - **Toolbars** are a specialized subset of the frame management system and should behave similarly to other docked components. However, they also require additional functionality, such as "spring-loaded" rebar support, "chevron" buttons and end-user customizability.
   - **Modeless Controls** expose a tool palette or set of options that float above the application content while allowing it to be accessed. Usually accessed by the toolbar, these controls disappear when an option is selected, but may also be "torn off" the toolbar into a floating frame of their own.
   - **Look and Feel** encompasses the way controls are drawn, both when shown statically as well as when they are being moved. This aspect of user interface design incorporates "special effects" such as transparent window dragging as well as frame animation.

7.2 GUI application

The initial idea of the application was to organize it in the most usual, but flexible way. Thus, we adopted some elements from the Google Chrome, namely the notebook structure. By referring to the figure (13) and (14) we will discuss the structure in more detail.

wxFrame contains **File Menu**, **AUI Notebook** which invokes **class TabPanel** (see code) on which two different panels instantiated, namely **DataPanel** (Panel 1) and **ResultPanel** (Panel 2). Let us discuss this in more details.

1. **Menu** contains the following buttons

   - **Add** is the button which will create a new tab;
   - **Combine** is the button which will combine the resulting data from several tabs in one window;
   - **About** contains the small description of the program together with the updates done;
   - **Quit** is the button for closing the program;

2. **TabPanel** class contain two panels to show the images and other plots

3. **Panel 1** is the scrollable panel, which contains the array of canvases (see figure (14)) organized in the grid

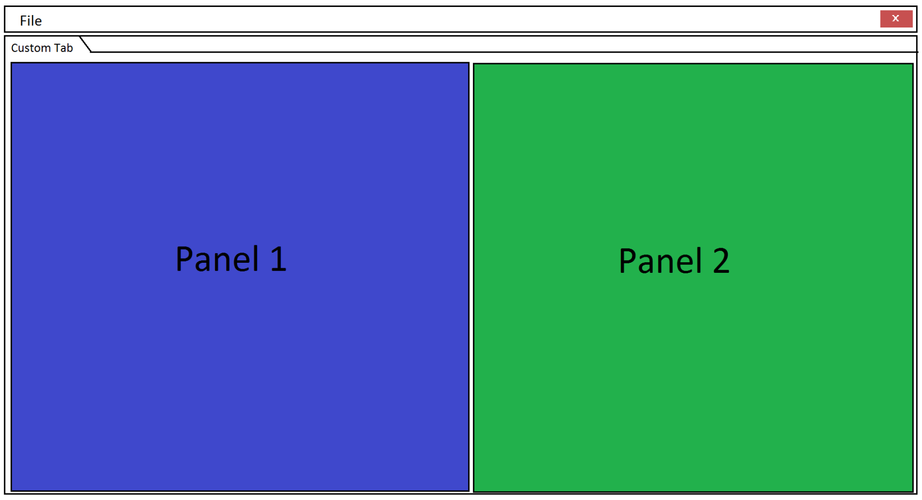


Figure 13: The basic structure of the GUI application (background).

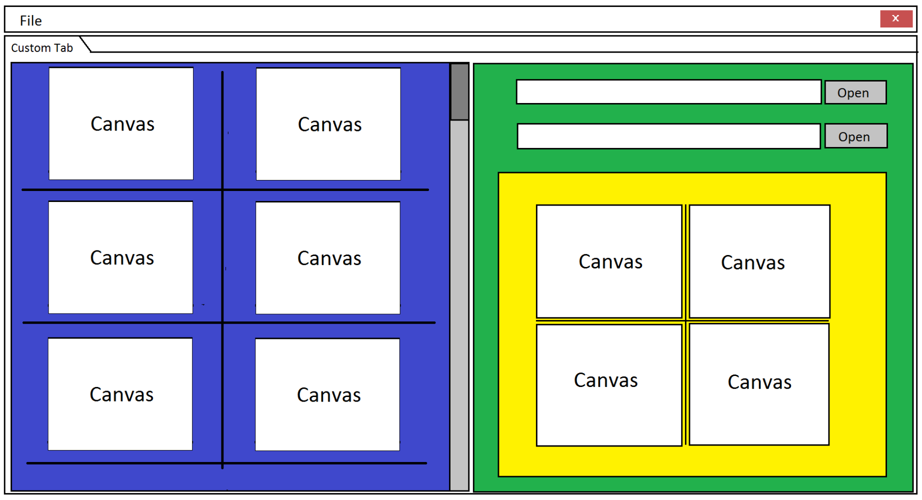
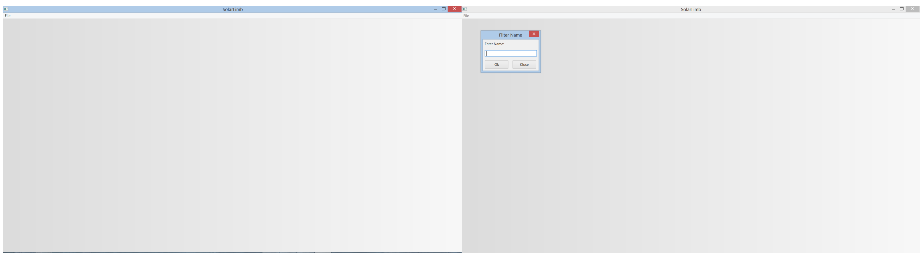


Figure 14: The basic structure of the GUI application.

4. **Panel 2** contains several buttons to browse files, the array of canvases to show resulting images and also **Save** button to save the results (see figure (14))

The application is very easy to handle and it works as follows:

1. Start the application by executing an appropriate file on you computer. The initial instance of the application is the blank window (figure (15a));
2. Go to the file menu and press **Add** button. The dialog frame will be created asking you to choose the name of your tab (figure (15b)). Note: the value will be stored and eventually shown in the table with resulting data;



(a) (b)

Figure 15: Figure (15a) shows the first instance of the application; figure (15a) shows the dialog window which asks to enter the name for the first tab (to be created)

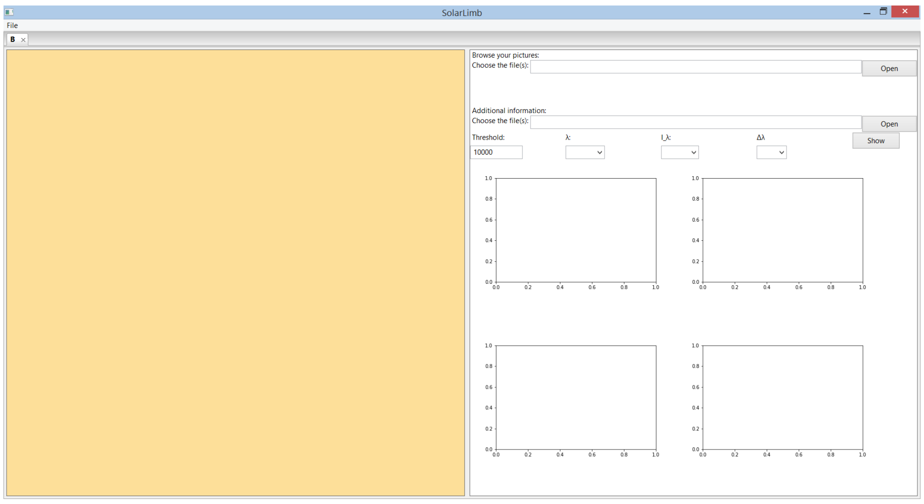
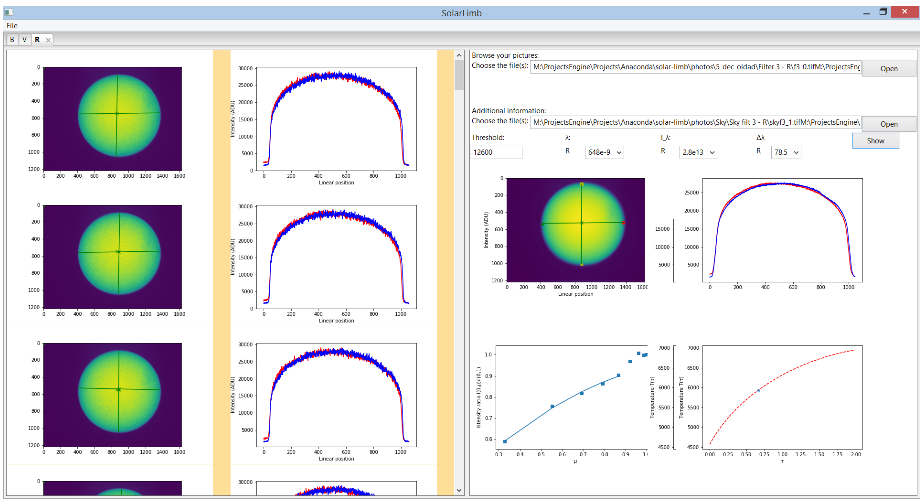


Figure 16: Example of the newly created tab with the name *B*, which stands for filter B in our case.

3. After the tab was created (figure (16)), user should choose the sun images (first **Open** button) and the ***same*** amount of sky images (second **Open** button) for the particular filter. Also the appropriate values for the threshold, λ, I_λ and $\delta\lambda$ parameters should be indicated. Note: we used the threshold equal to 4000 for the B filter, 2750 for V filter and 8000 for R filter;

4. To start the processing, user should press **Show** button. It takes some time to process the images (for instance, for 23 images of R filter it took us **4 minutes and 47 seconds** (memory consumption was 631.3 MB); it can be written that program does not responding, but it is not true). The program is designed in such way that it will draw you the exact amount of figures you have chosen. Eventually, user should see on the left all chosen sun images, with the found center, as well as the intensity profiles for each particular image; on the right, the resulting sun image will be available, its intensity profile, the intensity and temperature curves (see figure (17))

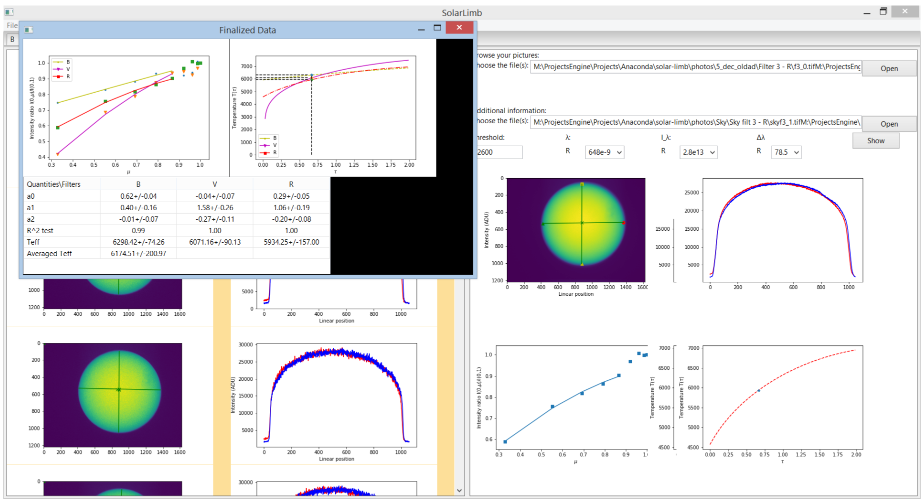


Browse your pictures:
Choose the file(s): M:\ProjectsEngine\Projects\Anaconda\solar-limb\photos\5_dec_oldad\Filter 3 - R\f3_0.tifM:\ProjectsEng

Additional information:
Choose the file(s): M:\ProjectsEngine\Projects\Anaconda\solar-limb\photos\Sky\Sky filt 3 - R\skyf3_1.tifM:\ProjectsEngine\

Threshold:	λ:	I_λ:	Δλ
12600	R 648e-9	R 2.8e13	R 78.5

Figure 17: Example of data processing.



Quantities\Filters	B	V	R
a0	0.62+/-0.04	-0.04+/-0.07	0.29+/-0.05
a1	0.40+/-0.16	1.58+/-0.26	1.06+/-0.19
a2	-0.01+/-0.07	-0.27+/-0.11	-0.20+/-0.08
R^2 test	0.99	1.00	1.00
Teff	6298.42+/-74.26	6071.16+/-90.13	5934.25+/-157.00
Averaged Teff	6174.51+/-200.97		

Figure 18: The frame with resulting data.

respectively;

5. The above process should be repeated for any particular filter. When all the data is calculated and the plots are shown, the user should go again to the **File** menu and press **Combine** button. It will open the new frame with the resulting curves and the table of the resulting data as well as the value for the average effective temperature (see figure (18)). Note: the combine button should be used after at least one tab was processed and the result obtained, otherwise it will give you an error;

Now, a few words about the code itself. One of the most important methods is the **method *ShowImages*** from the **class *Calculator***. It calculates almost everything in the program and it works in the following way:

- When the program starts, it receives the values of the chosen sun images as well as values for the sky images (to reduce the fixed pattern noise, described in subsection (3.1.3), we needed to obtain the *flat field images* of the sky for each particular filter), the number of the chosen images, the value for threshold and tabulated values of I_λ, λ and $\delta\lambda$.

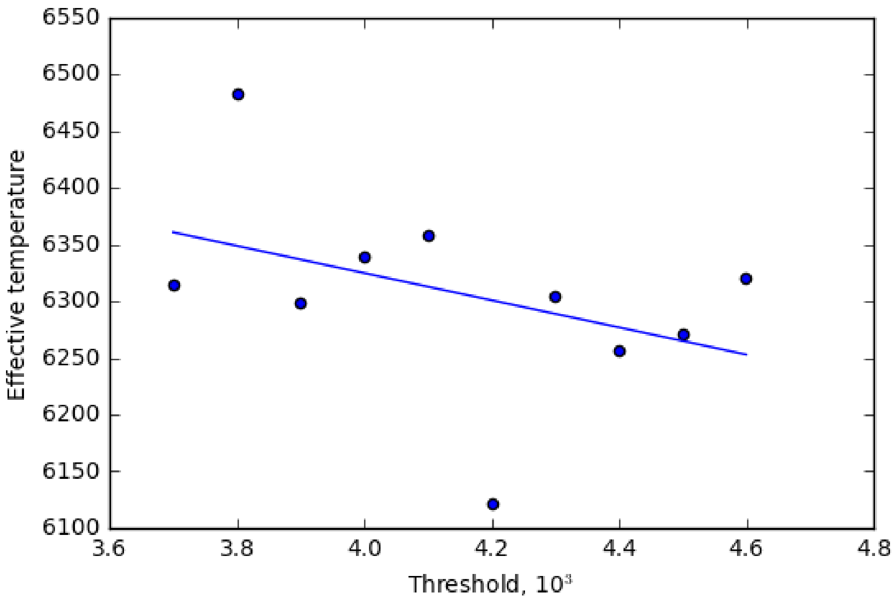


Figure 19: The dependence

- Each image is the array of numbers, which is opened and reshaped to be of the form of (1219,1619). This is important as we want to find the solar center for each image. The simplest algorithm implies finding the four (two vertical and two horizontal) points of the sun and then find their intersection. To do this we need firstly separate the sky and the sun on each picture, i.e. find the border. This is there parameter ***threshold*** comes in hand, because it helps us to check the value of intensity (not real one). The idea is rather simple - if the current value of the reshaped array is less than the threshold, then it is not the point of the sun. This parameter is particularly important, because it will give us different values for the effective temperature depending on the value of the threshold (see tables (10), (11) and (12)). We plotted (see figures (19), (20) and (21)) these values with the second order polynomial and found the variables which will give us the minimal possible value and at the same time will also correlate with others. These values are: 3900, 2750 and 12600 for filters B, V and R respectively. The same values for threshold we used to calculate the effective temperature of the Sun for the *new* adapter (see subsection (6.4)). Note: while it may seem that there are better values for the threshold and thus the effective temperature, we used these, because, first, they do not cut the final intensity curves for filters and, second, they will give us physically reasonable final intensity curves, whereas, for

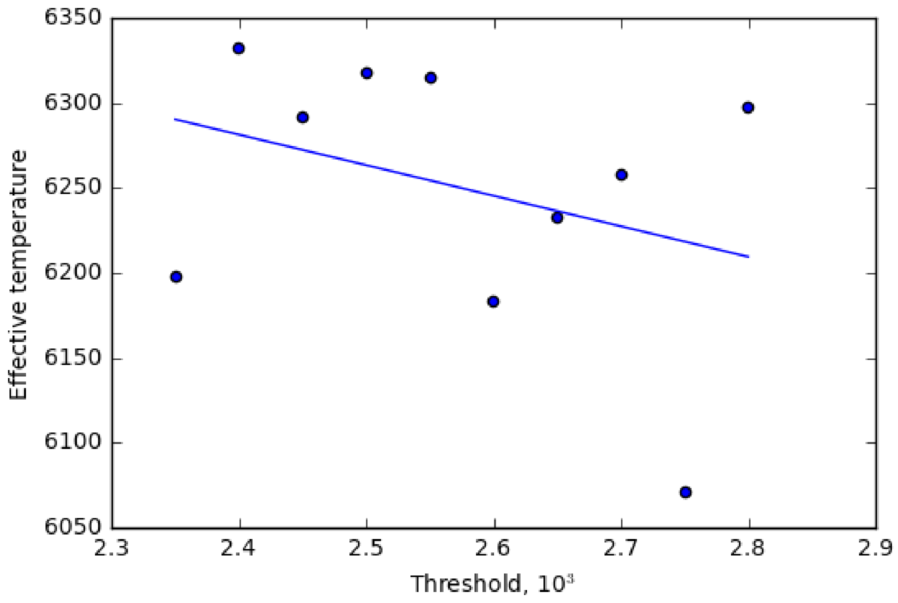


Figure 20: The frame with resulting data.

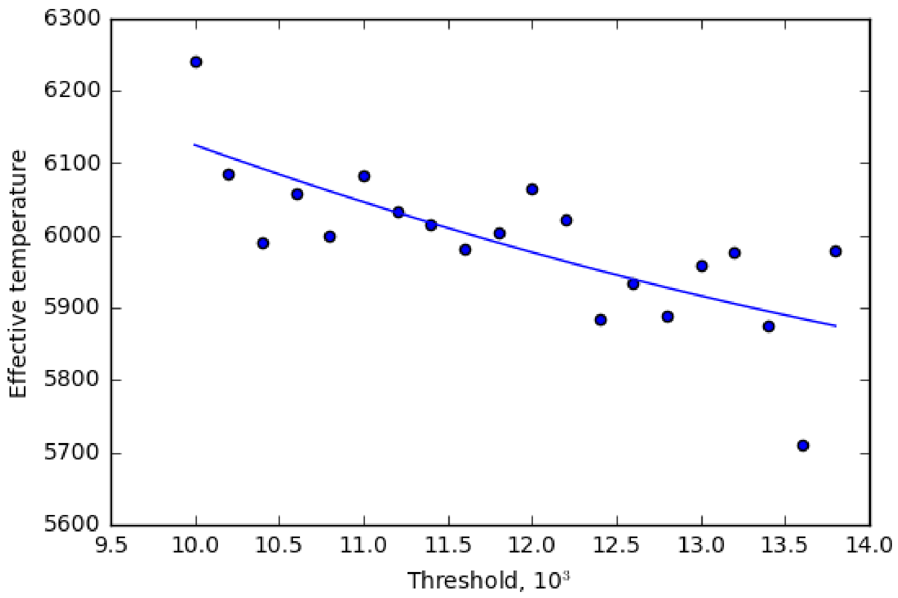


Figure 21: The frame with resulting data.

example, if we choose 4500 as the value for the threshold we can get the badly shaped resulting intensity curve (see figure (22)). All this provides us with more reasonable estimates.

- After we checked an image and found the four points and the intersection

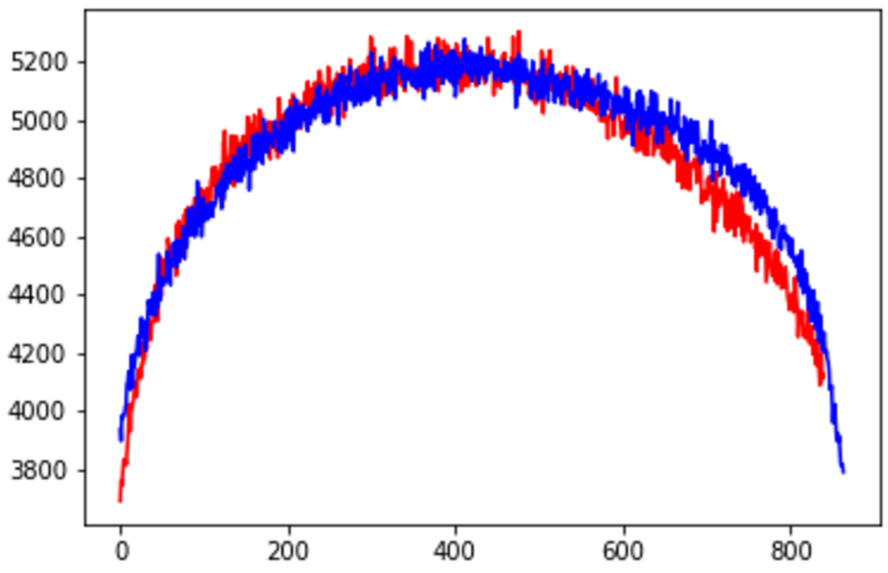


Figure 22: Example of badly shaped resulting intensity curve for the data obtained with filter B.

Threshold	3.7×10^3	3.8×10^3	3.9×10^3	4×10^3	4.1×10^3
Temperature	6315 ± 73	6483 ± 73	6298 ± 74	6340 ± 73	6358 ± 73
Threshold	4.2×10^3	4.3×10^3	4.4×10^3	4.5×10^3	4.6×10^3
Temperature	6121 ± 73	6305 ± 73	6257 ± 257	6272 ± 73	6321 ± 140

Table 10: The table with the averaged effective temperature for filter B. We used this values to identify the optimal value for the threshold we need to use to obtain the lowest (but physically justified) averaged value for the effective temperature for this band.

between them, we are plotting the each image of the sun and its intensity profile (for which we subtract the sky value). The same procedure holds for every chosen image for this particular filter and it goes until the number of operations will become equal to number of the images initially chosen;

- Almost the same procedure holds to identify the averaged picture of the sun, with the small difference, i.e. we fix first picture in the row (more precisely its center) and shift every other picture's center into the same position. Here we also calculate the center of the resulting picture as well as other parameters such as coefficients a_0, a_1, a_2, the R^2 test and also the intensity and the effective temperature of the sun for each particular filter. We plot the resulting curves (approximated by polynomials, as described in subsection (2.2)), which can be seen at the bottom of the right panel. Together with this, the program stores the calculated values in the array under the index of the current page of the notebook;
- These values then invoked in the **class** ***CombinedInfoPanel*** to build

Threshold	2.35×10^3	2.4×10^3	2.45×10^3	2.5×10^3	2.55×10^3
Temperature	6198 ± 90	6332 ± 90	6292 ± 90	6318 ± 90	6315 ± 90
Threshold	2.6×10^3	2.65×10^3	2.7×10^3	2.75×10^3	2.8×10^3
Temperature	6183 ± 90	6233 ± 90	6258 ± 90	6071 ± 90	6298 ± 90

Table 11: The table with the averaged effective temperature for filter V. We used this values to identify the optimal value for the threshold we need to use to obtain the lowest (but physically justified) averaged value for the effective temperature for this band.

Threshold	10×10^3	10.2×10^3	10.4×10^3	10.6×10^3	10.8×10^3
Temperature	6241 ± 157	6086 ± 157	5991 ± 157	6059 ± 157	5999 ± 157
Threshold	11×10^3	11.2×10^3	11.4×10^3	11.6×10^3	11.8×10^3
Temperature	6082 ± 157	6033 ± 157	6016 ± 157	5982 ± 157	6004 ± 157
Threshold	12×10^3	12.2×10^3	12.4×10^3	12.6×10^3	12.8×10^3
Temperature	6066 ± 157	6023 ± 157	5885 ± 157	5934 ± 157	5888 ± 157
Threshold	13×10^3	13.2×10^3	13.4×10^3	13.6×10^3	13.8×10^3
Temperature	5959 ± 157	5977 ± 157	5876 ± 157	5710 ± 157	5979 ± 157

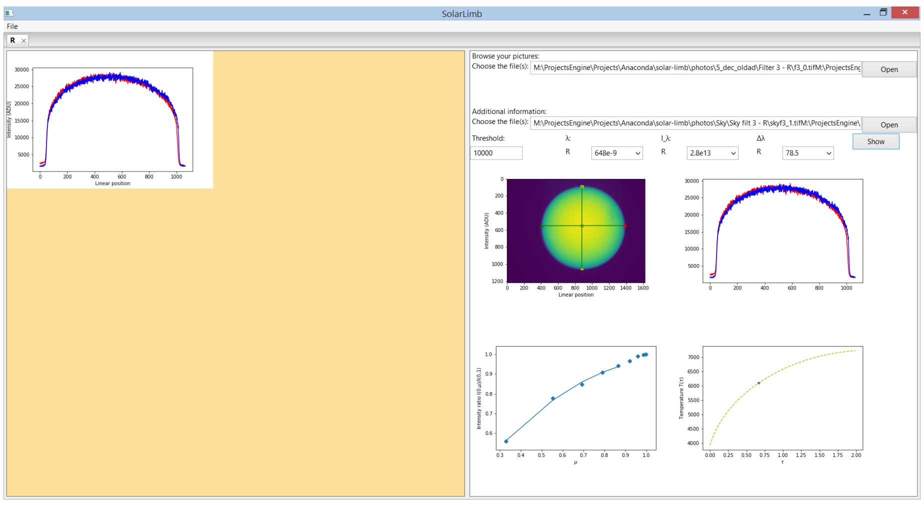
Table 12: The table with the averaged effective temperature for filter R. We used this values to identify the optimal value for the threshold we need to use to obtain the lowest (but physically justified) averaged value for the effective temperature for this band.

the resulting curves. The resulting values are shown in the table, where each column has the same name as the pages of the notebook.

8 Suggestions for future improvements

Here we briefly give some suggestions for eliminating some defects of the code and also future improvements to the program making it more user friendly (for the improvements of the observational part please refer to conclusion section in [1]).

- As the program designed in such a way that it will draw you the exact amount of figures you have chosen, there is a small defect associated with it: when it first draws the images, it does not show you the array, but only the last plot (i.e. intensity curve for the last image opened; see figure (23)). The problem is easily solved when user expands/shrinks the window. Thus, the possible way to solve this problem is to instantiate the methods *Update()* and/or *Refresh()*;
- Due to the huge amount of data contained in each image, the memory consumption is also quite big (631.3 MB for processing 23 images of R filter). Therefore, it is advisable to divide the process into two parts - the one which will process data on the left and the one which process it on the right - by implementing several buttons to save and retrieve the processed information;



Browse your pictures:
Choose the file(s): M:\ProjectsEngine\Projects\Anaconda\solar-limb\photos\5_dec_oldad\Filter 3 - R\f3_0.tifM:\ProjectsEng

Additional information:
Choose the file(s): M:\ProjectsEngine\Projects\Anaconda\solar-limb\photos\Sky\Sky filt 3 - R\skyf3_1.tifM:\ProjectsEngine\

Threshold: 10000
λ: R 648e-9
I_λ: R 2.8e13
Δλ R 78.5

Figure 23: Example of the defect encountered after the data processing.

- Changing the **Additional information** part - making it more interactive by creating a hidden panel, which will be available if the user clicks on BitMap button, for example. This part should at contain the Threshold part (manually adding the threshold; it has the value of 10000 by default) and the **Open** button (for browsing sky images);
- Add the method which will enable user to choose between fitting to polynomials of higher orders
- Implementing **FloatCanvas** instead of **Canvas**: Basically, you have a canvas that you can put lines, polys, rects, circles, bitmaps, and text on. The canvas takes care of redraws and sizing for you. It comes with a toolbar for zooming and panning. It has a full set of mouse events and a callback mechanism for reacting to users clicking, etc. on the objects (see [12] for more information);
- **About** button: Separate the description of the program and the changes made (updates) in two panels, both of which will be contained in one frame.
- Instantiate the method, which will allow user save any particular resulting image by double clicking on it (or from the menu - right click $\rightarrow$ save)

9 Conclusions

The goal of our work was to determine Sun effective temperature by improving the work, which was initially done by [2] (using the same experimental setup), and later upgraded by [1]. The first idea about the "dragging effect" due to transfer inefficiency (simply speaking, read out ability) of the CCD, proposed by [2], was checked by [1] by performing a calibration of the CCD together with the usage of the slit method in order to improve its work.

After setting up the instrumentation, we measured the focal length of the achromatic doublet and got $f = 45.5 \pm 0.04$cm using the focuser, designed by [1] (who obtained $f = 44.7 \pm 0.3$cm). Then, we performed the solar observations through the UBVRI wavelength bands.

The analysis of the images showed similar anomalous effect, "blooming" effect (called "ghost" effect by [1]). They concluded that this effect, while being an artifact of the camera, is appearing in the absence of the mechanical shutter (leading to the increase of the light, which marks the CCD sensor during the reading process). In other words, having an outer origin. Although we propose another reason for this behavior, thinking about deviance (blooming) as the feature imprinted into the sensor itself, i.e. having an inner origin. So, we considered the perspective possibility for improvement of the faint charge transfer inefficiency, by acting directly on the CCD sensor in order to test both hypotheses simultaneously. For this reason, we conducted all the observations twice using different power supplies with voltages 12.57V and 20.07V (for convenience, we called them "old" and "new" adapters, respectively). Analyzed data partially confirmed both ideas.

Proceeding with the analysis, we found the orders of sensitivity improvement. As was expected using higher power supply increases the sensitivity of the CCD sensor, giving us more precise result and decreasing the blooming effect. However, R and B band filters gave us visible improvement 21% and 2% times (even being smaller for the second one, but still positive), the V-band filter gave the opposite result - decrease in efficiency about 2%. Hence, the first two filter bands play in favor to our hypothesis, while the third one showed that outer effect plays bigger role than inner one. We explained it as dark current (arising from Joule–Lenz heating) instead of the ghost effect (proposed by [1]). Here, the natural questions arise, which role of the outer effects (noises) is more significant? And how to minimize them? The answers for these questions are beyond the scope of this work.

To obtain the averaged values of the temperatures for each adapter, we developed a self-consistent GUI application which will enable solving similar tasks in future much faster and easier. The application adopts data analysis approach (also described in [1]) in our realization. With its help we obtained following values for the effective temperature for new and old adapters:

$$T_{eff}^{(new)} = (6055 \pm 189)\,K\,, \quad T_{eff}^{(old)} = (6175 \pm 201)\,K\,, \tag{39}$$

and showed that effective temperature approaches its typical, nominal value (being for the *old* and *new* adapters to be only 5% and 7% higher, respectively) with the increase of the CCD camera sensitivity. Obtained values suggest the conclusion of [1] about the CCD malfunction is wrong, as there is visible improvement in effective temperature value.

The obtained values of effective temperatures in works of [2] and [1] were $T_{eff} = (6066 \pm 142)$K and $T_{eff} = (5996 \pm 116)$K (which are 5% and 4% higher than typical value $T_{eff} = 5780$K), respectively. As can be easily seen, our values now better approach nominal value, being also compatible with other works.

Acknowledgement

We would like to thank Dr. Luca Giovanelli for his precious help and useful advices to improve our work and also Dr. Giorgio Viavattene for teaching us how to operate and mount the telescope.

References

[1] Elia Chiaraluce, Francesco Zuliani, "Solar center-to-limb darkening"

[2] L. Di Mascolo, C. Di Fronzo (2014).

[3] Erika Böhm-Vitense, "Introduction to stellar astrophysics: Volume 2", Cambridge, Cambridge University Press 1989.

[4] G. Rybicki, A. Lightman, "Radiative Processes in Astrophysics", 2004

[5] K. N. Liou, "An Introduction to Atmospheric Radiation, Volume 84", Academic Press 2002

[6] J. Janesick, "Scientific Charge-Coupled Devices", Spie Press Monographs, Pm83 (society of PhotoOptical), 2001

[7] wxWidget Official web-page: https://www.wxwidgets.org/about/

[8] Scipy Official web-page: https://www.scipy.org/

[9] NumPy Official web-page: http://www.numpy.org/

[10] Matplotlib Official web-page: https://matplotlib.org/

[11] AUI Official web-page: https://wxpython.org/Phoenix/docs/html/wx.lib.agw.aui.html

[12] Official web-page: https://wiki.wxpython.org/FloatCanvas

Appendices

A Code

labelsec:code
The latest version can be found at: https://github.com/hroney/solar-limb

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jan 09 14:15:16 2018

@author: Krad
"""
# to prevent error of division by 0
from __future__ import division
from __future__ import unicode_literals
#from __future__ import print_function

import os, os.path
import matplotlib
from scipy.interpolate import *
from pylab import *
import matplotlib.lines as mlines
# graphical interface libraries
import wx
# for making a scrollable panel
import wx.lib.scrolledpanel
from wx.lib.scrolledpanel import ScrolledPanel
# for browsing files
import wx.lib.filebrowsebutton
import wx.lib.agw.aui as aui
# for working with gridTables
import wx.grid as gridlib
# regular pubsub import - for transferring variables
from wx.lib.pubsub import pub
# the easiest way to get the full statistical results for the fit is to use sta
import statsmodels.api as sm
# to show borders around sizers and other stuff - making app more interactive
#import wx.lib.inspection
#import wx.lib.mixins.inspection
#wx.lib.inspection.InspectionTool().Show()
#import random
# for Showing, reshaping and calculating data
import matplotlib.pyplot as plt
import numpy as np
from scipy import misc
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigureCanvas
import random
import wx.lib.mixins.listctrl  as  listmix
#from Tkinter import *
#from matplotlib.figure import Figure

################################################################################
''' Class for reshaping and calculation all things '''
class Calculator():

    ''' Constructor of the class '''
    def __init__(self):
        ##############################################
        ## Definitions of values for Our Analysis ##
        ##############################################
        # variable to distinguish between even and odd indecies of gridsizer
        self.number = 0
        # the index of data in the list to find the right one
        self.index = 0

        self.im = []
        self.imresh = []
        # for the intensity curve of each picture
        self.imresh_intensity_curve = []
        # chosing a threshold
        self.threshold = 0
        self.Cx = [0 for col in range(100)]
        self.Cy = [0 for col in range(100)]
        #To find the center of the sun
        self.FirstPoint = []
        self.SecondPoint = []
        self.ThirdPoint = []
        self.FourthPoint = []
        #-----------------------------------------------------------------------#
        I = 1619
        J = 1219
        # For data Analysis part
        self.imreshSum = [[0 for col in range(I)] for row in range(J)]
        self.imSum = [[0 for col in range(I)] for row in range(J)]
        self.SkySum = [[0 for col in range(I)] for row in range(J)]
        self.SkySumFinal = [[0 for col in range(I)] for row in range(J)]
        self.imreshSumFinal = [[0 for col in range(I)] for row in range(J)]
        #-----------------------------------------------------------------------#
        self.rout = [0 for x in range(10)]
        self.rin = [0 for x in range(10)]
        self.r = [0 for x in range(10)]
        self.delr = [0 for x in range(9)]
        self.miuk = [0 for row in range(9)]
        self.delmiu = [0 for row in range(9)]
        self.miukSum = 0
        self.miukFIN = [0 for row in range(9)]
        self.x = 0
        #----------------------------------------------------------------------#
        self.Ik = [0 for row in range(10)]
        self.Il0 = 0   #[0 for row in range(10)]
        self.Il = [0 for row in range(10)]
        self.S = [0 for row in range(9)]
        #self.S = 0
        self.Ilratio = [0 for row in range(10)]
        self.miu = 0
        self.sIk = [0 for row in range(1619)]
        self.IkN = [[0 for col in range(1619)] for row in range(1219)]
        self.IkSum = [[0 for col in range(1619)] for row in range(1219)]
        self.IkFIN = [[0 for col in range(1619)] for row in range(1219)]
        self.IlSum = [[0 for col in range(10)] for row in range(10)]
        self.IlFIN = [[0 for col in range(10)] for row in range(10)]


        ########################
        ########################
        self.Error = [0 for col in range(9)]
        self.ErrorSum = 0
        #########################
        #----------------------------------------------------------------------#
        # The bunch of constants
        #self.lambd = [1, 420e-9, 547e-9, 871e-9, 648e-9]
        #self.Ilambd = [1, 3.6e13, 4.5e13, 1.6e13, 2.8e13]
        #self.deltalambd = [17.5, 45, 16.5, 118, 78.5]

        self.hPl = 6.626e-34
        self.kBolz = 1.38e-23
        self.clight = 3e8
        #What changes per filter:
        #self.Ratio1 = [0 for col in range(7)]
        #self.Ratio2 = [0 for col in range(7)]

        #self.p2Array = [0 for col in range(7)]
        #self.IlArray = [0 for col in range(7)]
        #self.TeffArray = [0 for col in range(7)]
        #self.TtaulArray = [0 for col in range(7)]

        self.Ratio1 = [0 for col in range(100)]
        self.Ratio2 = [0 for col in range(100)]

        self.p2Array = [0 for col in range(100)]
        self.IlArray = [0 for col in range(100)]
        self.TeffArray = [0 for col in range(100)]
        self.TtaulArray = [0 for col in range(100)]


        self.delMiuk = 0
```

```
def ShowImages(self, DataPanel, listLength, listPaths,
               ResultPanel, globalPageIndex):

    print ResultPanel.comboSelection[0]
    print ResultPanel.comboSelection[1]
    print ResultPanel.comboSelection[2]

    # taking the value of threshold
    self.threshold = ResultPanel.threshold
    #stands for filter in Ruslan's interpretation goes through values from
    #important to remember it is not index or y, but anather variabl for ar
    self.f = globalPageIndex# just for testing purposes
    print ('self.f', self.f)
    #-------------------------------------------------------------------#
    #retrieving the values of constants
    #self.lambd[self.f] = ResultPanel.comboSelection[0]
    #self.Ilambd[self.f] = ResultPanel.comboSelection[1]
    #self.deltalambd[self.f] = ResultPanel.comboSelection[2]
    self.lambd = ResultPanel.comboSelection[0]
    self.Ilambd = ResultPanel.comboSelection[1]
    self.deltalambd = ResultPanel.comboSelection[2]
    # variable for making matrix of data
    sizerRow = listLength * 2
    self.index = 0
    #Creating a sizer which will hold all canvases with data
    DataPanel.gridSizerShow = wx.GridSizer(rows = sizerRow, cols = 2, hgap=
    # This loop is for showing obtained images
    for y in xrange(0, sizerRow):
        DataPanel.listShowFigure.append(plt.figure())
        # creating the axes
        DataPanel.listShowAxe.append(DataPanel.listShowFigure[y].add_subplo
        DataPanel.listShowFigureCanvas.append(FigureCanvas(DataPanel, -1, I
        # To put all observed data in the left column we need to distinguis
        # between even and odd numbers
        if self.number % 2 == 0:
            # Drawing the data on the canvas
            DataPanel.listShowFigure[y].set_canvas(DataPanel.listShowFigure
            # Clearing the axes
            DataPanel.listShowAxe[y].clear()
            # reading an image from the same folder
            self.im = misc.imread(listPaths[self.index])
            # rehsping an array (changing the )
            self.imresh = self.im.reshape(1219,1619)
            # reading an image from the same folder
            self.imresh_intensity_curve = misc.imread(listPaths[self.index]
            self.imresh_intensity_curve_reshaped = self.imresh_intensity_cu
            #print ("reshaped image ", self.imresh_intensity_curve_reshaped
            # for drawing the intensity curve (it will evenatually forget
            #the iniial value, so we need to save it somewhere)
```

```
#self.imresh_intensity_curve = self.im

# the reshaping for sky images
self.Sky = misc.imread(DataPanel.listSkyPaths[self.index])
self.Skyresh = (self.Sky.reshape(1219,1619))/1000
####################
##Reshaping images##
####################
''' Making the condition if image less than threshold etc. '''
for j in range(0,1619):
    for i in range(0,1219):
        if (self.imresh[i,j] < self.threshold):
            self.imresh[i,j] = 0
        else: self.imresh[i,j] = 1

##################################
##Finding the center of each Sun##
##################################
# Now, it is time to find the center of the Sun
found = False
for j in range(0,1619):
    if found:
        break
    for i in range(0,1219):
        if (self.imresh[i,j] == 1):
            self.FirstPoint = [j,i]
            found = True
            break


found = False
for j in range(1619-1, -1, -1):
    if found:
        break
    for i in range(0,1219):
        if (self.imresh[i,j] == 1):
            self.SecondPoint = [j,i]
            found = True
            break


found = False
for i in range(0,1219):
    if found:
        break
    for j in range(0,1619):
        if (self.imresh[i,j] == 1):
            self.ThirdPoint = [j,i]
            found = True
            break
```

```
found = False
for i in range(1219-1, -1, -1):
    if found:
        break
    for j in range(0, 1619):
        if (self.imresh[i,j] == 1):
            self.FourthPoint = [j,i]
            found = True
            break

#Now let us mathematically find an intersection of these 2 line
# Center of the sun is C = (Cx, Cy)
self.Cx[self.index] = ((self.FirstPoint[0]*self.SecondPoint[1]-
self.Cy[self.index] = ((self.FirstPoint[0]*self.SecondPoint[1]-
# Let's not shift the center of the next image to the center of
#order to find an average
self.imreshSum = self.imreshSum +
(np.roll(np.roll(self.imresh_intensity_curve_reshaped, (self.Cy
self.Cy[self.index]), axis=None),
(self.Cx[0]-self.Cx[self.index]), axis=0))
self.SkySum = self.SkySum + self.Skyresh
#And plot everything, together with the intersection point:
#------------------------------------------------------------#
# invoking native method for showing images -Sun
DataPanel.listShowAxe[y].imshow(self.imresh_intensity_curve_resh

# Showing the intersection of two lines and the center of each
DataPanel.listShowAxe[y].plot((self.FirstPoint[0], self.SecondP
(self.FirstPoint[1], self.SecondPoint[1]), color = 'g')
DataPanel.listShowAxe[y].plot((self.ThirdPoint[0], self.FourthP
(self.ThirdPoint[1], self.FourthPoint[1]), color = 'g')
DataPanel.listShowAxe[y].plot([self.Cx], [self.Cy], marker='x',
#DataPanel.listAxe[x].imshow(Center[x], cmap="hot")
# draw canvas
DataPanel.listShowFigureCanvas[y].draw()

#################
##Data Analysis##
#################

self.R1 = self.Cx[self.index] - self.FirstPoint[0]
self.R2 = self.SecondPoint[0] - self.Cx[self.index]
self.R = (self.R1+self.R2)/2

self.r1 = self.FirstPoint[0]
self.r2 = self.FirstPoint[0] + (2*self.R/9)
self.x = 0
self.y = (2*self.R/9)
###########   ###########  ##############
```

```
# We should find the Distance parameters $\mu$ and Intensity
#ratios for each k-th region of the sun
# Then conduct the Polynomial fit of their dependance
#=> relating to the place parallel approximation
# we can find the optical depth $\tau$ and then
#effective temperature of the sun (which is our goal)
self.R = float(self.SecondPoint[0]-self.Cx[self.index])
self.rin[0] = float(0)
self.rout[0] = float(self.R/9)

# Here we dicvide our sun for nine regions (concentric rings)
#with the center in the solar center,
# and look for distance parameter and intensity ratio:
for k in range(0,9):
    self.DEV = 0
    self.sIk = 0
    self.r[k] = (self.rin[k]+self.rout[k])/2
    self.delr[k] = (self.rin[k]-self.rout[k])/2

    self.Error[k] = self.delr[k]/self.r[k]

    self.miuk[k] = (np.sqrt(1-(self.r[k]**2/self.R**2))).astype


    self.delmiu[k] = (self.r[k]*self.delr[k])/(np.sqrt(1- (self
    self.IkN = np.asarray([[0 for col in range(1619)] for row i
    #Now we determine the average intensity for each k-th regio

    self.Ik[k] = (self.imresh_intensity_curve_reshaped[self.Cy[
                 (self.Cx[self.index]+int(self.rin[k]))]+self.

    self.dev = ((self.imresh_intensity_curve_reshaped[self.Cy[s
                                                        (self.Cx[
    self.DEV = self.DEV + self.dev

    #Standard deviation sI of Intensity Ii:
    self.sIk = np.sqrt(np.fabs(self.DEV)/(2))

    self.rin[k+1] = self.rin[k] + (self.R/9)
    self.rout[k+1] = self.rout[k] + (self.R/9)

    self.I10 =
    self.imresh_intensity_curve_reshaped[self.Cx[self.index],se
    self.Il[k] = self.Ik[k].astype(float)/self.Ik[0]
    self.ErrorSum += abs(self.Error[k])

self.ErrorI = self.ErrorSum/9

print ("miuk", self.miuk)
print ("Ik ", self.Ik)
```

```
            #Weighted mean overl all the pictures for each filter:
            np.array(self.IlSum)[self.f][k] += np.array(self.Il)[k]
            #####################
            self.IlFIN[self.f][k] = float(self.IlSum[self.f][k])/(self.inde

        else:
            # Drawing the data on the canvas
            DataPanel.listShowFigure[y].set_canvas(DataPanel.listShowFigure
            # Clearing the axes
            DataPanel.listShowAxe[y].clear()
            # Intensities for every picture in cycle:
            DataPanel.listShowAxe[y].set_ylabel('Intensity␣(ADU)')
            DataPanel.listShowAxe[y].set_xlabel('Linear␣position')
            # for each variable we need to account the skies to get reason
            intensity curves
            image_minus_sky = self.imresh_intensity_curve_reshaped - self.S
            print ('image_minus_sky', image_minus_sky)
            #Vertical profile
            DataPanel.listShowAxe[y].plot(image_minus_sky[(self.ThirdPoint[
            (self.FourthPoint[1]+50),self.Cx[self.index]], 'r')
            #Horizontal profile
            DataPanel.listShowAxe[y].plot(image_minus_sky[self.Cy[self.inde
            (self.FirstPoint[0]-50):(self.SecondPoint[0]+50)], 'b')
            # draw canvas
            DataPanel.listShowFigureCanvas[y].draw()
            # keeping this index he same until now to draw nice intensity p
            self.index = self.index + 1

        self.number = self.number + 1

        #adding plots to sizer
        DataPanel.gridSizerShow.Add(DataPanel.listShowFigureCanvas[y], 0, w


# Setting a sizer to a panel
DataPanel.SetSizer(DataPanel.gridSizerShow)
#DataPanel.Update()

self.imreshSumFinal = self.imreshSum/self.index
self.SkySumFinal = self.SkySum/self.index
self.imreshSumFinal = self.imreshSumFinal - self.SkySumFinal

print ('self.imreshSumFinal', self.imreshSumFinal)
np.savetxt('data_a.txt', self.imreshSumFinal)
###########################################################
## Looking for the center of the Sun (resulting picture) ##
###########################################################
b = np.loadtxt('data_a.txt')
c = np.loadtxt('data_a.txt')
```

```
for j in range(0,1619):
    for i in range(0,1219):
        if (b[i,j] < self.threshold):
            b[i,j] = 0
        else: b[i,j] = 1

# 4 cycles in order to find (left, right, top and bottom) points of the
#then we will connect them with the lines and find the center by inters
2 lines
found = False
for j in range(0,1619):
    if found:
        break
    for i in range(0,1219):
        if (b[i,j] == 1) and (b[i+1,j] == 1):
            self.FirstPoint = [j,i]
            found = True
            break


found = False
for j in range(1619-1, -1, -1):
    if found:
        break
    for i in range(0,1219):
        if (b[i,j] == 1) and (b[i+1,j] == 1):
            self.SecondPoint = [j,i]
            found = True
            break


found = False
for i in range(0,1219):
    if found:
        break
    for j in range(0,1619):
        if (b[i,j] == 1) and (b[i+1,j] == 1):
            self.ThirdPoint = [j,i]
            found = True
            break

found = False
for i in range(1219-1, -1, -1):
    if found:
        break
    for j in range(0, 1619):
        if (b[i,j] == 1) and (b[i+1,j] == 1):
            self.FourthPoint = [j,i]
            found = True
```

```
                break
#NAgain, mathematical formula for the intersection of the two lines (i.
sun):
self.Gx = ((self.FirstPoint[0]*self.SecondPoint[1]-self.FirstPoint[1]*s
*(self.ThirdPoint[0]-self.FourthPoint[0])
      -(self.FirstPoint[0]-self.SecondPoint[0])*(self.ThirdPoint[0]*self
      -self.ThirdPoint[1]*self.FourthPoint[0])
      )//((self.FirstPoint[0]-self.SecondPoint[0])*(self.ThirdPoint[1]-s
      -(self.FirstPoint[1]-self.SecondPoint[1])*
         (self.ThirdPoint[0]-self.FourthPoint[0]))

self.Gy = ((self.FirstPoint[0]*self.SecondPoint[1]-self.FirstPoint[1]*s
*(self.ThirdPoint[1]-self.FourthPoint[1])
      -(self.FirstPoint[1]-self.SecondPoint[1])*(self.ThirdPoint[0]*self
      -self.ThirdPoint[1]*self.FourthPoint[0])
      )//((self.FirstPoint[0]-self.SecondPoint[0])*(self.ThirdPoint[1]-s
      -(self.FirstPoint[1]-self.SecondPoint[1])*
         (self.ThirdPoint[0]-self.FourthPoint[0]))
# Drawing the data on the canvas
ResultPanel.listFigure[0].set_canvas(ResultPanel.listFigureCanvas[0])
# Clearing the axes
ResultPanel.listAxe[0].clear()
ResultPanel.listAxe[0].imshow(c)
# Setting the axes labels
ResultPanel.listAxe[0].set_xlabel(r'Linear position')
ResultPanel.listAxe[0].set_ylabel('Intensity (ADU)')
# Setting the central points of the sun
ResultPanel.listAxe[0].plot([self.FirstPoint[0]], [self.FirstPoint[1]],
marker='^', color='g')
ResultPanel.listAxe[0].plot([self.SecondPoint[0]], [self.SecondPoint[1]
marker='o', color='r')
ResultPanel.listAxe[0].plot([self.ThirdPoint[0]], [self.ThirdPoint[1]],
marker='s', color='y')
ResultPanel.listAxe[0].plot([self.FourthPoint[0]], [self.FourthPoint[1]
marker='^', color='y')
ResultPanel.listAxe[0].plot((self.FirstPoint[0], self.SecondPoint[0]),
(self.FirstPoint[1], self.SecondPoint[1]), color = 'g')
ResultPanel.listAxe[0].plot((self.ThirdPoint[0], self.FourthPoint[0]),
(self.ThirdPoint[1], self.FourthPoint[1]), color = 'g')

# Putting the center to the picture
ResultPanel.listAxe[0].plot([self.Gx], [self.Gy], marker='x', color='g'
ResultPanel.listFigureCanvas[0].draw()
###########################################
##The intensity plot (resulting picture)##
###########################################
# Drawing the data on the canvas
ResultPanel.listFigure[1].set_canvas(ResultPanel.listFigureCanvas[1])
# Clearing the axes
ResultPanel.listAxe[1].clear()
ResultPanel.listAxe[1].plot(self.imreshSumFinal[
        (self.ThirdPoint[1]-50):(self.FourthPoint[1]+50),self.Gx], 'r')
#Horizontal profile

ResultPanel.listAxe[1].plot(self.imreshSumFinal[
        self.Gy,(self.FirstPoint[0]-50):(self.SecondPoint[0]+50)], 'b')

ResultPanel.listFigureCanvas[1].draw()
####################
## Polynomial fit ##
####################
#We should firstly make both arrays having the same size
#(Il array has one extra element because it helped us with the calculat

#Calculating \tau, optical depth in order to find the source function
# (with source function we can calculate an effective temperature from

del self.IlFIN[-1]
del self.Il[-1]
print('Intensity␣ratio:␣' + str(self.Il))

self.miuk1 = [0 for x in range(5)]
self.Il1 = [0 for x in range(5)]

j=0;
for i in range (9):
    if self.miuk[i] < 0.9:
        self.miuk1[j] = self.miuk[i]
        self.Il1[j]=self.Il[i]
        j=j+1

print ("miuk1␣", self.miuk1)
print ("Il1␣", self.Il1)

n=3
# roughly speaking it provides you with any possible test
results = sm.OLS(self.Il1, np.vander(self.miuk1, n)).fit()
print (results.summary())
# these parameters should be the same as for polyfit (and they are the
# the only difference here is that n=3, I don't know why:))
print ('Parameters:', results.params)
self.p2 = results.params
self.p2Array[self.f] = self.p2
print('Standard␣errors:␣', results.bse)

print('R2:␣', results.rsquared)

#Coefficients
global globalA0
global globalA1
global globalA2
#Errors to be shown in the final table
global globalA0Error
global globalA1Error
global globalA2Error
# Result for R squared test to be shown in the resulting table
global globalRSquaredTest

print("␣")
print("p2␣=␣" + str(self.p2))
print("a0␣=␣" + str(self.p2[2]))
print("a1␣=␣" + str(self.p2[1]))
print("a2␣=␣" + str(self.p2[0]/2))
print("␣")

#getting the values of the coefficients for one particular filter
globalA0 = self.p2[2]
globalA1 = self.p2[1]
globalA2 = self.p2[0]/2
# errors for coefficients
globalA0Error = results.bse[2]
globalA1Error = results.bse[1]
globalA2Error = results.bse[0]/2
# R squared test
globalRSquaredTest = results.rsquared

self.tau = np.arange(0., 2., 0.01)
#self.S = (self.Ilambd[self.f])*(self.p2[2] + self.tau*self.p2[1] + (se
self.S = (self.Ilambd)*(self.p2[2] + self.tau*self.p2[1] + (self.tau**2

#self.Ratio1[self.f] = (self.hPl*self.clight)/((self.kBolz*self.lambd[s
self.Ratio1[self.f] = (self.hPl*self.clight)/((self.kBolz*self.lambd))
print ("Ratio1␣", self.Ratio1)
#self.Ratio2[self.f] = (2*self.hPl*(self.clight**2))/(self.lambd[self.f
self.Ratio2[self.f] = (2*self.hPl*(self.clight**2))/(self.lambd**5)
print ("Ratio2␣", self.Ratio2)

self.Ttaul = self.Ratio1[self.f] / (np.log(1 + self.Ratio2[self.f]/self

self.tau23 = 0.666
#self.Seff = (self.Ilambd[self.f])*(self.p2[2] + (self.tau23)*self.p2[1
self.Seff = (self.Ilambd)*(self.p2[2] + (self.tau23)*self.p2[1] + (self

self.Teff = self.Ratio1[self.f] / (np.log(1 + self.Ratio2[self.f]/self.
# calculating the result error for the coefficients
k=9
WholeError = ((results.bse[2]/results.params[2])**2 + (results.bse[1]/r
              +(results.bse[0]/results.params[0]/2)**2
              +2*(results.bse[2]/results.params[2])*(results.bse[1]/res
                +2*(results.bse[1]/results.params[1])*(results.bse[0]/res
                +2*(results.bse[0]/results.params[0]/2)*(results.bse[2]/r
#SEr = np.sqrt(WholeError**2 + self.deltalambd[self.f]**2 )
SEr = np.sqrt(WholeError**2 + self.deltalambd**2 )
#TeffError = np.sqrt((self.deltalambd[self.f])**2 + (np.var(self.miuk)/
#                 +2*self.deltalambd[self.f]*np.var(self.miuk)/(k+1) +2*s
TeffError = np.sqrt((self.deltalambd)**2 + (np.var(self.miuk)/(k+1))**2
                 +2*self.deltalambd*np.var(self.miuk)/(k+1) +2*self.delta

self.IlArray[self.f] = self.Il
self.TeffArray[self.f] = self.Teff
self.TtaulArray[self.f] = self.Ttaul

self.polyval = np.polyval(self.p2Array[self.f],self.miuk1)
#-----------------------------------------------------------------------#
# assiging global variables to retrieve their values later
global globalTeff
globalTeff = self.Teff
global globalTeffError
globalTeffError = TeffError
global globalTtaul
globalTtaul = self.Ttaul
global globalIl
globalIl = self.Il

global globalmiuk
globalmiuk = self.miuk
global globalmiuk1
globalmiuk1 = self.miuk1

global globalPolyval
globalPolyval = self.polyval

self.ColShape = ['rv', 'rv', 'bs', 'c.', 'm^', 'b:', 'rv', 'rv', 'bs',
'c.', 'm^', 'b:', 'rv', 'rv', 'bs', 'c.', 'm^', 'b:']
self.ColShape2 = ['rv', 'rv', 'r-', 'b-.', 'm--', 'b:', 'rv', 'rv',
'r-', 'b-.', 'm--', 'b:', 'rv', 'rv', 'r-', 'b-.', 'm--', 'b:']
self.Color = ['y', 'm', 'r', 'b', 'c', 'm', 'y', 'm', 'r', 'b',
'c', 'm', 'y', 'm', 'r', 'b', 'c', 'm']
self.Color2 = ['y', 'r', 'r', 'b.', 'm', 'b', 'y', 'r', 'r', 'b.',
'm', 'b', 'y', 'r', 'r', 'b.', 'm', 'b']
self.Shape = [u'D', u'v', u's', u'.', u'^', u'o',u'D', u'v', u's', u'.'
self.Shape2 = [u'--', u'-', u'--', u'-.', u'--', u':', u'--', u'-', u'-
self.lines = [0 for col in range(7)]
self.labels = [0 for col in range(7)]

#Drawing the pre-last curve
# Drawing the data on the canvas
ResultPanel.listFigure[2].set_canvas(ResultPanel.listFigureCanvas[2])
# Clearing the axes
```

```
        ResultPanel.listAxe[2].clear()
        ResultPanel.listAxe[2].set_xlabel(r'$\mu$')
        ResultPanel.listAxe[2].set_ylabel('Intensity ratio I(0,' + r'$\mu$)' +
        ResultPanel.listAxe[2].scatter(self.miuk, self.IlArray[self.f], marker=
        ResultPanel.listAxe[2].plot(self.miuk1, self.polyval)

        ResultPanel.listFigureCanvas[2].draw()

        #Drawing the last curve
        # Drawing the data on the canvas
        ResultPanel.listFigure[3].set_canvas(ResultPanel.listFigureCanvas[3])
        # Clearing the axes
        ResultPanel.listAxe[3].clear()
        ResultPanel.listAxe[3].set_xlabel(r'$\tau$')
        ResultPanel.listAxe[3].set_ylabel('Temperature T(' + r'$\tau$)')

        ResultPanel.listAxe[3].plot(self.tau, self.Ttaul, linestyle = self.Shap
        ResultPanel.listAxe[3].plot(self.tau23, self.Teff, marker = '*')
        ResultPanel.listFigureCanvas[3].draw()


#################################################################################
''' Our TabPanel Class to calculate and show things:
    It is the Main Panel which will contain others,
    such as DataPanel and ResultPanel '''
class TabPanel(wx.Panel):
    """
    A simple wx.Panel class
    """
    #----------------------------------------------------------------------
    def __init__(self, parent):
        wx.Panel.__init__(self, parent)
        #First retrieve the screen size of the device
        screenSize = wx.DisplaySize()
        screenWidth = screenSize[0]/2

        ''' DataPanel Class will retrieve and show data obtained '''
        DataPanel = ScrolledPanel(self,-1, size=(screenWidth,400),
                                                    style=wx.SIMPLE_BORDER)
        DataPanel.SetupScrolling()
        DataPanel.SetBackgroundColour('#FDDF99')

        self.dataPanel = DataPanel

        ''' ResultPanel Class will show us the result of the calculation '''
        ResultPanel = wx.Panel(self,-1,size=(screenWidth,400), style=wx.SIMPLE_
        ResultPanel.SetBackgroundColour('#FFFFFF')

        #creating a sizer to manage two panels on parent panel
        self.panelSizer = wx.BoxSizer(wx.HORIZONTAL)
        self.panelSizer.Add(DataPanel, 0, wx.EXPAND|wx.ALL,border=5)
        self.panelSizer.Add(ResultPanel, 0, wx.EXPAND|wx.ALL,border=5)
        self.SetSizer(self.panelSizer)
        #------------------------------------------------------------------------#
        DataPanel.listPaths = []
        DataPanel.listLength = 0
        # This is for additional information for instance skyiamges which we ar
        DataPanel.listSkyPaths = []
        DataPanel.listSkyLength = 0
        # variable to distinguish between even and odd indecies of gridsizer
        self.number = 0
        # the index of data in the list to find the right one
        self.index = 0
        # Chosing a threshold
        DataPanel.listShowFigure = [] # empty list
        DataPanel.listShowAxe = []
        DataPanel.listShowFigureCanvas = []

        DataPanel.listFigure = [] # empty list
        DataPanel.listAxe = []
        DataPanel.listFigureCanvas = []
        #------------------------------------------------------------------------#
        ResultPanel.label1 = wx.StaticText(ResultPanel, label = " Browse your p
        ResultPanel.label2 = wx.StaticText(ResultPanel, label = " Additional in
        #------------------------------------------------------------------------#
        ResultPanel.labelChoseFile = wx.StaticText(ResultPanel, label = " Choos
        ResultPanel.chosenPath = wx.TextCtrl(ResultPanel, value = "", size = (-
        # for testing aponing button
        ResultPanel.browseButton = wx.Button(ResultPanel, -1, "Open")
        # Adding an event handling to the browseButton
        ResultPanel.browseButton.Bind(wx.EVT_BUTTON, lambda event,
                                arg1 = DataPanel, arg2 = ResultPanel: self.onOp
        #------------------------------------------------------------------------#
        # For the additional information
        ResultPanel.labelChoseAdditionalFile2 = wx.StaticText(ResultPanel, labe
        Choose the file(s): ")
        ResultPanel.chosenAdditionalPath = wx.TextCtrl(ResultPanel, value = "",
        # for testing oponing button
        ResultPanel.browseAdditionalButton = wx.Button(ResultPanel, -1, "Open")
        # Adding an event handling to the browseButton
        ResultPanel.browseAdditionalButton.Bind(wx.EVT_BUTTON, lambda event,
                                arg1 = DataPanel, arg2 = ResultPanel: self.onOp
                                arg1, arg2))
        #------------------------------------------------------------------------#
        ResultPanel.labelThreshold = wx.StaticText(ResultPanel, label = " Thres
        ResultPanel.textCtrlThreshold = wx.TextCtrl(ResultPanel, value = "2500"
        #------------------------------------------------------------------------#
        self.Filters = ["U", "B", "V", "R", "I"]
        self.lambd = ["366e-9", "420e-9", "547e-9", "648e-9", "871e-9"]
        self.Ilambd = ["4.2e13", "4.5e13", "3.6e13", "2.8e13", "1.6e13"]
```

```
self.deltalambd = ["17.5", "36.5", "45", "78.5", "118"]
ResultPanel.labelFilterLambda = wx.StaticText(ResultPanel, label = '␣Cu
ResultPanel.labelFilterILambda = wx.StaticText(ResultPanel, label = '␣C
ResultPanel.labelFilterDeltaLambda = wx.StaticText(ResultPanel, label =
# creating the sizer for holding these labels and respective comboboxes
ResultPanel.sizerFilterLambda = wx.BoxSizer(wx.HORIZONTAL)
ResultPanel.sizerFilterILambda = wx.BoxSizer(wx.HORIZONTAL)
ResultPanel.sizerFilterDeltaLambda = wx.BoxSizer(wx.HORIZONTAL)
# for finding the right value of the right combobox
ResultPanel.comboSelection = ['' for f in xrange(0, 3)]

ResultPanel.labelLambda = wx.StaticText(ResultPanel, label = u'\u03BB'
# create a combo box
ResultPanel.comboboxLambda = wx.ComboBox(ResultPanel, choices=self.lamb
ResultPanel.comboboxLambda.Bind(wx.EVT_COMBOBOX, lambda event,
                        arg1 = ResultPanel: self.OnLambdaCombo(event, ar

ResultPanel.labelILambda = wx.StaticText(ResultPanel, label = u'I␣'+u'\
# create a combo box
ResultPanel.comboboxILambda = wx.ComboBox(ResultPanel, choices=self.Ilar
ResultPanel.comboboxILambda.Bind(wx.EVT_COMBOBOX, lambda event,
                        arg1 = ResultPanel: self.OnILambdaCombo(event, a

ResultPanel.labelDeltaLambda = wx.StaticText(ResultPanel, label = '\u03
# create a combo box for \Delta\lambda
ResultPanel.comboboxDeltaLambda = wx.ComboBox(ResultPanel, choices=self
ResultPanel.comboboxDeltaLambda.Bind(wx.EVT_COMBOBOX, lambda event,
                        arg1 = ResultPanel: self.OnDeltaLambdaCombo(even
#---------------------------------------------------------------------------#
ResultPanel.sizerFilterLambda.Add(ResultPanel.labelFilterLambda, propor
flag = wx.ALL)
ResultPanel.sizerFilterLambda.Add(ResultPanel.comboboxLambda,
proportion = 2, flag = wx.ALL)

ResultPanel.sizerFilterILambda.Add(ResultPanel.labelFilterILambda, prop
ResultPanel.sizerFilterILambda.Add(ResultPanel.comboboxILambda, proport

ResultPanel.sizerFilterDeltaLambda.Add(ResultPanel.labelFilterDeltaLamb
proportion = 1, flag = wx.ALL)
ResultPanel.sizerFilterDeltaLambda.Add(ResultPanel.comboboxDeltaLambda,
proportion = 2, flag = wx.ALL)
#---------------------------------------------------------------------------#
# Button for showing data
ResultPanel.buttonShowData = wx.Button(ResultPanel, -1, "Show")
# Adding an event handling to the showButton
ResultPanel.buttonShowData.Bind(wx.EVT_BUTTON, lambda event,
                        arg1 = DataPanel, arg2 = ResultPanel: self.onSho
#---------------------------------------------------------------------------#
ResultPanel.sizer1 = wx.BoxSizer(wx.VERTICAL)
ResultPanel.sizer2 = wx.BoxSizer(wx.VERTICAL)
```

```
#------------------------------------------------------------------------------#
# Sizer for holding stuff in Result Panel
ResultPanel.mainSizer = wx.BoxSizer(wx.VERTICAL)
#sizer for browsing information
ResultPanel.infoSizer = wx.BoxSizer(wx.VERTICAL)
ResultPanel.pathSizer = wx.BoxSizer(wx.HORIZONTAL)
# Sizer for adding additional information such as
ResultPanel.additionalInfoSizer = wx.BoxSizer(wx.HORIZONTAL)
#sizer for holding additional data such as thresholds and various parar
ResultPanel.sizerOtherData = wx.BoxSizer(wx.HORIZONTAL)
ResultPanel.sizerThreshold = wx.BoxSizer(wx.VERTICAL)

ResultPanel.sizerLambda = wx.BoxSizer(wx.VERTICAL)
ResultPanel.sizerILambda = wx.BoxSizer(wx.VERTICAL)
ResultPanel.sizerDeltaLambda = wx.BoxSizer(wx.VERTICAL)
# for holding results
ResultPanel.resultSizer = wx.GridSizer(rows = 2, cols=2, hgap=1, vgap=1

ResultPanel.infoSizer.Add(ResultPanel.sizer1, proportion = 1, flag = wx
ResultPanel.infoSizer.Add(ResultPanel.sizer2, proportion = 1, flag = wx

ResultPanel.sizer1.Add(ResultPanel.label1)
ResultPanel.sizer1.Add(ResultPanel.pathSizer)

ResultPanel.sizer2.Add(ResultPanel.label2)
ResultPanel.sizer2.Add(ResultPanel.additionalInfoSizer)
ResultPanel.sizer2.Add(ResultPanel.sizerOtherData)

# Adding button and textctrl to the pathSizer
ResultPanel.pathSizer.Add(ResultPanel.labelChoseFile, proportion = 1, f
ResultPanel.pathSizer.Add(ResultPanel.chosenPath, proportion = 6, flag
ResultPanel.pathSizer.Add(ResultPanel.browseButton, proportion = 1, fla
# Adding button and textctrl to the addtionalInfoSizer
ResultPanel.additionalInfoSizer.Add(ResultPanel.labelChoseAdditionalFil
proportion = 1, flag = wx.ALL)
ResultPanel.additionalInfoSizer.Add(ResultPanel.chosenAdditionalPath,
proportion = 6, flag = wx.ALL)
ResultPanel.additionalInfoSizer.Add(ResultPanel.browseAdditionalButton,
        proportion = 1, flag = wx.ALL)
# Adding the last things - threshold, parameters and show button
ResultPanel.sizerThreshold.Add(ResultPanel.labelThreshold, proportion =
ResultPanel.sizerThreshold.Add(ResultPanel.textCtrlThreshold, proportio
#for \lambda varable
ResultPanel.sizerLambda.Add(ResultPanel.labelLambda, proportion = 1, fl
ResultPanel.sizerLambda.Add(ResultPanel.sizerFilterLambda, proportion =
#for \I_\lambda variable
ResultPanel.sizerILambda.Add(ResultPanel.labelILambda, proportion = 1,
ResultPanel.sizerILambda.Add(ResultPanel.sizerFilterILambda, proportion
#ResultPanel.sizerILambda.Add(ResultPanel.textCtrlThreshold, proportion
#for \Delta\lambda variable
```

```
ResultPanel.sizerDeltaLambda.Add(ResultPanel.labelDeltaLambda, proporti
ResultPanel.sizerDeltaLambda.Add(ResultPanel.sizerFilterDeltaLambda, pr
#ResultPanel.sizerDeltaLambda.Add(ResultPanel.textCtrlThreshold, propo

#combining all sizers (for variable and show button) together
ResultPanel.sizerOtherData.Add(ResultPanel.sizerThreshold, proportion =
ResultPanel.sizerOtherData.Add(ResultPanel.sizerLambda, proportion = 2,
ResultPanel.sizerOtherData.Add(ResultPanel.sizerILambda, proportion = 2
ResultPanel.sizerOtherData.Add(ResultPanel.sizerDeltaLambda, proportion


ResultPanel.sizerOtherData.Add(ResultPanel.buttonShowData, proportion =
# Adding list of canvases which represnts and will show the final data
ResultPanel.listFigure = [] # empty list
ResultPanel.listAxe = []
ResultPanel.listFigureCanvas = []
# adding canvases to a grid
for n in xrange(0, 4):
    #adding the figures to the list
    ResultPanel.listFigure.append(plt.figure())
    # creating the axes
    ResultPanel.listAxe.append(ResultPanel.listFigure[n].add_subplot(1,
    ResultPanel.listFigureCanvas.append(FigureCanvas(ResultPanel, −1, F
    # Adding canvases to the grid sizer
    ResultPanel.resultSizer.Add(ResultPanel.listFigureCanvas[n], 0, wx.
# Combining sizers together
ResultPanel.mainSizer.Add(ResultPanel.infoSizer, proportion = 1, flag =
ResultPanel.mainSizer.Add(ResultPanel.resultSizer, proportion = 3, flag
# Setting main ResultPanel Sizer
ResultPanel.SetSizer(ResultPanel.mainSizer)
#————————————————————————————————————————————————————————————————————————#
# identifying global variables to plot the resulting graph
global globalIlArray

global globalIl
#print ("globalIlArray ", globalIlArray)
global globalTeffArray

global globalTtaulArray

global globalmiukArray
global globalmiuk1Array
global globalPolyvalArray
#identifying global variable to have an array of coefficients
global globalA0Array
global globalA1Array
global globalA2Array
#identifying tha same for errors in coefficients
global globalA0ErrorArray
global globalA1ErrorArray
```

```
    global globalA2ErrorArray
    # error in the temperatures
    global globalTeffErrorArray
    # For R sqruared test
    global globalRSquaredTestArray

''' ComboBox Methods '''
def OnLambdaCombo(self, event, ResultPanel):

    data = ResultPanel.comboboxLambda.GetValue()
    ResultPanel.comboSelection[0] = float(data)
    if data == self.lambd[0]:
        ResultPanel.labelFilterLambda.SetLabel(self.Filters[0])
    if data == self.lambd[1]:
        ResultPanel.labelFilterLambda.SetLabel(self.Filters[1])
    if data == self.lambd[2]:
        ResultPanel.labelFilterLambda.SetLabel(self.Filters[2])
    if data == self.lambd[3]:
        ResultPanel.labelFilterLambda.SetLabel(self.Filters[3])
    if data == self.lambd[4]:
        ResultPanel.labelFilterLambda.SetLabel(self.Filters[4])

def OnILambdaCombo(self, event, ResultPanel):

    data = ResultPanel.comboboxILambda.GetValue()
    ResultPanel.comboSelection[1] = float(data)
    if data == self.Ilambd[0]:
        ResultPanel.labelFilterILambda.SetLabel(self.Filters[0])
    if data == self.Ilambd[1]:
        ResultPanel.labelFilterILambda.SetLabel(self.Filters[1])
    if data == self.Ilambd[2]:
        ResultPanel.labelFilterILambda.SetLabel(self.Filters[2])
    if data == self.Ilambd[3]:
        ResultPanel.labelFilterILambda.SetLabel(self.Filters[3])
    if data == self.Ilambd[4]:
        ResultPanel.labelFilterILambda.SetLabel(self.Filters[4])

def OnDeltaLambdaCombo(self, event, ResultPanel):

    data = ResultPanel.comboboxDeltaLambda.GetValue()
    ResultPanel.comboSelection[2] = float(data)
    if data == self.deltalambd[0]:
        ResultPanel.labelFilterDeltaLambda.SetLabel(self.Filters[0])
    if data == self.deltalambd[1]:
        ResultPanel.labelFilterDeltaLambda.SetLabel(self.Filters[1])
    if data == self.deltalambd[2]:
        ResultPanel.labelFilterDeltaLambda.SetLabel(self.Filters[2])
    if data == self.deltalambd[3]:
        ResultPanel.labelFilterDeltaLambda.SetLabel(self.Filters[3])
    if data == self.deltalambd[4]:
        ResultPanel.labelFilterDeltaLambda.SetLabel(self.Filters[4])

''' Defining a dialog to open a file(or multiple files)'''
def onOpenFile1(self, event, DataPanel, ResultPanel):

    flat_list=[]
    # create a file dialog
    dlg = wx.FileDialog(
            self, message = "Choose␣a␣file",
            defaultDir = os.getcwd(),
            defaultFile = '',
            wildcard = "TIFF␣files␣(*.tif)|*.tif|BMP␣and␣GIF␣files
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(*.bmp;*.gif)|*.bmp;*.gif|PNG␣files␣(*.
            style=wx.FD_OPEN | wx.FD_MULTIPLE | wx.FD_CHANGE_DIR
            )
    # os.getcwd() (returns "a string representing the current working direc
    if dlg.ShowModal() == wx.ID_OK:
        # adding the values to the paths
        DataPanel.listPaths.append(dlg.GetPaths())
        #creating one list out of list of lists
        for sublist in DataPanel.listPaths:
            for item in sublist:
                flat_list.append(item)
            #print flat_list
            DataPanel.listPaths = flat_list
            # getting the number of elements in the list of data (self.list
            DataPanel.listLength = len(DataPanel.listPaths)
    #close the dialog
    dlg.Destroy()

    # putting the vaalues into the text control
    str1 = ''.join(DataPanel.listPaths)
    ResultPanel.chosenPath.SetValue(str1)

''' Defining a dialog to open a file(or multiple files)'''
def onOpenFile2(self, event, DataPanel, ResultPanel):

    # This method is for chosing additional information (for instance your
    flat_list=[]
    # create a file dialog
    dlg = wx.FileDialog(
            self, message = "Choose␣a␣file",
            defaultDir = os.getcwd(),
            defaultFile = '',
            wildcard = "TIFF␣files␣(*.tif)|*.tif|BMP␣and␣GIF␣files
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(*.bmp;*.gif)|*.bmp;*.gif|PNG␣files␣(*.png)|*.p
            style=wx.FD_OPEN | wx.FD_MULTIPLE | wx.FD_CHANGE_DIR
            )
    # os.getcwd() (returns "a string representing the current working direc
    if dlg.ShowModal() == wx.ID_OK:
        # adding the values to the paths
        DataPanel.listSkyPaths.append(dlg.GetPaths())
        #creating one list out of list of lists
        for sublist in DataPanel.listSkyPaths:
            for item in sublist:
                flat_list.append(item)
            #print flat_list
            DataPanel.listSkyPaths = flat_list
            # getting the number of elements in the list of data (self.list
            DataPanel.listSkyLength = len(DataPanel.listSkyPaths)
            #print listLength
    #close the dialog
    dlg.Destroy()

    str2 = ''.join(DataPanel.listSkyPaths)
    ResultPanel.chosenAdditionalPath.SetValue(str2)

''' Method for showing the data '''
def onShowImages(self, event, DataPanel, ResultPanel):
    # to retrieve the index of the chosen page
    #(the one we are working with now)
    global globalPageIndex

    # getting the value for the threshold from text Control
    ResultPanel.threshold = float(ResultPanel.textCtrlThreshold.GetValue())
    # Invoking methods from class Calculator
    Calculator().ShowImages(DataPanel, DataPanel.listLength, DataPanel.listl
              ResultPanel, globalPageIndex)

    #----------------------------------------------------------------------#
    #retrieving calculated data
    global globalTeff
    print ("globalTeff ", globalTeff)
    # error in effective temperature for each particular filter
    global globalTeffError

    global globalTtaul
    print ("globalTtaul ", globalTtaul)

    global globalmiuk

    global globalmiuk1

    global globalPolyval
    #instantiating the coefficients
    global globalA0
    global globalA1
    global globalA2
    #instantiating errors for coefficients
    global globalA0Error
    global globalA1Error
    global globalA2Error
    # result for rsquared test for each particular filter
    global globalRSquaredTest
    #---------------------------------------------------------------------------#
    #filling in the global array to show its data on the resulting screen
    globalTeffArray[globalPageIndex] = globalTeff
    print ("globalTeffArray", globalTeffArray)
    globalTeffErrorArray[globalPageIndex] = globalTeffError
    print ("globalTeffErrorArray", globalTeffErrorArray)
    globalTtaulArray[globalPageIndex] = globalTtaul
    globalIlArray[globalPageIndex] = globalIl

    globalmiukArray[globalPageIndex] = globalmiuk
    globalmiuk1Array[globalPageIndex] = globalmiuk1

    globalPolyvalArray[globalPageIndex] = globalPolyval
    #putting coefficients for different tabs into one array
    globalA0Array[globalPageIndex] = globalA0
    globalA1Array[globalPageIndex] = globalA1
    globalA2Array[globalPageIndex] = globalA2
    #putting errors for coefficients into one array
    globalA0ErrorArray[globalPageIndex] = globalA0Error
    globalA1ErrorArray[globalPageIndex] = globalA1Error
    globalA2ErrorArray[globalPageIndex] = globalA2Error
    #putting the results for each r squred test into array of data
    globalRSquaredTestArray[globalPageIndex] = globalRSquaredTest

################################################################################
''' Custom Dialog Class '''
class NameDialog(wx.Dialog):
def __init__(self, parent, id=-1, title="Filter Name"):
    wx.Dialog.__init__(self, parent, id, title, size=(240, 165))

    self.mainSizer = wx.BoxSizer(wx.VERTICAL)
    self.buttonSizer = wx.BoxSizer(wx.HORIZONTAL)

    self.label = wx.StaticText(self, label="Enter Name:")
    self.field = wx.TextCtrl(self, value = "", size=(240, 20))
    # creating the Ok button
    self.okButton = wx.Button(self, label = "Ok", id = wx.ID_OK)
    # creating the Close button, which will close dialog
    self.closeButton = wx.Button(self, label = "Close", id = wx.ID_CLOSE)

    self.mainSizer.Add(self.label, 0, wx.ALL, 8 )
    self.mainSizer.Add(self.field, 1, wx.ALL, 8 )

    # Adding OkButton to sizer
    self.buttonSizer.Add(self.okButton, 1, wx.ALL, 8 )
    # Adding CloseButton to sizer
    self.buttonSizer.Add(self.closeButton, 1, wx.ALL, 8 )

    self.mainSizer.Add(self.buttonSizer, 0, wx.ALL, 0)

    # Binding an event to  okButton
    self.Bind(wx.EVT_BUTTON, self.OnOK, id = wx.ID_OK)
    self.Bind(wx.EVT_TEXT_ENTER, self.OnOK)
    # Binding an event to closeButton
    self.Bind(wx.EVT_BUTTON, self.OnCancel, id = wx.ID_CLOSE)
    self.Bind(wx.EVT_TEXT_ENTER, self.OnCancel)

    self.SetSizer(self.mainSizer)
    #self.Layout()

    self.tabResult = None
    self.nameResult = None

''' Method for okButton '''
def OnOK(self, event):
    self.nameResult = self.field.GetValue()
    #if self.nameResult == '':
    if self.nameResult == '':
        #self.nameResult = None
        None
    else:
        self.tabResult = self.nameResult
        self.Destroy()

''' Method for closeButton '''
def OnCancel(self, event):
    self.nameResult = None
    self.tabResult = ''
    self.Destroy()
################################################################################
class AboutFrame(wx.Frame):
def __init__(self):
    #First retrieve the screen size of the device
    screenSize = wx.DisplaySize()
    screenWidth = screenSize[0]/2
    screenHeight = screenSize[1]/2

    wx.Frame.__init__(self, None, wx.ID_ANY, "About the program", size = (s

    DescriptionPanel = ScrolledPanel(self, -1, style=wx.SIMPLE_BORDER)
    VersionPanel = ScrolledPanel(self, -1, style=wx.SIMPLE_BORDER)

    DescriptionPanel.SetupScrolling()
    DescriptionPanel.SetBackgroundColour('#FDDF99')

    VersionPanel.SetupScrolling()
    VersionPanel.SetBackgroundColour('#FFFFFF')
    # creating an AUI manager
    self.auiManager = aui.AuiManager()
    # tell AuiManager to manage this frame
    self.auiManager.SetManagedWindow(self)
    # defining the notebook variable
    self.auiNotebook = aui.AuiNotebook(self)

    #self.auiNotebook.AddPage(self.panel, tabName)
    self.auiNotebook.AddPage(DescriptionPanel, 'Program Description')
    self.auiNotebook.AddPage(VersionPanel, 'Program Version')
    self.auiManager.Update()
##########################################################################
''' Method for managing listCTrl '''
class EditableListCtrl(wx.ListCtrl, listmix.TextEditMixin):
''' TextEditMixin allows any column to be edited. '''

#----------------------------------------------------------------------
def __init__(self, parent, ID=wx.ID_ANY, pos=wx.DefaultPosition,
             size=wx.DefaultSize, style=0):
    """Constructor"""
    wx.ListCtrl.__init__(self, parent, ID, pos, size, style)
    listmix.TextEditMixin.__init__(self)
##########################################################################
''' Panel for merging data '''
class CombinedInfoPanel(wx.Panel):
def __init__(self, parent):
    wx.Panel.__init__(self, parent)

    # for holding results
    self.resultSizer = wx.BoxSizer(wx.VERTICAL)
    self.graphSizer = wx.GridSizer(rows = 1, cols=2, hgap=1, vgap=1)

    self.tableSizer = wx.BoxSizer(wx.HORIZONTAL)
    #--------------------------------------------------------------------#
    #to retrieve the number of pages
    global globalNumberOfPages
    # retrieving the names of pages
    #global globalPageName
    global globalListPageName
    #--------------------------------------------------------------------#
    # identifying global variables to plot the resulting graph
    global globalIlArray

    global globalTeffArray
    global globalTeffErrorArray

    global globalTtaulArray
    #print ("globalTtaulArray ", globalTtaulArray)
    global globalmiukArray
    global globalmiuk1Array

    global globalPolyvalArray
    #Initialysing an array of coefficients
    global globalA0Array
    global globalA1Array
    global globalA2Array
    #initialising an array for errors of coefficients
    global globalA0ErrorArray
    global globalA1ErrorArray
    global globalA2ErrorArray
    # Instantiating the array of variable for R squared test
    global globalRSquaredTestArray
    #Weighted average of effective temperature
    self.TeffSum = 0
    self.TeffErrorSum = 0
    self.TeffErrorSum2 = 0
    self.TeffFinal = 0
    #self.numberOfTemperatures = 0
    for f in range(0, globalNumberOfPages):
        #self.TeffSum += globalTeffArray[f]
        self.TeffSum += globalTeffArray[f]/np.power(globalTeffErrorArray[f]
        self.TeffErrorSum += 1/np.power(globalTeffErrorArray[f],2)
        self.TeffErrorSum2 += np.power((globalTeffErrorArray[f]* np.power(g

    self.TeffFinal = self.TeffSum/self.TeffErrorSum
    # Calculating the error associated with final effective temperature
    self.TeffFinalError = self.TeffFinal*np.sqrt(self.TeffErrorSum2)

    self.tau = np.arange(0., 2., 0.01)
    self.tau23 = 0.666

    #-------------------------------------------------------------------------#
    self.listControlFinalDataTable = EditableListCtrl(self, style = wx.LC_R
    #-------------------------------------------------------------------------#
    # Adding list of canvases which represnts and will show the final data
    self.listFigure = [] # empty list
    self.listAxe = []
    self.listFigureCanvas = []
    # variables for grid table
    self.listGrid = []
    # adding canvases to a grid
    for n in xrange(0, 2):
        #adding the figures to the list
        self.listFigure.append(plt.figure())
        # creating the axes
        self.listAxe.append(self.listFigure[n].add_subplot(1, 1, 1))
        self.listFigureCanvas.append(FigureCanvas(self, -1, self.listFigure
        # Adding canvases to the grid sizer
        self.graphSizer.Add(self.listFigureCanvas[n], 0, wx.ALL)
```

```
#------------------------------------------------------------------------------------#
self.ColShape = ['rv', 'rv', 'bs', 'c.', 'm^', 'b:', 'rv', 'rv', 'bs',
self.ColShape2 = ['rv', 'rv', 'r-', 'b-.', 'm--', 'b:', 'rv', 'rv', 'r-
self.Color = ['y', 'm', 'r', 'b', 'c', 'm', 'y', 'm', 'r', 'b', 'c', 'm
#self.Color = ['' for col in range (globalNumberOfPages)]
self.Color2 = ['y', 'r', 'r', 'b.', 'm', 'b', 'y', 'r', 'r', 'b.', 'm',
self.Shape = [u'.', u'v', u's', u'.', u'^', u'o', u'.',
              u'v', u's', u'.', u'^', u'o', u'.', u'v', u's', u'.', u
self.Shape2 = [u'--', u'-', u'-.', u'-.', u'--', u':', u'--',
               u'-', u'-.', u'-.', u'--', u':', u'--', u'-', u'-.', u'
self.lines = [0 for col in range(globalNumberOfPages)]
self.lines2 = [0 for col in range(globalNumberOfPages)]
self.labels = [0 for col in range(globalNumberOfPages)]
self.labels2 = [0 for col in range(globalNumberOfPages)]
#------------------------------------------------------------------------------------#
#List of all available colors in python#
#for name, hex in matplotlib.colors.cnames.iteritems():
#    COLORS = name
#self.Color = random.choice(COLORS)
#print (self.Color)
#self.line = []
self.currentDirectory = os.getcwd()
self.saveButton = wx.Button(self, label="Save")
self.saveButton.Bind(wx.EVT_BUTTON, self.onSaveFile)
#------------------------------------------------------------------------------------#
#working with graphs
self.listAxe[0].set_xlabel(r'$\mu$')
self.listAxe[0].set_ylabel('Intensity␣ratio␣I(0,' + r'$\mu$)' + '/I(0,1
#------------------------------------------------------------------------------------#
self.listAxe[1].set_xlabel(r'$\tau$')
self.listAxe[1].set_ylabel('Temperature␣T(' + r'$\tau$)')

#drawing actual (resulting) data
for f in xrange(0, globalNumberOfPages):
    #making the graph more interactive
    self.line2 = mlines.Line2D([], [], color=self.Color[f], marker=self
                               markersize=5, label=globalListPageName[f]
    self.lines[f] = self.line2

    self.line3 = mlines.Line2D([], [], color=self.Color[f], marker=self
                               markersize=5, label=globalListPageName[f]
    self.lines2[f] = self.line3
    #for plotting in the first canvas
    self.listAxe[0].scatter(globalmiukArray[f], globalIlArray[f], marke
    self.listAxe[0].plot(globalmiuk1Array[f], globalPolyvalArray[f], co
    #for plotting on the second canvas
    self.listAxe[1].plot(self.tau, globalTtaulArray[f], linestyle = sel
    self.listAxe[1].plot(self.tau23, globalTeffArray[f], marker = '*')
    #horizontal line for temperature (different for each)
    self.listAxe[1].axhline(globalTeffArray[f], 0, self.tau23/2,
                    color='black', linestyle='--')
        # plotting the vertical line for temperature (supposed to be the same f
        self.listAxe[1].plot([self.tau23, self.tau23], [0, np.amax(globalTeffAr
                    color='black', linestyle='--')

        self.labels = [self.line.get_label() for self.line in self.lines]
        self.listAxe[0].legend(self.lines, self.labels)

        self.labels2 = [self.line.get_label() for self.line in self.lines2]
        self.listAxe[1].legend(self.lines2, self.labels2)

        self.listFigureCanvas[1].draw()
        #---------------------------------------------------------------------------#
        #putting data in the table
        self.listControlFinalDataTable.InsertColumn(0, "Quantities\Filters", wx
        for columnIndex in xrange (0, globalNumberOfPages):

            #here we are inserting the column
            self.listControlFinalDataTable.InsertColumn((columnIndex + 1), glob
        # inserting the names in the first column:
        self.listControlFinalDataTable.InsertStringItem(0, 'Averaged_Teff')
        self.listControlFinalDataTable.InsertStringItem(0, 'Teff')
        self.listControlFinalDataTable.InsertStringItem(0, 'R^2_test')
        self.listControlFinalDataTable.InsertStringItem(0, 'a2')
        self.listControlFinalDataTable.InsertStringItem(0, 'a1')
        self.listControlFinalDataTable.InsertStringItem(0, 'a0')

        self.trickyIndex = 0
        for columnIndex in xrange (1, (globalNumberOfPages+1)):
            #here we are inserting the first value of the row
            self.listControlFinalDataTable.SetStringItem(0, columnIndex, "%.2f"
            self.listControlFinalDataTable.SetStringItem(1, columnIndex, "%.2f"
            self.listControlFinalDataTable.SetStringItem(2, columnIndex, "%.2f"
            self.listControlFinalDataTable.SetStringItem(3, columnIndex, "%.2f"
            self.listControlFinalDataTable.SetStringItem(4, columnIndex, "%.2f"
            self.trickyIndex = self.trickyIndex + 1
        self.listControlFinalDataTable.SetStringItem(5, 1, "%.2f" % self.TeffFi
        #putting table in the sizer
        self.tableSizer.Add(self.listControlFinalDataTable, 1, wx.ALL)
        #---------------------------------------------------------------------------#
        #mixing everything up
        self.resultSizer.Add(self.graphSizer, 1, wx.ALL)
        self.resultSizer.Add(self.tableSizer, 1, wx.ALL)

        self.SetSizer(self.resultSizer)


#-----------------------------------------------------------------------------------#
''' Method for saving final table as .png file '''
def onSaveFile(self, event):
        """
        Create and show the Save FileDialog
        """
        dlg = wx.FileDialog(
            self, message="Save file as ...",
            defaultDir=self.currentDirectory,
            defaultFile="",
            wildcard="PNG files (*.png)|*.png|TIFF files
                        (*.tif)|*.tif|BMP (*.bmp)|*.bmp|GIF files (*.gif)|*.gif
            style=wx.FD_SAVE
            )
        if dlg.ShowModal() == wx.ID_OK:
            path = dlg.GetPath()
            print "You chose the following filename: %s" % path
        dlg.Destroy()

##########################################################################################
''' This frame is for merged data '''
class CombinedInfoFrame(wx.Frame):
    def __init__(self):
        #First retrieve the screen size of the device
        screenSize = wx.DisplaySize()
        screenWidth = screenSize[0]/2
        screenHeight = screenSize[1]/2

        wx.Frame.__init__(self, None, wx.ID_ANY, "Finalized Data", size = (scre
        combinedInfoPanel = CombinedInfoPanel(self)
        combinedInfoPanel.SetBackgroundColour('black')

        # Creating a sizers to show data

    def ShowFinalData(self, event):
        None
##########################################################################################
class MainFrame(wx.Frame):

    """
    wx.Frame class
    """
    #----------------------------------------------------------------------
    def __init__(self):
        #First retrieve the screen size of the device
        screenSize = wx.DisplaySize()
        screenWidth = screenSize[0]/1.1
        screenHeight = screenSize[1]/1.1

        wx.Frame.__init__(self, None, wx.ID_ANY, "SolarLimb",
                          size = (screenWidth, screenHeight))
        # defining a global (ok it is not that global) variablbe, which
        #is the identifier of the tab
```

```
self.tabIndex = 1

# creating the array of filter names
global globalListPageName
#putting a limited value for filters ,
#because there is not that many existing :)
globalListPageName = ["" for f in xrange(0, 100)]
# creating an AUI manager
self.auiManager = aui.AuiManager()
# tell AuiManager to manage this frame
self.auiManager.SetManagedWindow(self)
# defining the notebook variable
self.auiNotebook = aui.AuiNotebook(self)

# Bind an event to page of notebook
self.auiNotebook.Bind(aui.EVT_AUINOTEBOOK_PAGE_CHANGED, self.OnPageClic
self.auiNotebook.Bind(aui.EVT_AUINOTEBOOK_PAGE_CLOSE, self.OnPageClose)
# show the menu
self.MenuBar()
#-----------------------------------------------------------------------#
# identifying global variables to plot the resulting graph
global globalIlArray
globalIlArray = [0 for col in range(100)]

global globalTeffArray
globalTeffArray = [0 for col in range(100)]
# array of errors for temperatures
global globalTeffErrorArray
globalTeffErrorArray = [0 for col in range(100)]

global globalTtaulArray
globalTtaulArray = [0 for col in range(100)]

#array of distance parameters
global globalmiukArray
globalmiukArray = [0 for col in range(100)]
global globalmiuk1Array
globalmiuk1Array = [0 for col in range(100)]

# for graphics
global globalPolyvalArray
globalPolyvalArray = [0 for col in range(100)]

#for coeeficients a0, a1 and a2
global globalA0Array
globalA0Array = [0 for col in range(100)]
global globalA1Array
globalA1Array = [0 for col in range(100)]
global globalA2Array
globalA2Array = [0 for col in range(100)]
```

```
        #initialising an array for errors of coefficients
        global globalA0ErrorArray
        globalA0ErrorArray = [0 for col in range(100)]
        global globalA1ErrorArray
        globalA1ErrorArray = [0 for col in range(100)]
        global globalA2ErrorArray
        globalA2ErrorArray = [0 for col in range(100)]

        global globalRSquaredTestArray
        globalRSquaredTestArray = [0 for col in range(100)]

    ''' defining a menu bar method '''
    def MenuBar(self):

        # Setting up the menu.
        fileMenu= wx.Menu()
        addTab = fileMenu.Append(103, "Add", "Add")
        combinedData = fileMenu.Append(104, "Combine","Combine")
        aboutTheProgram = fileMenu.Append(wx.ID_ABOUT, "About","About")
        quitTheApp = fileMenu.Append(wx.ID_EXIT,"Exit","Close")

        # Creating the menubar.
        menuBar = wx.MenuBar()
        menuBar.Append(fileMenu,"File") # Adding the "filemenu" to the MenuBar
        self.SetMenuBar(menuBar)  # Adding the MenuBar to the Frame content.

        # adding an eventhandling to the addTab variable: creating new tab,
        # but before that the dialog window will apper asking about the name
        # of the tab
        self.Bind(wx.EVT_MENU, self.GetName, addTab)

        self.Bind(wx.EVT_MENU, self.CombinedData, combinedData)
        # adding the event handling for the about variable
        self.Bind(wx.EVT_MENU, self.AboutTheProgram, aboutTheProgram)
        # adding an eventhandling to the quitTheApp variable
        self.Bind(wx.EVT_MENU, self.OnQuit, quitTheApp)

        self.Centre()
        self.Show(True)

        #print self.pageIndex

    ''' defining method to quit the program from the menu '''
    def OnQuit(self, event):
        self.Close()

    ''' Method for adding new tab '''
    def AddTab(self, event, tabName):
        self.panel = TabPanel(self.auiNotebook)
        self.auiNotebook.AddPage(self.panel, tabName)
        self.auiManager.Update()
        #Getting the index of newly created tab
        #----------------------------------------------------------------------#
        # Splitting the page programmatically
        #self.auiNotebook.Split(self.tab_num, wx.RIGHT)
        # tell the manager to "commit" all the changes just made

        #self.tab_num += 1

        #to account for any changes, we need to do the following procedure
        # indentifying the global variable to address it from other classes
        global globalNumberOfPages
        globalNumberOfPages = self.auiNotebook.GetPageCount()


    ''' Method for rewriting the tab label (like a dialog window) '''
    def GetName(self, event):
        dialog = NameDialog(self)
        dialog.ShowModal()
        tabName = dialog.tabResult
        if tabName != "":
            self.AddTab(event, tabName)


    ''' Method for getting the index of the chosen tab (by click) '''
    def OnPageClicked(self, event):
        # to retrieve the index of the chosen page
        global globalPageIndex
        globalPageIndex = event.GetSelection()
        #print globalPageIndex
        # indentifying the global variable to address it from other classes
        global globalNumberOfPages
        globalNumberOfPages = self.auiNotebook.GetPageCount()
        # to retrieve the name of the page
        global globalPageName
        globalPageName = self.auiNotebook.GetPageText(globalPageIndex)
        #inserting the name of the tab into the array for showing final data
        globalListPageName[globalPageIndex] = globalPageName

    ''' method which will erase the page and you can get some information about
    def OnPageClose(self, event):
        # For accounting the deleted page (i.e. the reduced number),
        #we change the value of the global variable

        # to retrieve the index oof the chosen page
        global globalPageIndex
        globalPageIndex = event.GetSelection()
        #print globalPageIndex
        # indentifying the global variable to address it from other classes
        global globalNumberOfPages
        globalNumberOfPages = self.auiNotebook.GetPageCount()
        # to retrieve the name of the page
        global globalPageName
        globalPageName = self.auiNotebook.GetPageText(globalPageIndex)

    ''' Method for combining data from different tabs '''
    def CombinedData(self, event):

        global globalNumberOfPages
        globalNumberOfPages = self.auiNotebook.GetPageCount()
        # creating a condition that if we have a tab we can merge data
        if globalNumberOfPages > 0:
            combinedInfoFrame = CombinedInfoFrame()
            combinedInfoFrame.Show()

    ''' Method for about button from the Menu '''
    def AboutTheProgram(self, event):
        aboutFrame = AboutFrame()
        aboutFrame.Show()
#----------------------------------------------------------------------------
# Run the program
if __name__ == "__main__":
    #app = wx.PySimpleApp()
    app = wx.App(False)
    frame = MainFrame()
    frame.Show()
    #wx.lib.inspection.InspectionTool().Show()
    app.MainLoop()
    # To solve the problem - PyNoAppError: The wx.App object must be created fi
    del app
```