Documentation and report of Astrophysics with Artificial Intelligence(Astropy and AstroML) – UVES Spectroscopy with Astropy

Donghun Kim

* **All the information is based on and originated from ‘astropy.org’, ‘atsroml.org’, and ‘wikipedia.org’**

<1> The fundamental knowledge to utilize astropy and astroML for astrophysics

1. Accretion Disk

The accretion disk which is also known as circumstellar disk is the structure which is created by diffuse material in orbital motion around a massive central body. Typically, the central body is the star such as protostar, white dwarf, neutron star, or black hole and also there are theoretical stars such as black dwarf and blue dwarf. The instabilities in the disk causes redistribution of the angular momentum and to move spiral the materials inward toward the central body. In this process, the gravitational energy and frictional forces are converted to the thermal energy by compressing and raising the temperature of the material and this occurs the emission of electromagnetic radiation on the surface of the disk. The range of the frequency of the emission of electromagnetic radiation depends on the central body. For instance, the accretion disk of the protostar radiates the infrared and the neutron star and black hole emit the X-ray.

The equation of the angular momentum

L = r × p = r × mv

(m : mass of the material,

P : linear momentum of the material,

r : position vector from origin 0 to material,

v : velocity of the material)

According to the above principle, when the matter falls inward, the gravitational energy and angular momentum are decreased. However, the total angular momentum has to be sustained as comparable value, which is known as the law of angular momentum conservation. The equation of this:

dL / dt = (dr/dt × p) + (r × dp/dt)

= (v × mv) + (r × F)

# = 𝜏

( : torque(or moment) = r × F)

In other words, the loss of the angular momentum falling inward toward the central body should be compensated by the acquiring of the angular momentum far from the center. In short, angular momentum should be transported outwards from central body for material to accrete. This is because while the airframe of the inner orbit of the disk has rapid angular momentum, the outer one has slow angular momentum. In this process, the transportation of the angular momentum is occurred. According to the ‘Rayleigh stability criterion’,

∂(R2Ω) / ∂R > 0

(Ω : the angular velocity of a fluid element,

R : the distance to the rotation center)

In this, the accretion disk is supposed and expected as a laminar flow.

Plus, if it is needed, the angular momentum can be quantized.

L = nh / 2𝝅 = nħ (n = 1, 2, 3, ⋯)

sħ = h / 2𝝅

(h : Planck constant, = 6.626 × 10-34 [J·s],

ħ : Dirac’s constant, = 1.054 × 10-34 [J·s])

When it comes to the radiated energy of the accretion disk, when one of the proton does free falling, the relation between kinetic energy and gravitational energy is

1/2 × mv2 = GMm / r

When the material reaches at the surface of the star(r = R), the kinetic energy is radiated as form of thermal energy. If the proportion of the accretion which the matter accretes to the mass is dm/dt, the ratio of the attenuation of the energy at the surface of the star is 1/2 × dm/dt × v2. Thus, the luminosity of the star is

L = 1/2 × dm/dt × v2 = GMdm / Rdt

=  {\displaystyle \epsilon}εc2 × dm / dt

(ε : accretion efficiency = 2GM / c2R = 1/2 × Rsch/R)

This efficiency(ε) means the one which rest mass energy of the accreting matter converts into the thermal energy. In the above equation, according to the equation of accretion efficiency, the accretion efficiency is directly proportional with the how much the star is compacted.

1. Protostar

The protostar is the young star arose in the initial stage of star evolution which is made by concentrating of the molecular cloud in the interstellar medium. This star is initiated from when the density of the central molecular cloud increases, ended up with the stage of the T tauri star. In the final stage, the T tauri stellar wind is occurred, which epitomizes that the star starts the radiant of the energy at inner point after the stage which the star pulls the mass.

The thermal pressure of the dust grains and molecular which are the components of the molecular cloud and the gravitational restraint energy of the cloud sustain the equilibrium. However, once the derangement such as shock wave occurred by supernova explosion, spiral density wave, or encounter or collision with other molecular clouds effects on the molecular cloud and this derangement is sufficiently strong, the equilibrium is upset and the mass is clustered on the specific point by occurring the gravitational instability. Once the cloud initiates the shrinking, the minimum mass of the molecular cloud for gravitational concentration can be expressed and this is also known as Jeans Instability.

Mj = (9/4) × (1/2𝝅n)1/2 × (1/m2 ) × (kT/G)3/2

(n : the particle number density,

m : average mass of the gas particle in the interstellar cloud,

T : temperature of the gas)

There is the turbulent flow in the molecular cloud which the stars are formed and this compresses the gas in the form of shock wave which has variety of magnitude and density, and makes lump or striped shape structure. Once some part of structure overs the Jeans Instability, the gravity becomes instability, then it split and forms the multiple star system or solitary star system.

Because of the collision between the molecules they are laid excited states, and it is decayed by radiating the radioactive rays. When the cloud is shrunk, the molecular number density is increased and radiated radioactive rays are being difficult to flee away from cloud. This makes the gas to be invisible and the cloud has higher temperature.

1. Chain reaction

When it comes to the proton-proton chain reaction, this is one of the processes of the nuclear fusion reactions which the star converts the hydrogen to the helium. This phenomenon is only occurred when the temperature of the proton (in short, the mean kinetic energy) is sufficiently high to overcome the Coulomb force.

The first reaction is that the two of hydrogen atomic nucleus are fused to one of deuterium by converting one of proton to the neutron and through this process the positron and electron neutrino are released. This process depends on the weak interaction (also known as weak force or weak nuclear force) and the equation of this is

1H + 1H → 2H + [e+](https://ko.wikipedia.org/wiki/%EC%96%91%EC%A0%84%EC%9E%90) + [ν](https://ko.wikipedia.org/wiki/%EC%A4%91%EC%84%B1%EB%AF%B8%EC%9E%90)[e](https://ko.wikipedia.org/wiki/%EC%A4%91%EC%84%B1%EB%AF%B8%EC%9E%90)

(1H : hydrogen atomic nucleus,

2H : deuterium,

[e+](https://ko.wikipedia.org/wiki/%EC%96%91%EC%A0%84%EC%9E%90) : positron,

[νe](https://ko.wikipedia.org/wiki/%EC%A4%91%EC%84%B1%EB%AF%B8%EC%9E%90) : electron neutrino)

The positron is did pair annihilation with the electron of hydrogen and the energy is released as two of gamma ray photon.

e+ + e− → 2[γ](https://ko.wikipedia.org/wiki/%EA%B0%90%EB%A7%88%EC%84%A0) + 1.02 MeV

(eV : electronvolt, 1 eV = 1.602 × 10-19 J)

After the above process, the deuterium which is made in the first process fuses with other hydrogen and the isotope of helium, helium-3(3He) is created.

2H + 1H → 3He + [γ](https://ko.wikipedia.org/wiki/%EA%B0%90%EB%A7%88%EC%84%A0) + 5.49 MeV

Then, there are 4 methods which are generating the helium-4(4He). The first one is proton-proton I, also known as PP I and in this branch, helium-4 is fused from two of helium-3 nuclei; in the PP II and PP III branches, helium-3 is fused with pre-existing helium-4 forming the beryllium-7, but each processes utilize the different branches; in the PP IV, the helium-3 is formed by directly reacting the helium-3 with proton and this branch is also called as HeP(Helium-Proton reaction).

To be more specific, in the PP I branch, about 26.7 MeV of net energy. PP I process is dominant under the 10 ~ 14 MK of temperature. Below the 10 MK of temperature, the helium-4 is not much created.

3He +3He → 4He + 1H + 1H + 12.86 MeV

The PP II branch is dominant at temperatures of 14 ~ 23 MK.

3He + 4He → 7[Be](https://ko.wikipedia.org/wiki/%EB%B2%A0%EB%A6%B4%EB%A5%A8) + [γ](https://ko.wikipedia.org/wiki/%EA%B0%90%EB%A7%88%EC%84%A0)

7Be + e− →7[Li](https://ko.wikipedia.org/wiki/%EB%A6%AC%ED%8A%AC) + [ν](https://ko.wikipedia.org/wiki/%EC%A4%91%EC%84%B1%EB%AF%B8%EC%9E%90" \o "중성미자)[e](https://ko.wikipedia.org/wiki/%EC%A4%91%EC%84%B1%EB%AF%B8%EC%9E%90" \o "중성미자) (electron capture)

7Li + 1H → 4He + 4He

(electron capture : this is occurred when the parent nuclide absorbs one of the orbital electron radiating the neutrino)

In the second equation, the 90 percentage of forming neutrino has 0.861 MeV of energy and another 10 percentage has 0.383 MeV of energy. This depends on the condition whether the lithium-7 is at the excited state of ground state.

When it comes to the PP III branch, this is dominant at temperatures over 23 MK. This is not requisite resource at the sun which the inner temperature is not higher enough. However, with regard to the sun’s neutrino problem, it is one of the keystones for resolve it because, in this process, 14.06 of high energy of neutrino is created.

3He + 4He → 7Be + [γ](https://ko.wikipedia.org/wiki/%EA%B0%90%EB%A7%88%EC%84%A0)

7Be + 1H → 8[B](https://ko.wikipedia.org/wiki/%EB%B6%95%EC%86%8C) + [γ](https://ko.wikipedia.org/wiki/%EA%B0%90%EB%A7%88%EC%84%A0)

8B→ 8Be + e+ + [ν](https://ko.wikipedia.org/wiki/%EC%A4%91%EC%84%B1%EB%AF%B8%EC%9E%90" \o "중성미자)[e](https://ko.wikipedia.org/wiki/%EC%A4%91%EC%84%B1%EB%AF%B8%EC%9E%90" \o "중성미자)

8Be↔ 4He + 4He

Finally, the PP IV branch is the hypothesis which is the theoretically predicted, not actually observed. In this reaction, the helium-3 directly reacts with proton and forms the helium-4.

3He + 1H → 4He + νe + e+

In addition to those reactions, there is rare reaction which is called PEP reaction (Proton-Electron-Proton). This occurs instead of PP reaction. The proportion which this reaction occurs is lower than the one of other PP reactions, but this has higher energy of neutrino. In the PP I reaction, the neutrino has 0.42 MeV as maximum energy. In contrast with it, in the PEP reaction, the neutrino has 1.44 MeV of energy.

1. T tauri star

The fixed stars of the T tauri sort are the pre-main-sequence stars, which are composed with the F, G, K, M spectral type. The alternation of continuous spectrum and whole brightness are occurred with sporadic advent of the emission line. Those stars are similar with the main sequence in the mass, but the radiuses are larger, so generally, they are brighter than main sequences. However, the T tauri stars’ central temperatures are lower than main sequence, so it is impossible to be occurred the proton-proton chain reaction. In the process which the T tauri stars become the main sequences by being concentrated, the gravitational energy is emitted. Furthermore, they have 1 ~ 12 days of rotation periods which are drastically shorter than main sequence’s one.

With regard to the alternation of the luminosity, there are huge alternation in the ultraviolet and X-ray spectral. Especially, the change of X-ray luminosity in some stars reaches ten times per day. This phenomenon is concerned as the flare which is occurred because of the explosion of the activity of the photosphere.

In the spectrum of the T tauri star, there are some emission lines such as Balmer lines, ionized calcium, and other metals’ one. In some spectra of the T tauri stars, there is forbidden line which is the typical characteristic of the gaseous nebula. Moreover, there is more prolific lithium than in the sun and other main-sequence stars. This is because lithium is not destroyed under the 2.5 MK of the temperature.

The one of the main resources of pre-main-sequence star including T tauri star is the lithium burning through the proton-proton chain reaction. According to the conservation of angular momentum, the older the T tauri stars are, the faster the rotating speed is. This is because when the star is getting older, this becomes condensed. This causes the inner lithium of T tauri star to be faded quickly. The lithium burning is increased with higher temperature and mass. According to the proton-proton chain reaction, the lithium burning one is

p+ + 6Li → 7Be (unstable)

7Be + e- → 7Li + [γ](https://ko.wikipedia.org/wiki/%EA%B0%90%EB%A7%88%EC%84%A0)

p+ + 7Li → 8Be (unstable)

8Be → 4He + 4He + energy

1. Doppler effect(Doppler shift)

The Doppler shift is the change in frequency of a wave and wavelength itself in relation to an observer who moves relative to the wave source and vice versa. The wave which depends on the transmission medium such as sound has different relative velocity for the observer and wave source. However, like a light or magnetism which are not depended on the transmission medium, only the relative velocity between the observer and the wave source effects on the Doppler shift.

To begin with, when the observer moves close to the wave source,

f’ = (v + v0 / v) × f

Secondly, when the observer moves far from the wave source,

f’ = (v – v0 / v) × f

(f: genuine frequency of the wave,

f’: observed frequency by an observer,

v: the velocity of the wave in the transmission medium,

v0: the velocity of the observer over the transmission medium)

When it comes to this effect when the wave source moves, the equation is different with above one.

Firstly, when the wave source moves close to an observer,

f’ = (v / v – vs) × f

Secondly, when the wave source moves far from the observer,

f’ = (v / v + vs) × f

Finally, the general Doppler equation is:

f’ = (v + v0 / v – vs) × f

(vs: velocity of the wave source in the transmission medium)

With regard to the Relativistic Doppler effect, when the speed of observer and wave source is faster than the radio wave, or when the distance between the observer and wave source is smaller than the wavelength of radio wave, above equations are not reasonable and useful. This is because the relativistic Doppler effect presupposes the special waves which have to be concerned the time dilation effect and do not involve the medium of propagation such as light or magnetism.

f’ = [(1 + v / c)1/2 / (1 – v / c)1/2] × f

(v: the relative velocity of light source and observer,

f’: the frequency which an observer measures,

f: the frequency of light measured in the inertial frame of reference)

<2> Developing the programs

(https://learn.astropy.org/rst-tutorials/UVES.html?highlight=filtertutorials)

(Code)

**import** tarfile

**from** astropy.utils.data **import** download\_file

url **=** 'http://data.astropy.org/tutorials/UVES/data\_UVES.tar.gz'

f **=** tarfile**.**open**(**download\_file**(**url**,** cache**=True),** mode**=**'r|\*'**)**

working\_dir\_path **=** '.' *# CHANGE TO WHEREVER YOU WANT THE DATA TO BE EXTRACTED*

f**.**extractall**(**path**=**working\_dir\_path**)**

This allows us to get the essential data files for extracting the information of UVES by downloading and extracting the tar file.

By changing the ‘working\_dir\_path’, it is possible to save the information. If there is already exists the UVES file in the path which is entered in ‘working\_dir\_path’, this can provoke the ‘Permission denied’ error. By eliminating the pre-existing UVES file, changing the path (or creating the subordinate file like ‘C:/Users/’ -> ‘C:/Users/new\_file’), or just utilizing the existing UVES file, this problem can be resolved.

**from** glob **import** glob

**import** os

**import** numpy **as** np

**from** astropy.wcs **import** WCS

**from** astropy.io **import** fits

*# os.path.join is a platform-independent way to join two directories*

globpath **=** os**.**path**.**join**(**working\_dir\_path**,** 'UVES/\*.fits'**)**

print**(**globpath**)**

*# glob searches through directories similar to the Unix shell*

filelist **=** glob**(**globpath**)**

*# sort alphabetically - given the way the filenames are*

*# this also sorts in time*

filelist**.**sort**()**

sp **=** fits**.**open**(**filelist**[0])**

sp**.**info**()**

The glob method helps to handle the files. In above code, all the ‘.fits’[[1]](#footnote-1) files in the UVES file are returned as list form. For instance, glob(‘\*.txt’) returns all the ‘.txt’ files and directories like [‘astropy.txt’, ‘astroml.txt’] and if employ the glob(r‘C:\a\*’), it returns all the files and directories which have the alphabet ‘a’ at first place in the ‘C:’ file like [‘C:\astropy’, ‘C:\astroml’].

(Example code over the ‘glob’)

>>> glob('\*.txt')

['astropy.txt', 'astroml.txt']

>>> glob(r'C:\a\*')

['C:\astropy', 'C:\astroml']

The ‘.fits’ files in the UVES are saved list variable with ‘filelist = glob(globpath)’ and sorted with ‘filelist.sort()’. Then, utilize the ‘fits.open(‘list name’)’, read the information in the list. Therefore, in the above code, ‘sp’ variable contains the first file of the ‘filelist’, and ‘sp.info()’ method returns the saved information.

(The output value)

Filename**:** **./**UVES**/**r**.**UVES**.2011-08-11**T232352**.266-**A01\_0000**.**fits

No**.** Name Ver Type Cards Dimensions Format

**0** PRIMARY **1** PrimaryHDU **609** **(42751,)** float32

This is the information that the first file in the ‘filelist’, which is the one that in the UVES file, sorted alphabetically among ‘.fits’ files.

The data is given as the primary image and all other info is part of the primary header. Therefore, it is possible to extract the WCS[[2]](#footnote-2) from that header to get the wavelength coordinate.

(Code)

header **=** sp**[0].**header

wcs **=** WCS**(**header**)**

*#make index array*

index **=** np**.**arange**(**header**[**'NAXIS1'**])**

wavelength **=** wcs**.**wcs\_pix2world**(**index**[:,**np**.**newaxis**],** **0)**

wavelength**.**shape

*#Ahh, this has the wrong dimension. So we flatten it.*

wavelength **=** wavelength**.**flatten**()**

The ‘sp’ has the information about the first file of the ‘filelist’ and ‘header’ contains the header information about the first one of the ‘sp’, which means that ‘header’ variable has the header information of the ‘filelist’. The ‘index’ is the list which is arranged from 0 to ‘header[‘NAXIS1’]’, which has the 42751 number of values. This is because the ‘NAXIS1’ in the header is the number of data axes NAXIS1 = 42751 and this is comparable with the dimensions in the ‘sp.info()’. This can be checked by printing the ‘header’.

The ‘astropy.wcs’ contains some utilities which can manage the WCS transformations and this can work from pixel to world and from world to pixel. The ‘wavelength’ variable is the former way of the index variable from first value to ‘np.ewaxis’. Therefore, the ‘wavelength’ variable is made list form which contains the converted value from [3732.05623192] to [4999.73837591], which is the form which is all values in the list have the list value forms, so the ‘flatten’ process is necessary to adjust the wrong dimension.

(The process of the ‘wavelength.flatten()’)

Initial wavelength list:

[[3732.05623192] [3732.0858853] [3732.11553869] ... [4999.67906915] [4999.70872253] [4999.73837591]], shape: (42751, 1)

-> modified wavelength list:

[3732.05623192 3732.0858853 3732.11553869 ... 4999.67906915 4999.70872253 4999.73837591], shape: (42751,)

The flux is contained in the primary image.

flux **=** sp**[0].**data

sp : [<astropy.io.fits.hdu.image.PrimaryHDU object at 0x000001B41A256DC0>]. (len(sp) : 1)

The next step is creating the reusable function which returns the wavelength and flux arrays and the time of the observation.

(Code)

**def** read\_spec**(**filename**):**

*'''Read a UVES spectrum from the ESO pipeline*

*Parameters*

*----------*

*filename : string*

*name of the fits file with the data*

*Returns*

*-------*

*wavelength : np.ndarray*

*wavelength (in Ang)*

*flux : np.ndarray*

*flux (in erg/s/cm\*\*2)*

*date\_obs : string*

*time of observation*

*'''*

sp **=** fits**.**open**(**filename**)**

header **=** sp**[0].**header

wcs **=** WCS**(**header**)**

*#make index array*

index **=** np**.**arange**(**header**[**'NAXIS1'**])**

wavelength **=** wcs**.**wcs\_pix2world**(**index**[:,**np**.**newaxis**],** **0)**

wavelength **=** wavelength**.**flatten**()**

flux **=** sp**[0].**data

date\_obs **=** header**[**'Date-OBS'**]**

**return** wavelength**,** flux**,** date\_obs

This function is almost comparable with the above single codes excepting the part of ‘return’, previous of ‘sp’, and ‘date\_obs’. In this function, all the variables have each information, but those things do not return their own output values when their lines end, but return with the ‘return’ command. In short, all the variables in the function and above single variables have each same information, but, in the function, this information only can be returned with ‘return’ command key. The reason this function can be utilized reusable is the keystone of this code. In contrast with single code, in the function, ‘read\_spec(filename)’ and ‘sp’ parts have the ‘filename’ variable, which can get the different variables and returns different outputs. For instance,

if ‘filelist[10]’ is inserted in the ‘filename’, the output value is:

(array([3732.05623192, 3732.0858853 , 3732.11553869, ..., 4999.67906915, 4999.70872253, 4999.73837591]), <astropy.io.fits.hdu.image.PrimaryHDU object at 0x00000255692259A0>, '2011-08-12T02:57:42.000'),

but if ‘filelist[0]’ is inserted, the output value is:

(array([3732.05623192, 3732.0858853 , 3732.11553869, ..., 4999.67906915, 4999.70872253, 4999.73837591]), <astropy.io.fits.hdu.image.PrimaryHDU object at 0x000002556A986DC0>, '2011-08-11T23:23:52.266').

When it comes to the ‘date\_obs’ variable, this has the time as a value which is the total time of observing. The value is 2011-08-11T23:23:52.266

In the exercise part, above function can be applied to return other output values such as exposure time(EXPTIME), wavelength zero point (CRVAL1), and the arm used (HIERARCH ESO INS PATH).

(Code)

**def** read\_setup**(**filename**):**

*'''Get setup for UVES spectrum from the ESO pipeline*

*Parameters*

*----------*

*filename : string*

*name of the fits file with the data*

*Returns*

*-------*

*exposure\_time : float*

*wavelength\_zero\_point : float*

*optical\_arm : string*

*'''*

sp **=** fits**.**open**(**filelist**[0])**

header **=** sp**[0].**header

**return** header**[**'EXPTIME'**],** header**[**'CRVAL1'**],** header**[**'HIERARCH ESO INS PATH'**]**

*# Let's just print the setup on the screen*

*# We'll see if it's all the same.*

**for** f **in** filelist**:**

print**(**read\_setup**(**f**))**

By employing the loop method, all the information in the ‘filelist’ can be printed.

(Output value)

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

**(1200.0013,** **3732.05623191818,** 'BLUE'**)**

This result shows that all the fits files have same values in the exposure time, wavelength zero point, and the arm used. This means that the UVES pipeline that was used to reduce the data employs a fixed wavelength grid, thus the wavelength is the same for all spectra. This makes it easy to define an array that can hold the fluxes of all observations.

Other functions which return other values can be created by searching the ‘header’. For example, the function returns the size projected into a detector pixel in axis 1(CDELT1), minimum pixel value(DATAMIN), maximum pixel value(DATAMAX) is:

(Code)

**def** test(filename):

sp = fits.open(filename)

header = sp[0].header

**return** header['CDELT1'], header['DATAMIN'], header['DATAMAX']

(Output value)

**(0.0296533834852385, 0.0, 4729.290936)**

**(0.0296533834852385, 0.0, 4634.535313)**

**(0.0296533834852385, -174.290196, 4340.260218)**

**(0.0296533834852385, -6.782761, 4047.015691)**

**(0.0296533834852385, -11.655434, 3950.1721)**

**(0.0296533834852385, 0.0, 4090.752343)**

**(0.0296533834852385, -5.336826, 5406.309813)**

**(0.0296533834852385, -7.975554, 3984.424107)**

**(0.0296533834852385, -22.41758, 3731.275665)**

**(0.0296533834852385, -19.845119, 3170.01441)**

**(0.0296533834852385, -43.368462, 2920.626045)**

**(0.0296533834852385, -61.746478, 2279.674324)**

**(0.0296533834852385, -81.399828, 2302.1379)**

**(0.0296533834852385, 0.0, 5567.682116)**

**(0.0296533834852385, -7.969487, 5155.392448)**

**(0.0296533834852385, 0.0, 4880.609084)**

**(0.0296533834852385, 0.0, 6485.555281)**

**(0.0296533834852385, 0.0, 6268.045751)**

**(0.0296533834852385, 0.0, 6533.820193)**

**(0.0296533834852385, 0.0, 6118.381622)**

**(0.0296533834852385, -44.095477, 4457.756508)**

**(0.0296533834852385, 0.0, 4729.231238)**

**(0.0296533834852385, 0.0, 4974.708572)**

**(0.0296533834852385, -22.499702, 4687.619661)**

**(0.0296533834852385, -37.995828, 4656.190906)**

By creating the ‘flux’ and ‘date’ lists which are composed with comparable length of ‘filelist’ and ‘wavelength’ one, but initial forms are consist of only zeros, the blank lists are ready. Next, with the ‘for’ loop with ‘enumerate’ method’, those lists can have flux and date values, which are sorted same with the ‘filelist’. This is because the ‘flux’ list has the each value which is comparable with the ‘f’ has through the pre-made ‘read\_spec’ function and ‘date’ list is also. Furthermore, the ‘date’ list has ‘S23’ as a data type, so it is necessary to alternate that as a ‘U23’.

(Code)

flux **=** np**.**zeros**((**len**(**filelist**),** len**(**wavelength**)))**

*# date comes as string with 23 characters (dtype = 'S23')*

date **=** np**.**zeros**((**len**(**filelist**)),** dtype **=** 'U23'**)**

**for** i**,** fname **in** enumerate**(**filelist**):**

w**,** f**,** date\_obs **=** read\_spec**(**fname**)**

flux**[**i**,:]** **=** f

date**[**i**]** **=** date\_obs

All the values have to have their own units because those values are made by plenty of totally different other values. While merely some constants can be computed with others, but, in some cases, others which have the different unit values cannot be calculated with others. For instance, the mass has the [kg] as an unit value and the time has the [s] as an unit value. In this case, the addition of those values cannot mean any value, [kg] + [s]. Therefore, it is important to declare the obvious unit for mathematical and physical operations.

‘Astropy’ offers the units and constants modules.

(Code)

**import** astropy.units **as** u

**from** astropy.constants.si **import** c**,** G**,** M\_sun**,** R\_sun

wavelength **=** wavelength **\*** u**.**AA

*# Let's define some constants we need for the exercises further down*

*# Again, we multiply the value with a unit here*

heliocentric **=** **-23.** **\*** u**.**km**/**u**.**s

v\_rad **=** **-4.77** **\*** u**.**km **/** u**.**s *# Strassmeier et al. (2005)*

R\_MN\_Lup **=** **0.9** **\*** R\_sun *# Strassmeier et al. (2005)*

M\_MN\_Lup **=** **0.6** **\*** M\_sun *# Strassmeier et al. (2005)*

vsini **=** **74.6** **\*** u**.**km **/** u**.**s *# Strassmeier et al. (2005)*

period **=** **0.439** **\*** u**.**day *# Strassmeier et al. (2005)*

inclination **=** **45.** **\*** u**.**degree *# Strassmeier et al. (2005)*

*# All numpy trigonometric functions expect the input in radian.*

*# So far, astropy does not know this, so we need to convert the*

*# angle manually*

incl **=** inclination**.**to**(**u**.**radian**)**

Generally, the unit of wavelength is expressed nanometer scale and ‘u.AA’ is the 0.1nm (1-10), so to have nanometer as an unit value, ‘u.AA’ is multiply with the variable ‘wavelength’. The variable ‘heliocentric’ has the [km/s] as an unit value, which means that this means the velocity. To be more specific, the ‘heliocentric’ means the speed of the planets revolve around the central body. The ‘c’ is speed of lights in vacuum, 299792458[m/s], ‘G’ is gravitational constant, 6.674 × 10-11, ‘v\_rad’ is the radio velocity, ‘M\_sun’ is the mass of solar, 1.9884099 × 1030[kg], ‘R\_sun’ is the radius or solar, 6.957 × 108[m], and ‘vsini’ is projected rotational velocity. The variable ‘inclination’ has the degree unit vale, but for the next calculation and general expression method, alternate this value as radian one. The ‘MN\_Lup’ is a T Tauri star.

The MN Lup has accretion disk and through the spectra, the signatures are appeared close to the free-falling velocity ‘v’ when it reaches stellar surface and the material has mass ‘m’. From the relation between kinetic energy and gravitational energy, in above ‘Accretion disk’ part, the falling speed can be computed.

Ek = Eg

1/2 × mv2 = GMm / r

V = (2GM / r)1/2

From this equation, the free-falling velocity of the MN Lup can be calculated.

(Code)

v\_accr **=** **(2.\*** G **\*** M\_MN\_Lup**/**R\_MN\_Lup**)\*\*0.5**

print**(**v\_accr**)**

*# Maybe astronomers prefer it in the traditional cgs system?*

print**(**v\_accr**.**cgs**)**

*# Or in some really obscure unit?*

**from** astropy.units **import** imperial

print**(**v\_accr**.**to**(**imperial**.**yd **/** u**.**hour**))**

The variable ‘v\_accr’ is the free-falling velocity in the accretion disk. This is constituted with gravitational constant ‘G’, the mass of MN Lup ‘M\_MN\_Lup’ which is comparable with the ‘M’ in above equation, and radius of MN Lup ‘R\_MN\_Lup’ which is same with the ‘r’ in above equation. Then, convert this value to have the ‘CGS’ unit value. Not only ‘CGS’ system, but also other coordinate systems can be applied such as imperial units.

(Output value)

**504329.2809002659** m **/** s *# Fundamental unit(SI)*

**50432928.09002659** cm **/** s *# ‘CGS’ system*

**1985548350.0010464** yd **/** h *# Imperial units*

When it comes to the relation between accretion velocity and rotational one, the radial velocity observed through line broadening depends on the inclination of the pole of the star to the line-of-sight and this equation is:

ve sin(i) = v

ve = v / sin(i)

(ve: rotational velocity,

v: projected rotational velocity)

(Code)

v\_rot **=** vsini **/** np**.**sin**(**incl**)**

v\_accr **/** v\_rot

**(**v\_accr **/** v\_rot**).**decompose**()**

The variable ‘v\_rot’ is ve(rotational velocity, accretion velocity) and ‘vsini’ is v(= ve sin(i), projected rotational velocity). Therefore, the proportion of free-falling velocity in the accretion disk over the accretion velocity is 4780.357298599373 [m/km]. However, this result value has the wired unit value, so it is necessary to convert the length units by utilizing the ‘decompose’ method. Thus, the output value is 4.780357298599373. Not only the velocity values, but the wavelength values are also need to be alternated their units. The equation of the wavelength scale converting the heliocentric velocity scale is:

λheliocentric = λbarycentric × (1 + vheliocentric / c)

(λ : wavelength)

(Code)

wavelength **=** wavelength **\*** **(1.** **+** heliocentric**/**c**)**

However, this modified wavelength list also has wired unit value, [km/m] and ‘1.’ is just constant which has not any scale. To correct this problem, clarify the each scale.

(Code)

wavelength **=** wavelength **\*** **(1.** **\*** u**.**dimensionless\_unscaled**+** heliocentric**/**c**)**

Command ‘u.dimensionless\_unscaled’ means that the parameter which is multiplied with this has any scale or unit value.

The barycentric wavelength list:

[3732.05623192 3732.0858853 3732.11553869 ... 4999.67906915 4999.70872253 4999.73837591] Angstrom

-> the heliocentric wavelength list:

[3731.76990953 3731.79956064 3731.82921174 ... 4999.29549506 4999.32514617 4999.35479727] Angstrom

-> modified wavelength list(utilizing the ‘astropy.units.dimensionless\_unscaled’:

[3731.4836091 3731.51325794 3731.54290677 ... 4998.9119504 4998.94159923 4998.97124806] Angstrom

(When above code is run variety of times in the comparable console condition, the wavelength list can have different values each time)

This wavelength list can be converted as energy or frequency list by employing the equivalencies through the ‘to()’ conversion method.

(Code)

wavelength**.**to**(**u**.**keV**,** equivalencies**=**u**.**spectral**())**

wavelength**.**to**(**u**.**Hz**,** equivalencies**=**u**.**spectral**())**

In the first line, the wavelength list is converted to the energy list which has the [keV] unit value (called as electronvolt), and in the second line, the wavelength list is converted to the frequency list which has the [Hz] unit value (called as Hertz). The parameter ‘spectral()’ returns a list of equivalence pairs that handle spectral wavelength, wave number, frequency, and energy equivalences. The wavelength, energy, and frequency formula is:

v = f × λ

f = v / λ = ω / 2𝝅

(v: velocity, f: frequency, λ: wavelength, ω: angular frequency)

E = hf = ℏ ω

(h: Planck constant, = 6.602 × 10-34 [m2 kg / s],

ℏ: Dirac’s constant, = h / 2𝝅 = 1.054 × 10-34 [Js])

(Output value)

Energy: [0.00332265 0.00332263 0.0033226 ... 0.00248022 0.00248021 0.00248019] keV

Frequency: [8.03413573e+14 8.03407190e+14 8.03400806e+14 ... 5.99715420e+14

5.99711863e+14 5.99708306e+14] Hz

Spectroscopically, MN Lup is classified as spectral type M0 V, so the gravitational acceleration of it on the surface log(g) has similar value with the sun’s one. The gravitational acceleration at the sun is given as log(g) = 4.4. The gravitational acceleration ‘g’ is induced from ‘G × M / R2’. Therefore, the value of gravitational acceleration for MN Lup can be calculated with above given values of mass and radius of the MN Lup.

F1 = F2 = G × m1m2 / (r1 + r2 + r3)2 = G × m1m2 / R2

F = G × m1m2 / R2

mg = G× Mm / R2

g = G × M / R2

(r1, r2, r3: the radius of m1 and m2 and the distance of between m1 and m2)

(Code)

print**(**np**.**log10**((**G**\***M\_MN\_Lup**/**R\_MN\_Lup**\*\*2)/**u**.**cm**\***u**.**second**\*\*2))**

The value of it is ‘4.3077338588081275’.

Create the function which alternates the wavelength scale into a velocity scale. This function utilizes the ‘Doppler effect’ when the list of wavelength is given.

(Code)

waveclosetoHa **=** np**.**array**([6562.,6563,6565.])** **\*** u**.**AA

*# This function uses the Doppler equivalency between wavelength and velocity*

**import** astropy.units **as** u

**def** wave2doppler**(**w**,** w0**):**

w0\_equiv **=** u**.**doppler\_optical**(**w0**)**

w\_equiv **=** w**.**to**(**u**.**km**/**u**.**s**,** equivalencies**=**w0\_equiv**)**

**return** w\_equiv

print**(**wave2doppler**(**waveclosetoHa**,** **656.489** **\*** u**.**nm**).**to**(**u**.**km**/**u**.**s**))**

(Output value)

**[-131.9748242** **-86.30879506** **5.0232632** **]** km **/** s

Now, create the function which turns the wavelength array into the Doppler shift and by subtracting the radial velocity of MN Lup, expresses the resulting velocity in units of vsini.

(Example code)

**def** w2vsini**(**wavelength\_array**,** wavelength\_line**):**

w0\_equiv **=** u.doppler\_optical(w0)

w\_equiv **=** w.to(u.km/u.s, equivalencies**=**w0\_equiv)

array\_of\_shifts\_vsini **=** w\_equiv **\*** np.sin(incl)

**return** array\_of\_shifts\_in\_vsini

(Example output value)

**[-93.32029314 -61.02953426 3.55198348] km / s**

(Code)

**def** w2vsini**(**w**,** w0**):**

v **=** wave2doppler**(**w**,** w0**)** **-** **4.77** **\*** u**.**km**/**u**.**s

**return** v **/** vsini

The ‘v’ is [-136.7448242 -91.07879506 0.2532632] [km / s] and ‘vsini’ is 47.6 [km / s], so the resulting value of v / vsini is [-1.83304054 -1.22089538 0.00339495].

By utilizing the ‘astropy.time’, it is possible to convert the time. ESO FITS headers have the observation times in different systems and this can be read. To convert the times, the ‘format’ and ‘scale’ parameters should be obvious.

(Code)

**from** astropy.time **import** Time

t1 **=** Time**(**header**[**'MJD-Obs'**],** format **=** 'mjd'**,** scale **=** 'utc'**)**

t2 **=** Time**(**header**[**'Date-Obs'**],** scale **=** 'utc'**)**

t1

t1**.**isot

t2

t1**.**tt

(Output value)

**<**Time object**:** scale**=**'utc' format**=**'isot' value**=2011-08-11**T23**:23:52.266>**

**<**Time object**:** scale**=**'tt' format**=**'mjd' value**=55784.97567650852>**

This shows that times can be converted in different systems, formats, and initialized, so the time differences can be computed.

(Code)

obs\_times **=** Time**(**date**,** scale **=** 'utc'**)**

delta\_t **=** obs\_times **-** Time**(**date**[0],** scale **=** 'utc'**)**

delta\_p **=** delta\_t**.**value **\*** u**.**day **/** period

The ‘obs\_times’ is the observation time and in the variable ‘delta\_t’, the ‘Time(date[0])’ means the initial time of the observation. Therefore, the subtraction of initial time from ‘obs\_times’ means the time difference between two times. However, because of the variable ‘delta\_t’ has the ‘days’ as an unit value and the ‘astropy.time.Time’ and ‘astropy.units.Quantity’ do not work together, it is necessary to explicit the exact unit value.

(Output value)

-> delta\_t : [0. 0.01492581 0.02985729 0.0445986 0.05975544 0.07454605

0.08948135 0.10417763 0.11887623 0.13379172 0.14849229 0.16342042

0.17811449 0.99731943 1.01327258 1.02826374 1.05347384 1.06956417

1.08426017 1.09918675 1.11393485 1.12863501 1.14356171 1.15829649

1.17299041]

-> delta\_p : [0. 0.03399957 0.06801205 0.10159134 0.13611717 0.16980878

0.20382996 0.23730667 0.27078867 0.30476475 0.33825123 0.37225607

0.40572777 2.27179825 2.308138 2.34228642 2.39971263 2.43636484

2.46984094 2.50384225 2.53743701 2.57092258 2.60492418 2.63848859

2.67195992]

To estimate the equivalent width or make reasonable plots, it is essential to normalize the flux to the local continuum. In this specific process, the emission line is bright and the continuum can be described reasonably by a second-order polynomial and this allows estimate the time evolution of a single specific emission line in the spectrum.

To begin with, define two regions left and right of the emission line to fit the polynomial.

(Code)

**def** region\_around\_line**(**w**,** flux**,** cont**):**

*'''cut out and normalize flux around a line*

*Parameters*

*----------*

*w : 1 dim np.ndarray*

*array of wavelengths*

*flux : np.ndarray of shape (N, len(w))*

*array of flux values for different spectra in the series*

*cont : list of lists*

*wavelengths for continuum normalization [[low1,up1],[low2, up2]]*

*that described two areas on both sides of the line*

*'''*

*#index is true in the region where we fit the polynomial*

indcont **=** **((**w **>** cont**[0][0])** **&** **(**w **<** cont**[0][1]))** **|((**w **>** cont**[1][0])** **&** **(**w **<** cont**[1][1]))**

*#index of the region we want to return*

indrange **=** **(**w **>** cont**[0][0])** **&** **(**w **<** cont**[1][1])**

*# make a flux array of shape*

*# (number of spectra, number of points in indrange)*

f **=** np**.**zeros**((**flux**.**shape**[0],** indrange**.**sum**()))**

**for** i **in** range**(**flux**.**shape**[0]):**

*# fit polynomial of second order to the continuum region*

linecoeff **=** np**.**polyfit**(**w**[**indcont**],** flux**[**i**,** indcont**],** **2)**

*# divide the flux by the polynomial and put the result in our*

*# new flux array*

f**[**i**,:]** **=** flux**[**i**,**indrange**]** **/** np**.**polyval**(**linecoeff**,** w**[**indrange**].**value**)**

**return** w**[**indrange**],** f

wcaII**,** fcaII **=** region\_around\_line**(**wavelength**,** flux**,**

**[[3925\***u**.**AA**,** **3930\***u**.**AA**],[3938\***u**.**AA**,** **3945\***u**.**AA**]])**

The parameter ‘cont’ is [[3925\*u.AA, 3930\*u.AA], [3938\*u.AA, 3945\*u.AA]]. The symbol ‘&’ is intersection and ‘|’ is union. The variable ‘indcont’ is the union between intersections of first and second values of first ‘cont’ list (cont[0][0], cont[0][1]) which are each smaller and bigger values than wavelength and first and second values of second ‘cont’ list(cont[1][0], cont[1][1]). Each values of ‘cont’ mean that the maximum and minimum values of regions left and right of the emission line. The variable ‘indrange’ is the intersection of first value in the first ‘cont’ list(cont[0][0]) which is composed with smaller values than wavelength and second value of second ‘cont’ list(cont[1][1]) which is composed with bigger values than wavelength. However, variable ‘incont’ and ‘indragne’ have 42751 length, but all the components are consisted of ‘False’ values. The ‘flux.shape[0]’ is 25 and ‘indrange.sum()’ is 675, so the variable ‘f’ is 25 by 675 zero matrix. To be more specific, the variable ‘f’ is consisted of the number of 25 of rows and 675 of columns. The method ‘np.polyfit’ works to find the line of the grape with given parameters and the third one is the dimension, so in above code, the line is drawn as 2 dimension. The method ‘np.polyval’ is polynomial calculation method and this receives input list and input value. Once those values are given, the method regards the input list as polynomial and returns the result by inserting the input value into the given polynomial. For instance, if the input list is the ‘np.poly1d([1,-2,3])’ and input value is 2, the method convert the list to ‘x2-2x+3’ and substitutes the input value 2 into that equation, so the result is 3.

Calculate the equivalent width in Angstroms of the emission line for the first spectrum.

(Code)

ew **=** fcaII**[0,:]** **-** **1.**

ew **=** ew**[:-1]** **\*** np**.**diff**(**wcaII**.**to**(**u**.**AA**).**value**)**

print**(**ew**.**sum**())**

(Output value)

**20.21238214515653**

It is possible to process all spectra at once, if employ the ‘numpy’ array notation.

(Code)

delta\_lam **=** np**.**diff**(**wcaII**.**to**(**u**.**AA**).**value**)**

ew **=** np**.**sum**((**fcaII **-** **1.)[:,:-1]** **\*** delta\_lam**[**np**.**newaxis**,** **:],** axis**=1)**

All the first codes are included in second code variable ‘ew’, but this is utilized ‘numpy’ method. First of all, declare the ‘fcaII[0,:] – 1’ and multiply this with ‘np.diff(wcaII.to(u.AA).value)’. Finally, calculate the total summary of the list. In the second code, those processes are run at once in variable ‘ew’ with ‘np.sum’. Variable ‘delta\_lam’ is comparable with the first code second line. However, in contrast with the first one, this variable is declared independence one, so, in the second line of the second code, this has the range, which means that once the range is given, all the result values are returned as list.

(Output value)

**[20.21238214 21.24135599 20.64641441 21.93071095 20.33765219 18.72018334**

**18.88745051 19.64525953 20.71218865 19.10589383 20.32812425 18.12296315**

**17.60833685 19.13613271 18.98084058 18.60405201 19.81860353 20.14831276**

**20.76952636 20.2554641 20.25731161 21.0749123 21.21869543 22.53802562**

**19.47661968]**

With observation time, period, and equivalent width, make the LaTeX table. For this, the first process is collecting the information and making the table object.

(Code)

**from** astropy.table **import** Column**,** Table

**from** astropy.io **import** ascii

datecol **=** Column**(**name **=** 'Obs Date'**,** data **=** date**)**

pcol **=** Column**(**name **=** 'phase'**,** data **=** delta\_p**,** format **=** '{:.1f}'**)**

ewcol **=** Column**(**name **=** 'EW'**,** data **=** ew**,** format **=** '{:.1f}'**,** unit **=** '\\AA'**)**

tab **=** Table**((**datecol**,** pcol**,** ewcol**))**

*# latexdicts['AA'] contains the style specifics for A&A (\hline etc.)*

tab**.**write**(**os**.**path**.**join**(**working\_dir\_path**,** 'EWtab.tex'**),** latexdict **=** ascii**.**latexdicts**[**'AA'**])**

The variables ‘datecol’, ‘pcol’, and ‘ewcol’ have the information which is given as name parameter in the each data parameter. Therefore, each variable are ordered column data forms. With regard to the variable ‘tab’, this is the table which is receiving the above variables and this is made as data file named ‘EWtab.tex’.

After setting the data as a file which is table consisted of each given data in the form of column, create the plot by utilizing the ‘matplotlib’. The x-axis in the plot is Doppler shift expressed in units of the rotational velocity. In this process, the maximum flux rotationally modulated sticks out in range between -1 and 1.

(Code)

x **=** w2vsini**(**wcaII**,** **393.366** **\*** u**.**nm**).**decompose**()**

*# set reasonable figsize for 1-column figures*

fig **=** plt**.**figure**()**

ax **=** fig**.**add\_subplot**(1,1,1)**

ax**.**plot**(**x**,** fcaII**[0,:],** marker**=**''**,** drawstyle**=**'steps-mid'**)**

ax**.**set\_xlim**([-3,+3])**

ax**.**set\_xlabel**(**'line shift [v sin(i)]'**)**

ax**.**set\_ylabel**(**'flux'**)**

ax**.**set\_title**(**'Ca II H line in MN Lup'**)**

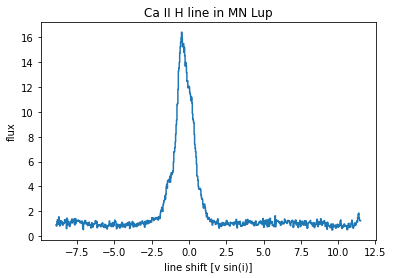
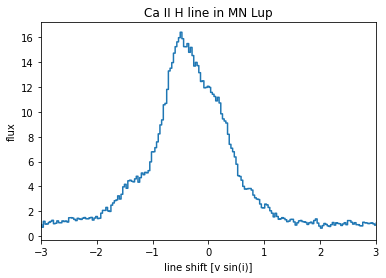
*# when using this interface, we need to explicitly call the draw routine*

plt**.**draw**()**

In this code, the method ‘add\_subplot’ makes the number of graphs wanting to draw. The first parameter is row, the second one is column, and the final one is selecting the subplot which is drawn in the ‘figure’. For example, if the variable ‘ax’ is ‘fig.add\_subplot(2, 1, 1)’ means that make the 2 by 1 matrix and select the first subplot, and if that is ‘fig.add\_subplot(2, 1, 2)’, make the 2 by 1 matrix, but select the second subplot. Furthermore, by employing the method ‘set\_xlim’, restrict the range of x-axis from -3 to 3.

The x-axis is substituted function ‘w2vsini’ and y-axis is substituted the variable ‘fcaII**’** which is the result value returned by the function ‘region\_around\_line’ range from first component to terminal one. The variable ‘fcaII’ is the list which is constituted with 25 length of sub-list and each one has 675 number of list. In the ‘numpy’, the ‘shape’ means the dimension of the matrix. For instance, if there is the matrix expressed (4, 2) is constituted with 4 number of rows and 2 number of columns. According to that, the variable ‘fcaII’ is the matrix which is composed with 25 number of rows and 675 number of columns.

(Graph)

This graph shows merely single spectrum. Draw all spectra into a only one plot and introduce the sensible offset between them by making that to follow the time evolution of the line.

The first graph shows the full size of graph which is made by above code by excepting the method ‘set\_xlim’. The second graph shows the constricted range of graph by employing the method ‘set\_xlim([-3, +3]).

(Code)

yshift **=** np**.**arange**((**fcaII**.**shape**[0]))** **\*** **0.5**

*#shift the second night up by a little more*

yshift**[:]** **+=** **1.5**

yshift**[13:]** **+=** **1**

fig **=** plt**.**figure**()**

ax **=** fig**.**add\_subplot**(1,1,1)**

**for** i **in** range**(25):**

ax**.**plot**(**x**,** fcaII**[**i**,:]+**yshift**[**i**],** 'k'**)**

*#separately show the mean line profile in a different color*

ax**.**plot**(**x**,** np**.**mean**(**fcaII**,** axis **=0))**

ax**.**set\_xlim**([-2.5,+2.5])**

ax**.**set\_xlabel**(**'line shift [$v \\sin i$]'**)**

ax**.**set\_ylabel**(**'flux'**)**

ax**.**set\_title**(**'Ca II H line in MN Lup'**)**

fig**.**subplots\_adjust**(**bottom **=** **0.15)**

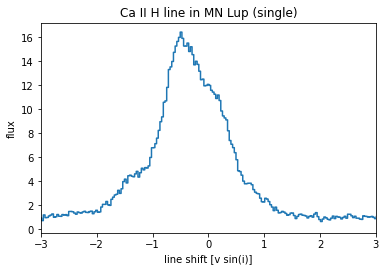
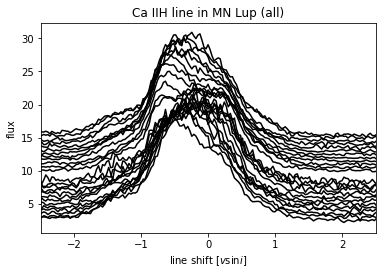
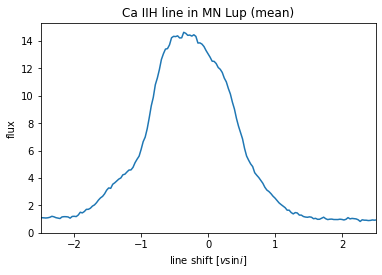
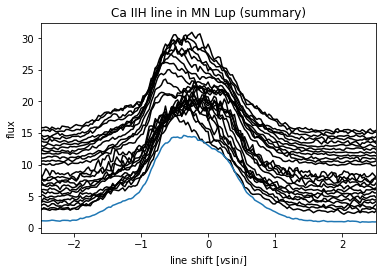
plt**.**draw**()**

Make the variable ‘yshift’ to have the 25 length of ‘numpy.ndarray’ which is comparable with the number of rows of ‘fcaII’ (like above explanation, ‘fcaII.shape[0]’ is the number of rows). Then, add the 1.5 to all the values in the ‘yshift’ list and add the 1 to the after 13 element values in the ‘yshift’.

In the ‘for’ loop paragraph, the range is set comparable with the length of ‘fcaII’. Then, plot the graph. In this process, the x-axis is same with the previous code which is drawn merely single spectrum, but, with regard to the y-axis, the shift value is added with the each element of ‘fcaII’ and this element value is depend on the unidentified value ‘i’, which means that all the values of ‘fcaII’ are different with each other ones.

The ‘ax.plot’ method in the paragraph 4 is drawing the graph of the mean value about the whole ‘fcaII’ values. Therefore, the graphs of above 4 parts of codes are:

(Graph)

According to above graphs, each flux has different value, so, by calculating the difference to the mean flux, it is possible to create the sophisticated and advance image graph.

(Code)

fmean **=** np**.**mean**(**fcaII**,** axis**=0)**

fdiff **=** fcaII **-** fmean**[**np**.**newaxis**,:]**

fig **=** plt**.**figure**()**

ax **=** fig**.**add\_subplot**(1,1,1)**

im **=** ax**.**imshow**(**fdiff**,** aspect **=** "auto"**,** origin **=** 'lower'**)**

However, this code is not setting the correct axis scale, gap between both nights, and proper labels, especially the gap between both nights should have the blank part. This is because above data is stemmed from the information depending on the observation date. Therefore, it is necessary to declare proper axis and separating the both nights’ spectra by creating the new variables to set the both nights and utilizing the ‘extent’ parameter in the ‘ax.imshow’ method to consider the axis.

(Code)

ind1 **=** delta\_p **<** **1** **\*** u**.**dimensionless\_unscaled

ind2 **=** delta\_p **>** **1** **\*** u**.**dimensionless\_unscaled

fig **=** plt**.**figure**()**

ax **=** fig**.**add\_subplot**(1,1,1)**

**for** ind **in** **[**ind1**,** ind2**]:**

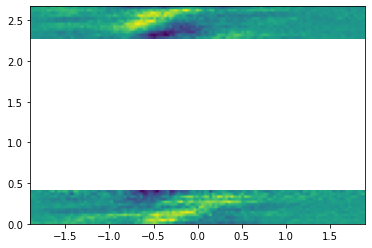
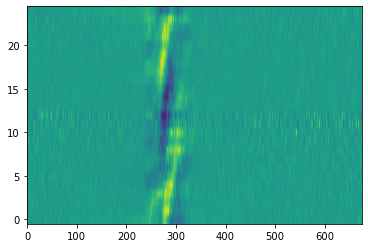
im **=** ax**.**imshow**(**fdiff**[**ind**,:],** extent **=** **(**np**.**min**(**x**),** np**.**max**(**x**),** np**.**min**(**delta\_p**[**ind**]),** np**.**max**(**delta\_p**[**ind**])),** aspect **=** "auto"**,** origin **=** 'lower'**)**

ax**.**set\_ylim**([**np**.**min**(**delta\_p**),** np**.**max**(**delta\_p**)])**

ax**.**set\_xlim**([-1.9,1.9])**

plt**.**draw**()**

(Graph)



The first image is not separated both nights and second image is separated both nights.

However, above terminal plot has some problems because this has not proper labels of axis, too large empty space, and improper scale. Regarding the last one, the parameter ‘extent’ gives the edges of the image, but parameters ‘x’ and ‘delta\_p’ have the middle points. Furthermore, the plot can have gray scale to save publication cost if it needs.

(Code)

*# shift a little for plotting purposes*

pplot **=** delta\_p**.**copy**().**value

pplot**[**ind2**]** **-=** **1.5**

*# image goes from x1 to x2, but really x1 should be middle of first pixel*

delta\_t **=** np**.**median**(**np**.**diff**(**delta\_p**))/2.**

delta\_x **=** np**.**median**(**np**.**diff**(**x**))/2.**

*# imshow does the normalization for plotting really well, but here I do it*

*# by hand to ensure it goes -1,+1 (that makes color bar look good)*

fdiff **=** fdiff **/** np**.**max**(**np**.**abs**(**fdiff**))**

fig **=** plt**.**figure**()**

ax **=** fig**.**add\_subplot**(1,1,1)**

**for** ind **in** **[**ind1**,** ind2**]:**

im **=** ax**.**imshow**(**fdiff**[**ind**,:],**

extent **=** **(**np**.**min**(**x**)-**delta\_x**,** np**.**max**(**x**)+**delta\_x**,**

np**.**min**(**pplot**[**ind**])-**delta\_t**,** np**.**max**(**pplot**[**ind**])+**delta\_t**),**

aspect **=** "auto"**,** origin **=** 'lower'**,** cmap **=** plt**.**cm**.**Greys\_r**)**

ax**.**set\_ylim**([**np**.**min**(**pplot**)-**delta\_t**,** np**.**max**(**pplot**)+**delta\_t**])**

ax**.**set\_xlim**([-1.9,1.9])**

ax**.**set\_xlabel**(**'vel in $v\\sin i$'**)**

ax**.**xaxis**.**set\_major\_locator**(**plt**.**MaxNLocator**(4))**

**def** pplot**(**y**,** pos**):**

'The two args are the value and tick position'

'Function to make tick labels look good.'

**if** y **<** **0.5:**

yreal **=** y

**else:**

yreal **=** y **+** **1.5**

**return** yreal

formatter **=** plt**.**FuncFormatter**(**pplot**)**

ax**.**yaxis**.**set\_major\_formatter**(**formatter**)**

ax**.**set\_ylabel**(**'period'**)**

fig**.**subplots\_adjust**(**left **=** **0.15,** bottom **=** **0.15,** right **=** **0.99,** top **=** **0.99)**

plt**.**draw**()**

1. fits : the standard data format used in astronomy, ‘Flexible Image Transport System’

   From : https://fits.gsfc.nasa.gov/ [↑](#footnote-ref-1)
2. WCS : World Coordinate System. Define keywords and usage that provide for the description of astronomical coordinate systems in a FITS image header and describe the geometric transformations between one set of coordinates and another. A common application is to map the pixel in an image onto the celestial sphere. Another common application is to map pixels to wavelength in a spectrum.

   From : <https://www.atnf.csiro.au/people/mcalabre/WCS/> [↑](#footnote-ref-2)