FUSS

For Use with Supernova Spectropolarimetry

v0.5

H. F. Stevance

January 20, 2017

Contents

1 Introduction **2**
- 1.1 Overview . . . 2
- 1.2 Python Dependencies . . . 2

2 Data Reduction **3**
- 2.1 IRAF Routines and associated PYTHON scripts . . . 3
- 2.2 Data Reduction Submodule: datred.py . . . 4
- 2.3 Data Reduction Procedure . . . 5
  - 2.3.1 Pre-reduction steps . . . 5
  - 2.3.2 Data Reduction Steps . . . 5

3 Utility Functions and Routines **7**
- 3.1 Functions . . . 7
- 3.2 Interactive Routines . . . 8

4 The PolData() class **9**
- 4.1 Core Attributes and Initialisation . . . 9
- 4.2 Optional attributes and related methods . . . 9
  - 4.2.1 Flux Spectrum . . . 9
  - 4.2.2 Interstellar Polarisation: finding, adding and/or removing the ISP. . . . 10
- 4.3 Creating $\mathbf{q} - \mathbf{u}$ plots . . . 10

5 Other Submodules **12**
- 5.1 Some Statistics: stat.py . . . 12
  - 5.1.1 Variance, Covariance and Covariance Matrix . . . 12
  - 5.1.2 Principal Components Analysis . . . 12
  - 5.1.3 Pearson Correlation . . . 12
- 5.2 Making Polar Plots: polplot.py . . . 13
  - 5.2.1 Creating the polar axis: polplot.axis() . . . 13
  - 5.2.2 Creating the data to plot: polplot.data() . . . 13

Chapter 1

Introduction

1.1 Overview

FUSS consists of a few IRAF routines written in IRAF CL, and a PYTHON 2.7 package. **make pretty flow chart**

1.2 Python Dependencies

The following packages are required:

- numpy
- math
- scipy (stats and odr)
- matplotlib (pyplot and gridspec)
- pylab
- astropy
- mpl_toolkits

Additionally, there are dependencies within FUSS: e.g the PolData class makes use of some basic fonctions defined within FUSS. Just to be sure, start your script with the following import statments:

```
from FUSS import *
import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt
import numpy as np
import math as m
```

To use datred.lin_specpol() you also need the chromatic zeror-angles for FORS2 whcih can be foud on the website[1]. In my computer I've put it in /home/heloise/FUSS/theta_fors2.txt. The location can be changed at the top of the datred.py file.

If you want to use the data reduction utility functions and spectrpolarimetry calculating functions you will need to load datred.py, e.g:
from FUSS **import** datred as r

If you want PCA and Pearson's test or to use covariance matrix or its elements, need to load stat.py, e.g:
from FUSS **import** stat as s

You should be good to go.

[1] http://www.eso.org/sci/facilities/paranal/instruments/fors/img/eps_theta_2000-01-25.txt

Chapter 2

Data Reduction

The actual data reduction is expected to be done in IRAF, and a few IRAF CL scripts have been written to make this process less tedious. A few functions are defined in datred.py to help set up the data reduction and to calculate the spectropolarimetry from the data exctracted in IRAF. They can be used interactively from a standard (not IRAF xgterm) terminal. The arguments in **bold** are compulsory and the other are optional. All input files have their columns **delimited by a space**

2.1 IRAF Routines and associated PYTHON scripts

The data reduction is principally performed in IRAF and routines have been written in IRAF CL to eliminate some of the tedium. Additionally some of the IRAF scripts call PYTHON scripts which perform tasks that are much more easily automated in PYTHON than in IRAF. Those scripts are also written in PYTHON 2.7 and are expected to be in the same location as the .cl files. You should check in the .cl files that the path to the PYTHON scripts is correct. Note that most of these scripts rely on the naming convention adopted by the datred.sort_red() script.

- **first_calibs.cl**: This script performs the first basic calibrations by removing biases and flats from the science images and arcs. It also combines all arcs images into one.

  Creating the Master Flat in IRAF is interactive. You need to select an aperture or apertures that will cover your science spectra. Then the response function is fitted; the default function for the fit is spline3 and the order is 35. Change the order if needed but best results werre obtained with spline3 (legendre has caused some trouble).

  Another version, called first_calib_no_flat.cl is available. Specrtropolarimetry does not absolutely require flat field removal since polarimetry is a differential effect. I tend to do it because most of the time I'm going to combine the polarised spectra into a flux spectrum eventually. If not then doing flats is just an extra step which can introduce oddities if the response function is not well fitted.

- **arc_apertures.cl**: After defining the apertures on all the science frames (using apall) this creates calibration arc images for each science frame by cutting out the same apertures in the combined arc created by first_calib.cl. It also adds to the headers of the SCIENCE_##.ms.fits images which is the calibration arc image. This imformation will be used by the task identify.

  Uses edit_list.py to update the filenames in one of the lists used by IRAF.

- **create_id.cl**: Its actually just calls the python script create_idcal.py, which copies a template idcal file and modifies it to create idcal files for each science frame.

- **disp2.cl**: This just removes dSCIENCE* files and calls dispcor again.

- **auto_calibrate.cl: GET BACK TO THAT WHEN I'VE USED IT PROPERLY** Uses airmass.py

- **toascii_pol.cl**: Takes data from the dSCIENCE* files and creates text files containing the flux of all individual apertures (2 per science frame), and seperate files for their errors. These can then be used by lin_specpol() to obtain Stokes parameters. MAKE SURE YOU HAVE CREATED A 1D_spectra FOLDER BEFORE USING. Nothing terrible will happen if you don't, just keeps the working folder neats.

  Uses rename_txt.py to rename the text files.

- **toascii_flux.cl**: Same but for the data reduced to produce the flux spectrum. These should be in c_dSCIENCE* files. They should be flux calibrated (unlike the data used for polarisation) and their wavelength bins are smaller (we bin the data used for polarisation to increase S/N)

  Uses rename_flux_txt.py to rename the text files.

You can create permanent tasks by updating the login.cl file: under "package user" define your own tasks e.g:

```
task    $firstcal = myraf$first_calibs.cl
task    $firstcal_noflat = myraf$first_calibs_no_flat.cl
task    $arc_aps = myraf$arc_apertures.cl
task    $idfiles = myraf$create_id.cl
task    $disp2 = myraf$disp2.cl
task    $ascii_pol = myraf$toascii_pol.cl
task    $auto_calib = myraf$auto_calibrate.cl
task    $ascii_f = myraf$toascii_flux.cl
```

Here "myraf" is my environment variabe leading to the location of my iraf scripts.

2.2 Data Reduction Submodule: datred.py

- **sort_red**():

  Description: Sorts and uncompresses the observation files (which MUST end in ".Z", which they should if they are FORS2 datafiles downloaded from the ESO website). Requires astropy.fits to read the headers of the fits files. It also renames files using the naming convention required for use with my .cl and python scripts used for data reduction in IRAF.

  Input format: None.

  Returns: None.

- **info**():

  Description: Creates table containing useful information on the images (taken from the headers). Use in folder containing the uncompressed FITS files.

  Input format: None.

  Output File Format: Filename, ESO label, Retarder Plate Angle, Exposure time,Airmass, Grism, Bin, umber of Pixels, 1/Gain, Read Out Noise, Date.

  Returns: None.

- **hwrpangles**():

  Description: Creates the hwrpangles file required by the lin_specpol and circ_specpol functions to function.

  Requirements: Needs to be called in the directory where the "SCIENCE_##.fits" images are (where ## is a 2 digit number).

  Note: To know what unique string to use to isolate to SN, polarised standard and zero polarisation standard images, have a look in the second column of the info file created by info().

  Optional parameters:

Argument	Type	Description	Default
sn_name	string	String uniue to the ESO name of the SN images	'CCSN'
zeropol_name	string	String uniue to the ESO name of the zero polarisation standard images	'Zero_'
polstd_name	string	String uniue to the ESO name of the polarised standard images. This is the most suseptible to change.	'NGC2024'
return		output_name + " created"	

  Output:
  Linear polarisation: 4 columns (0°, 22.5°, 45°, 67.5°) each containing the number in the name of files corresponding the those angles.
  Circular polarisation: 2 columns (45°, 315°) each containing the number in the name of files corresponding the those angles.

- **lin_specpol**():

  Description: Creates files containing: wl, p, Δp, q, Δq, u, Δu, θ, $\Delta\theta$

  Requirements: Need the file created by hwrpangles in the working directory as well as the text files of the extracted apertures at all retarder plate angles and their assoiated error files.

  Optional parameters:

Argument	Type	Description	Default
oray	string	Aperture correspnding to the ordinary ray: Either 'ap1' or 'ap2'.	'ap2'
hwrpafile	string	file listing image numbers corresponding to sets of HWRP angles	'hwrpangles.txt'
e_min_wl	int	Lower Wavelength cut-off for $\Delta\epsilon$ calculations.	3375
return	Arrays	Wavelengths, p, Δp, q, Δq, u, Δu, θ, $\Delta\theta$, $\Delta\epsilon$ (2D array - One set of values per data set)	

- **circ_specpol**():

  Description: Creates files containing: wl, v, Δv

  Requirements: Need the file created by hwrpangles in the working directory as well as the text files of the extracted apertures at all retarder plate angles and their assoiated error files.

  Optional parameters:

Argument	Type	Description	Default
oray	string	Aperture correspnding to the ordinary ray: Either 'ap1' or 'ap2'.	'ap2'
hwrpafile	string	file listing image numbers corresponding to sets of HWRP angles	'hwrpangles_v.txt'
e_min_wl	int	Lower Wavelength cut-off for ϵ calculations.	3375
return	Arrays	Wavelengths, v, Δv, $\epsilon_{\rm v}$ (2D array - One set of values per data set)	

- **lin_vband**():

  Description: Creates synthetic V band linear polarimetry data from spectropolarimetric data.

  Optional parameters:

Argument	Type	Description	Default
oray	string	Aperture correspnding to the ordinary ray: Either 'ap1' or 'ap2'.	'ap2'
hwrpafile	string	file listing image numbers corresponding to sets of HWRP angles	'hwrpangles_v.txt'
return	float	$p_{\rm V}$, $\Delta p_{\rm V}$, $q_{\rm V}$, $\Delta q_{\rm V}$, $u_{\rm V}$, $\Delta u_{\rm V}$, $\theta_{\rm V}$, $\Delta\theta_{\rm V}$	

- **flux_spectrum**():

  Description: Takes text files ouput by IRAF data reduction which contain the calibrated flux spectra and creates file containing: wl, flux, Δflux

2.3 Data Reduction Procedure

2.3.1 Pre-reduction steps

- Download the data and calibration frames from the FORS2 archive.
- In folder containing the compressed data files use **sort_red()**. Can use interactive python, if you do your working directory at the end of the task will be [start]/FITS/Data_reduction/ where the images are located.
- Use **info()**. Get rid of the calibration frame that don't fit with the data (e.g gain and RON not consistent with the SCIENCE frames).
- In IRAF, check exposure of the calibration, standard and science frames and remove poorly exposed images.
- Make a note of where the spectra are located on the chip (need to know where to place the apertures when doing flats).
- Use **hwrpangles()** to create the necessary files. Create a seperate file for the supernova, zero poalrisation standard and polarised standard.

2.3.2 Data Reduction Steps

General

- firstcal or firstcal_noflat
- noao, imred, specred, epar apall (@list_obj)

- arc_aps
- idfiles. Have template file in database folder already can do from folder above and just use as file name the path: databasidcal_SCIENCE_## (will have to make the python code better so I don't have to move template file to /database).
- identify (@list_cal)
- epar dispcor
- mkdir 1D_spectra

For Flux Spectrum

- epar dispcor (or disp2?) Change w1 but not dw.
- epar sensfunc (can I automate that a bit?)
- auto_calib
- ascii_f
- Need to add rmv_tell to FUSS

For Polarisation Data

- disp2. Change w1 and dw.
- ascii_pol
- In folder use **lin_specpol()**

Chapter 3

Utility Functions and Routines

3.1 Functions

- **get_spctr(filename**, wlmin=0, wlmax=100000, err=False, scale=True**)**

  Description: Imports flux spectrum from an ASCII file. Allows easy cropping (with wlmin and wlmax) and rescaling (using scale).

  File format: Wavelength(Å), Flux, Optional: Flux Error. Each column should be separated by a space and the file should not contain strings because get_spctr() uses numpy.loadtxt() which does not support strings.

Argument	Type	Description	Default
filename	string	Name of the ASCII file where the spectrum is	
wlmin	int/float	Lower wavelength cutoff	0
wlmax	int/float	Upper wavelength cutoff	100,000
err	boolean	If there is an error column, set to True	False
scale	boolean	Multiplies the spectrum (and error) by the median values of the flux.	True
return	arrays	Wavelength, Flux , (optional: Flux error)	

- **get_pol(filename**, wlmin=0, wlmax=100000**)**:

  Description: Imports values from polarisation files (best suited to those given by my specpol routine).

  File format: Required File format: 9 columns, delimited by spaces. First column **must be wavelength** (Å). Other 8 columns for stokes parameters, degree of pol and P.A, and associated errors. (See output format of data reduction script).

Argument	Type	Description	Default
filename	string	Name of the ASCII file where the spectrum is	
wlmin	int/float	Lower wavelength cutoff	0
wlmax	int/float	Upper wavelength cutoff	100,000
return	arrays	Nine 1D arrays containing the data. First array contains wavelengths bins, the others contain the stokes parameters, degree of pol and P.A, and associated errors in the order of the columns in the input ASCII file.	

- **dopcor(val, z)**:

  Description: Applies Doppler correction.

  Input format: val must be an array containing at least 2 columns, with val[0] being the wavelength.

Argument	Type	Description	Default
val	array	Array containing the data. val[0] MUST be the column containing the wavelength bins	-
z	float	Redshift	-
return	array	Array of same size as val but with val[0] being Doppler Corrected.	

- **ylim_def(wl, f**, wlmin, wlmax**)**:

  Description: Finds approriate limits for a plot

Argument	Type	Description	Default
wl	array	Array containing the wavelength bins	-
f	array	Array containing the flux valeus conrresponding to wavelength bins. Must have the same dimensions as wl.	-
wlmin	float/int	Start of wavelength range where min and max are found	4500-
wlmax	float/int	End of wavelength range where min and max are found	9500-
return	float	ymin and ymax	

- **rot_data(q, u, $\theta_{\mathbf{rot}}$)**

  Description: Rotates the data by θ_{rot}. Input in RADIANS.

  return: Rotated q and u, i.e P_d and P_o.

  Note: Obviously this can be used to rotate any 2D data set, not limited to Stokes parameters.

- **norm_ellipse(xc, yc, a, b, $\theta_{\mathbf{rot}}$, n)**

  Description: Creates ellipsoidal data set normally distributed around (xc,yc). The ellipse is described by 2 normal distributions $\mathcal{N}(\mathrm{xc}, \mathrm{a}^2)$ and $\mathcal{N}(\mathrm{yc}, \mathrm{b}^2)$. It is then rotated by θ_{rot} (in RADIANS). The number of data points created is n.

  return: Two arrays containg the x and y values of the created data.

3.2 Interactive Routines

- **ep_date()**

  Description: This allows you to find the epoch with respect a given date (e.g + or - X days with respect to V-band maximum) or a date given an epoch.

  *Options in interactive mode:*Just type the number corresponding to what you want to do.
  1 - Get epoch in days. Inputs: Date of epoch
  2 - Get date for an epoch in days. Inputs: Epoch in days (can be negative)
  3 - Update the V-band max date
  4 - Exit

- **vel()**

  Description: This routine give you the velocity of an element given the observed wavelength and the rest wavelength.

  Note: When asked whether to continue or not (Continue?(y/n):) you can continue by either pressing 'y' then 'Enter' or simply 'Enter'. Any other input will exit the routine.

Chapter 4

The PolData() class

In FUSS each set of polarisation data is treated as one entity. One set of data (e.g for each epoch) must contain the following quantities: the wavelength bins, the stokes parameters, the degree of polarisation, the angle of polarisation, and the errors associated with each of those quantities. Additionally, one can add the flux spectrum (and corresponding wavelength bins, which will probably be different from those associated with the polarisation) and the values for the ISP. Alternatively PolData contains a method (i.e function) to quantify and initialise the ISP, and another the remove the ISP that has been found using **find_isp()** or given with **add_isp()**. Other functions allow you to create $q - u$ plots.

4.1 Core Attributes and Initialisation

Not all attributes are initialised with a values when creating a new instance of PolData. The "core" attributes that make a set of sepctropolarimetric data are given a value when initialising a new instance of PolData. THey are:

- The wavelength bins (self.wlp) – in Å
- The degree of polarisation p (self.p) and the error on p (self.pr) – in %
- The normalised Stokes vector q (self.q) and the error on q (self.qr) – in %
- The normalised Stokes vector u (self.u) and the error on u (self.ur) – in %
- The polarisation angle (self.a) and the associated error (self.ar) – in degrees

There are 9 necessary quantities, which should all be contained within the text files produced by the data reduction process described earlier. You must specify the name of the instance you are creating and the name of the file containing those quantities when initialising. You can also define a wavelength range using wlmin and wlmax. The filename and wavelength limit are passed on to the get_pol() function (see 3.1) during initialisation.

If you are using a file that was not created using FUSS to initialise PolData it must have the following format (if you don't kwo errors just fill the error columns with zeros).

wavelength p Δp q Δq u Δu θ $\Delta\theta$

So to initialise a new instance of PolData just do:

example = **PolData('example', 'filename'**, wlmin, wlmax_data**)**

4.2 Optional attributes and related methods

There are additional aspects of the data that can be considered: The flux spectrum corresponding to a particular data set for example, or the ISP (especially if it needs to be removed).

4.2.1 Flux Spectrum

- **add_flux_data(**self, **flux,** err = False**)**:
- **flu_n_pol(** self, save = False**)**

  Description: Creates plots of p, q, u, θ and f. If the flux spectrum has not been added to a particular instance of PolData the flux spectrum subplot will be plotted, albeit empty.

  Option: The 'save' option is Boolean and determines whether the plot will be saved. By default it is False. If it is set to True the name of the plot will be [self.name]_fnp.png

Argument	Type	Description	Default
flux	Array	Contains 2 or 3 columns separated by a space: [Wavelengths, F, Optional: ΔF]	-
err	Boolean	If True, read in the error on the flux (i.e third) column	False
return		None	

4.2.2 Interstellar Polarisation: finding, adding and/or removing the ISP.

- **find_isp(**self, **wlmin, wlmax)**:

  Description: To find the ISP using the data in the current instance of PolData. It's just an average of q and u within a range given as input which shoud correspond to the region of line blanketting. The associated errors are the standard dev within that range. From q and u, p an θ are then derived as well as their errors. It initialises the values of $q_{\rm isp}, \Delta q_{\rm isp}, u_{\rm isp}, \Delta u_{\rm isp}$ for this particular instance of PolData.

  Input format: The wavelength range limits should be given in Å .

Argument	Type	Description	Default
wlmin	int/float	Lower limit	-
wlmax	int/float	Upper limit	-
return	floats	$q_{\rm isp}, \Delta q_{\rm isp}, u_{\rm isp}, \Delta u_{\rm isp}$	

- **add_isp(**self, **qisp, qispr, uisp, uispr)**:

  Description: Initialises $q_{\rm isp}, \Delta q_{\rm isp}, u_{\rm isp}, \Delta u_{\rm isp}$ from given values.

Argument	Type	Description	Default
$\mathbf{q_{isp}}$	int/float	Normalised Stokes Q contribution from ISP	-
$\mathbf{\Delta q_{isp}}$	int/float	Uncertainty on q	-
$\mathbf{u_{isp}}$	int/float	Normalised Stokes U contribution from ISP	-
$\mathbf{\Delta u_{isp}}$	int/float	Uncertainty on u	-
return		None	

- **rmv_isp(**self):

  Description: Removes ISP from the polarisation data stores in the present isntance of PolData. The previous attributes are **updated** and new attributes containing the original non ISP corected values are created (p0, p0r, q0, q0r, u0, u0r, a0, a0r). Returns None

  Prerequisite: Optional attributes $q_{\rm isp}, \Delta q_{\rm isp}, u_{\rm isp}, \Delta u_{\rm isp}$ MUST have been initialised, either through **add_isp()** or **find_isp()**.

4.3 Creating q − u plots

Before using the following methods it is good to pre-define a figure in matplotlib (e.g f = plt.figure (figsize = (12,10)). One can also use GridSpec (e.g gs1 = gridspec.GridSpec(2,2)). Also you must call **plt.show()** after using the following methods as, although they create the plots, they do not show them automatically (maybe should add an option to do that...). Side note: Even when using hspace = 0 or wspace = 0, the size of the graph has a strong influence ono how much space there will be between the pot, it is just a matter of fine tuning.

The plot-making method is **qu_plt()**. It returns the axis it is plotting on so that you can use that outside of the QUplot instance if you want to add something to the $q - u$ plot (e.g an elipse if you've done principal component analysis and want to show the result).

- **qu_plt(self,** *options*)**

  Description: Creates a $q - u$ plot with a color scale either representing wavelength or velocity.

Argument	Type	Description	Default
subplot_loc	?	3 number integer representing ubplot location OR gridspec locator (e.g gs1[0] if we take the same example given above)	111
wlmin	int/float	Start of wavelength range (Blue cut off)	None
wlmax	int/float	End of wavelength range (Red cut off)	100000

qlim	array	limits of the q axis	[-3, 3]
ulim	array	limits of the u axis	[-3, 3]
textloc	array	location of the text (which is the 'name' given to the QUplot instance when initialising)	[-2.7,-2.7]
cisp	string	Color of the ISP marker	'k'
fs	int	fontsize of the text	16
ls	int	(short for labelsize) size of the numbers on the axis ticks	14
isp	Boolean	Whether to plot ISP or not. $q_{\rm isp}, \Delta q_{\rm isp}, u_{\rm isp}, \Delta u_{\rm isp}$ should have been initialised preamptively.	-
colorbar	Boolean	If True the colorbar is plotted	True
wlrest	int	Rest wavelength of spectral feature of interest. Will change the colour scale from wwavelength to velocity	None
size_clbar	float	Changes the size of the colorbar. Oddly enough seems to sometimes change the size of the plot too??	0.05
line_color	string	If want a solid colour for the lines between the markers. Default gives lines cycling through rainbow colors to match the color of the point they are associated with.	None
marker	string	Type of marker to be used	'.'
lambda_xshift	float	That is to help the placement of the colorbar label. The x position of the label will be qmax + lambda_xshift, where qmax is the upper limit given by qlim.	1.7
fit	Boolean	Whether to plot the dominant axis fit to the data or not. The fit wil always be calculated and the best values returned even if fit is False.	True
qlab_vis	Boolean	Whether to make the q axis label visible	True
ulab_vis	Boolean	Whether to make the u axis label visible	True
qticks_vis	Boolean	Whether to make the q ticks labels visible	True
uticks_vis	Boolean	Whether to make the u ticks labels visible	True
return		The Axis the plot is being plotted on	

Chapter 5

Other Submodules

5.1 Some Statistics: stat.py

This submodule contains a few functions required for routines that are not used as often as the ones previously described. As its name indicates it provides statistics related methods. At present it only contains Principal Component Analyis (PCA) and the Pearson's Test (PT).

5.1.1 Variance, Covariance and Covariance Matrix

PCA and PT willmost likely be performed on the Stokes parameters, so for any point (q, u) on the $q - u$ plane the covariance matrix has the form:

$$C(q,u) = \begin{pmatrix} var(q) & cov(q,u) \\ cov(q,u) & var(u) \end{pmatrix} \tag{5.1}$$

- **cov_el(j, k, q, u, Δq, Δu)**

  Description: Creates the elements of $C(q, u)$. j and k are indicies of the columns and rows, respectively, so that for example $var(q) \equiv C(j = 0, k = 0)$.

  return: Covariance matrix element $C(j, k)$.

- **cov_mat(q, u, Δq, Δu))**

  Description: Creates the covariance matrix of the Stokes parameters (or any 2D data-set I guess but that's what I'm using it for).

  return: Covariance matrix

5.1.2 Principal Components Analysis

- **pca(q, u, Δq, Δu)**

  Description: Performs PCA on data provided.

  return: Axis ration (b/a), rot. angle of major axis ($\theta_{\text{major axis}}$ - degrees), rot. angle of minor axis (degrees)

- **draw_ellipse(q, u, a, axis ratio, $\theta_{\text{major axis}}$)**

  description: Creates ellipse that best fits the data to be drawn onto the $q - u$ plane.

  Note: a is the length of the major axis. Just pick a number that makes the ellipse look nice on top of your data. The axis ratio and $\theta_{\text{major axis}}$ come from doing **pca()**.

  return: The ellipse. To be used as input of the function **add_artist()** from matplotlib.axes.

5.1.3 Pearson Correlation

Pearson correlation allows you to determine whether a linear correlation is likely to exists in a given data set. Pearson's coefficient ρ is close to 1 or -1 if there is linear dependence and close to 0 when if there isn't. It is defined as:

$$\rho = \frac{cov(q,u)}{\sigma_q \sigma_u} \tag{5.2}$$

Where σ is the standard deviation. So from 5.1:

$$\rho = \frac{C(0,1)}{\sqrt{C(0,0)\,C(1,1)}} \tag{5.3}$$

- **pearson(q, u, Δq, Δu)**

  Description: Calculates the Pearson's coefficient ρ for the given data set.

  return ρ

5.2 Making Polar Plots: polplot.py

This submodule contains the functions that will help you create polar plots to represent the SN ejecta.

5.2.1 Creating the polar axis: polplot.axis()

This function creates and returns the polar axis where the data will be plotted.

Pre-requiste: You need to define a figure using matplotlib.pyplot before calling this function as the first argument of axis() is the figure on which you will be plotting the axes.

Argument	Type	Description	Default
fig	matplotlib.figure.Figure	The figure to plot the axis.	-
loc	3 digit int	Location of the subplot (add_subplot() argument)	111
num_ticks	int	*Scales* the number of ticks. Actual number of ticks is: int(maximum velocity/10000)*num_ticks	1
phot_vel	int/float	*Absolute value* of the photospheric velocity	None
vel_lim	Array	Velocity limits [0, max]	[0, 30000]
ang_grid	string	How many lines on angular grid. 'h' for heavy, 'l' for light, 'ul' for ultra-light	'ul'
rad_grid	string	How many lines on radial grid. 'h' for heavy, 'l' for light, 'ul' for ultra-light, or specify where want the velocity grid using a list of velocities (again give absolute values, no negative numbers)	'l'
return		The Axis the plot is being plotted on	

5.2.2 Creating the data to plot: polplot.data()

This functions creates the data points that can then be plotted on the polar axis created using polplot.axis()

Argument	Type	Description	Default
pa_start	int	Angle at start of range (in DEGREES)	-
pa_end	int	Angle at end of range (in DEGREES)	-
vel	int	Velocity	-
return		Two 1D arrays of the same length containing the angular component and radial component to be plotted.	