CORMATpy GUI Documentation

Release 0.0.1

Bruno Viola

Feb 04, 2019

CONTENTS

CHAPTER

ONE

PROJECT SUMMARY

This GUI is meant to be an useful tool for the KG1 responsible officer or anybody else with rights to operate with JET interpherometer related data (i.e. must have rights to write public ppf and access data)

CHAPTER

TWO

TUTORIAL

In this section I will explain how to use the tool and what is possible to achieve.

2.1 Installation process

To run and use the code the user must first install it and all its dependacies.

2.2 Running the code from Terminal

To tun the code:: cd /u/username/work/ python Cormat_main.py -h

> usage: Cormat_main.py [-h] [-d DEBUG]
>
> Run Cormat_main
>
> optional arguments: -h, –help show this help message and exit -d DEBUG, –debug DEBUG Debug level. 0: Info, 1: Warning, 2: Debug, 3: Error; default level is INFO

Alternatively is possible to run the code specifying the debug level to increase verbosity and show debug/warning/error messages.

By default the debug level is **INFO**

Once run an itialization process begins and a database of all the pulses validated so far is created and the documentation is updated.

After the inialization process if finished the user will be prompted with the GUI:

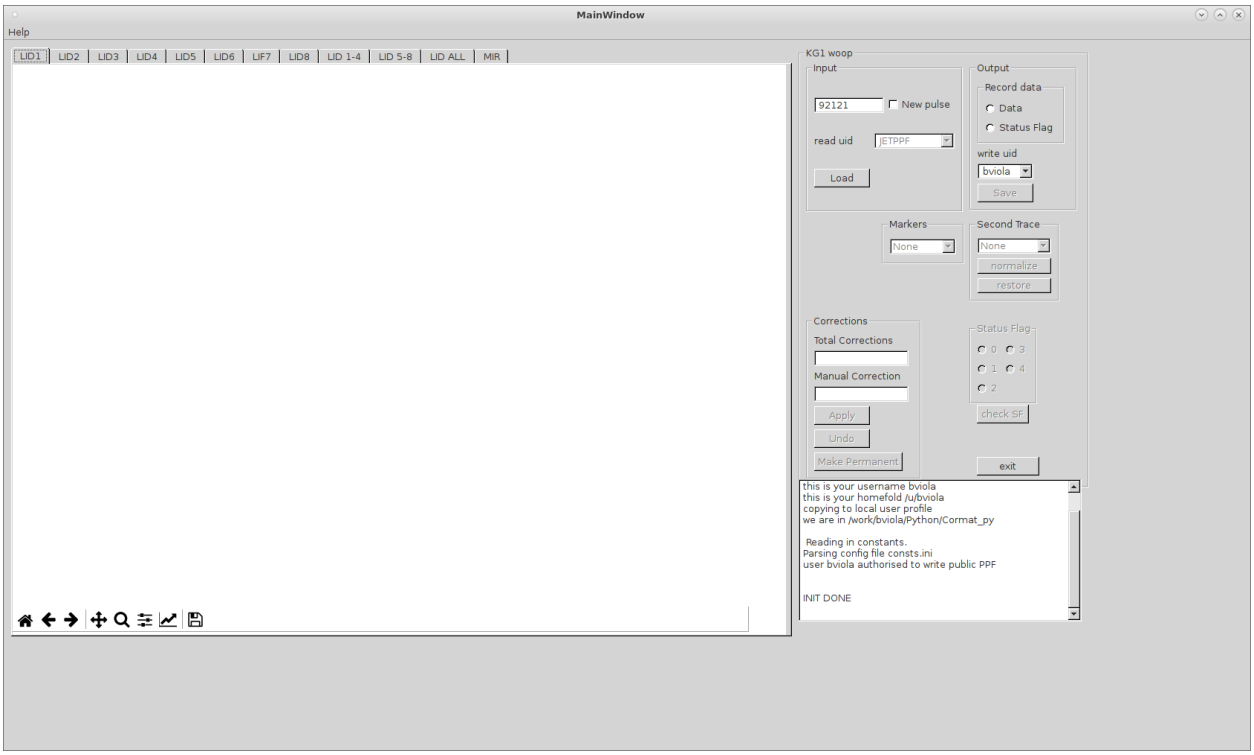

Fig. 2.1: GUI Main Window

CHAPTER

THREE

MAIN CONTROL FUNCTION

3.1 Cormat_main module

class Cormat_main.**woop**(*parent=None*)

Bases: PyQt4.QtGui.QMainWindow, *woop.Ui_CORMAT_py*, *texteditlogger.QPlainTextEditLogger*

Main control function for Woop GUI.

check_status(*button_newpulse*)

check if button "newpulse" is clicked

Parameters **button** –

Returns disable/enable combobox

checkStatuFlags()

reads the list containing pulse status flag :return:

tabSelected(*arg=None*)

function that convert arg number into tab name :param arg: :return:

checkstate(*button*)

connect tab number to LID channel and sets status flag :param button: :return:

set_status_flag_radio(*value*)

load_pickle()

save_to_pickle()

save_kg1()

dump_kg1()

handle_readbutton()

handle_no()

functions that ask to confirm if user wants NOT to proceed

to set read data for selected pulse

handle_yes()

functions that ask to confirm if user wants to proceed

to set read data for selected pulse

readdata()

function that reads data as described in const.ini

reads: - mag data - pellet data - elm data - kg1 data - kg4 data - nbi data - hrts data - lidar data

Returns save data to pickle files: one just for KG1 (pulsename_kg1) and one for everything else (pulsename.pkl)

plot_2nd_trace()
function that plots a second trace in the tabs(each canvas) at the moment clears the canvas and re-plot everything so probably a little slow.

Returns

plot_markers()
function that plots a marker traces in the tabs(each canvas) This version creates sub plot inside the canvas :return:

handle_checkbutton()

handle_savebutton()
save data user can save either Status Flags or ppf (and SF) :return:

handle_save_data_statusflag()

handle_save_statusflag()

handle_normalizebutton()

handle_button_restore()

handle_applybutton()

handle_makepermbutton()

handle_undobutton()

check_current_tab()

handle_help_menu()

handle_pdf_open()

Returns open pdf file of the guide

handle_exit_button()
Exit the application

handle_yes_exit()

class Cormat_main.**MyFormatter**(*fmt=None*, *datefmt=None*, *style='%'*)
Bases: logging.Formatter

class to handle the logging formatting

err_fmt = '%(levelname)-8s %(message)s'

dbg_fmt = '%(levelname)-8s [%(filename)s:%(lineno)d] %(message)s'

info_fmt = '%(levelname)-8s %(message)s'

format(*record*)

Cormat_main.**main**()
Main function

the only input to the GUI is the debug

by default is set to INFO

CHAPTER

FOUR

MODULES

4.1 find_disruption module

Code to read in disruption JPF and check for a disruption. Set the time-dependent status flags for the KG1 signals to 3 around the disruption.

Returns a boolean to say if there was a disruption or not. Also returns two times: [disruption time - disruption window, disruption time + disruption window]

The rest of the KG1 code will not attempt to make any corrections within this time window.

find_disruption.**find_disruption**(*shot_no*, *constants*, *kg1_signals=None*)

Find the disruption time from the JPF disruption signal

Parameters

- **shot_no** – shot number
- **constants** – Instance of Kg1Consts, contains JPF node names and size of time window around disruption to exclude.
- **kg1_signals** – Instance of Kg1Data

Returns Boolean for whether there was a disruption, [start disruption window, end disruption window]

4.2 consts module

Class for reading in and storing kg1 constants, signal names etc.

class consts.**Consts**(*config_name*, *code_version*)

Bases: object

DFR_DCN = 1.143e+19

DFR_MET = 1.876136e+19

MAT11 = 9.088193e+18

MAT12 = -5.536807e+18

MAT21 = 5.754791e-05

MAT22 = -9.445996e-05

CORR_NE = array([-5.60e+18, 9.10e+18, 3.50e+18, 1.46e+19, 2.58e+19, 2.02e+19, -2.10e+18, -7.60e+18, 3.49e+19, 2.37e+19,

CORR_VIB = array([-9.513e-05, 5.770e-05, -3.740e-05, 1.528e-04, 3.438e-04, 2.479e-04, -1.326e-04, -2.277e-04, 4.007e-04, 2.

FJ_DCN = array([0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3])

FJ_MET = array([1, 0, 1, -1, -3, -2, 2, 3, -3, -1, 1, 3, -2, -1, 1, 2])

JXB_FAC = [-1.6e-05, -3e-05, -4.5e-05, -3e-05]

get_phase_node_dcn(*chan*, *sig_type*)
Return the appropriate JPF node name for the DCN phase, given the signal type & channel number

Parameters

- **chan** – Channel number
- **sig_type** – Signal type: kg1r, kg1c_ldraw, kg1c_ld, kg1c_ldcor or kg1v

Returns JPF node name

get_phase_node_met(*chan*, *sig_type*)
Return the appropriate JPF node name for the MET phase, given the signal type & channel number

Parameters

- **chan** – Channel number
- **sig_type** – Signal type: kg1r, kg1c_ldraw, kg1c_ld, kg1c_ldcor or kg1v

Returns JPF node name

get_amp_node_dcn(*chan*, *sig_type*)
Return the appropriate JPF node name for the DCN amplitude signal, given the signal type & channel number

Parameters

- **chan** – Channel number
- **sig_type** – Signal type: kg1r, kg1c_ldraw, kg1c_ld, kg1c_ldcor or kg1v

Returns JPF node name

get_amp_node_met(*chan*, *sig_type*)
Return the appropriate JPF node name for the MET amplitude signal, given the signal type & channel number

Parameters

- **chan** – Channel number
- **sig_type** – Signal type: kg1r, kg1c_ldraw, kg1c_ld, kg1c_ldcor or kg1v

Returns JPF node name

get_sts_node_dcn(*chan*, *sig_type*)
Return the appropriate JPF node name for the DCN KG1C STS signal, given the signal type & channel number

Parameters

- **chan** – Channel number
- **sig_type** – Signal type: kg1c_ldraw, kg1c_ld or kg1c_ldcor

Returns JPF node name

get_sts_node_met(*chan*, *sig_type*)
Return the appropriate JPF node name for the MET KG1C STS signal, given the signal type & channel number

Parameters

- **chan** – Channel number
- **sig_type** – Signal type: kg1c_ldraw, kg1c_ld or kg1c_ldcor

Returns JPF node name

get_fj_node_dcn(*chan*, *sig_type*)
Return the appropriate JPF node name for the DCN KG1C FJ signal, given the signal type & channel number

Parameters

- **chan** – Channel number
- **sig_type** – Signal type: kg1c_ldraw, kg1c_ld or kg1c_ldcor

Returns JPF node name

get_fj_node_met(*chan*, *sig_type*)
Return the appropriate JPF node name for the MET KG1C FJ signal, given the signal type & channel number

Parameters

- **chan** – Channel number
- **sig_type** – Signal type: kg1c_ldraw, kg1c_ld or kg1c_ldcor

Returns JPF node name

set_time_windows(*ip_times*, *nbi_times*, *flattop_times*)
Set time windows

Parameters

- **ip_times** – [start ip, end ip]
- **nbi_times** – [start_nbi, end_nbi]
- **flattop_times** – [start_flat, end_flat]

4.3 library module

library.**norm**(*data*)

library.**normalise**(*signal*, *kg1_signal*, *dis_time*)

library.**get_seq**(*shot_no*, *dda*, *read_uid='JETPPF'*)

library.**get_min_max_seq**(*shot_no*, *dda='KG1V'*, *read_uid='JETPPF'*)

library.**check_SF**(*read_uid*, *pulse*)

library.**extract_history**(*filename*, *outputfile*)
running this script will create a csv file containing a list of all the ppf that have been created with Cormat_py code

the script reads a log file (generally in /u/user/work/Python/Cormat_py)

and writes an output file in the current working directory

the file is formatted in this way shot: {} user: {} date: {} seq: {} by: {} user is the write user id by is the userid of the user of the code the output is appended and there is a check on duplicates

if the user have never run KG1_py code the file will be empty

Parameters

- **filename** – name of KG1L (or KG1H) diary to be read
- **outputfile** – name of the output file

Returns

library.**check_string_in_file**(*filename*, *string*)

Parameters

- **filename** –
- **string** –

Returns checks if the string is in that file

library.**equalsFile**(*firstFile*, *secondFile*, *blocksize=65536*)

library.**copy_changed_kg1_to_save**(*src*, *dst*, *filename*)
src: is the

4.4 ppf_write

Wrapper for opening, writing & closing a PPF.

TO DO: Implement Time-Dependent Status Flags. I don't seem to be able to set tdsf's AND specify an itref in order to use a previous dtype's time-vector.

added output file with log shot,user,date,seq

ppf_write.**check_uid**(*shot_no*, *write_uid*)
Open PPF to check if UID is valid, then abort

Parameters

- **shot_no** – shot number
- **write_uid** – write_uid

Returns 1 if UID is invalid, 0 otherwise

ppf_write.**open_ppf**(*shot_no*, *write_uid*, *comment='CORRECTED KG1 DATA FROM KG1C AND KG1R '*)
Open PPF for writing

Parameters

- **shot_no** – shot number
- **write_uid** – write UID

Returns error code from PPF system. It will be 0 if there is no error.

ppf_write.**write_ppf**(*shot_no*, *dda*, *dtype*, *data*, *time=None*, *comment=None*, *unitd=None*, *unitt=None*, *itref=-1*, *nt=None*, *global_status=None*, *status=None*)
Write PPF DDA/DTYPE

Parameters

- **shot_no** – shot number
- **dda** – DDA

- **dtype** – DTYPE
- **data** – numpy array of data
- **time** – numpy array of time
- **comment** – comment
- **unitd** – units for data
- **unitt** – units for time
- **itref** – reference for timebase
- **nt** – size of the time vector
- **global_status** – status for the DTYPE
- **status** – time-dependent status

Returns error code (0 if everything is OK), itref for timebase written

ppf_write.**close_ppf**(*shot_no*, *write_uid*, *version*)
Close PPF

Parameters **shot_no** – shot number

Returns error code, 0 if everything is OK

4.5 status_flag module

Simple program to access status flags contains a main that creates a database for a given pulse list

status_flag.**find_disruption**(*pulse*)
given a pulse return if it has disrupted (True) or not (False) returns n/a if there is no information about disruptions
:param pulse: :return: boolean

status_flag.**initread**()
initialize ppf :return:

status_flag.**GetSF**(*pulse*, *dda*, *dtype*)

Parameters

- **pulse** –
- **dda** – string e.g. 'kg1v'
- **dtype** – string e.g. 'lid3'

Returns SF := status flag

status_flag.**Getnonvalidatedpulses**(*pulselist*, *dtypelist*, *SF_validated*)

Parameters

- **pulselist** – list of pulses
- **dtypelist** – string e.g. 'lid1','lid2',...
- **SF_validated** – integer rapresenting SF for validated shots

Returns array of integer with pulse numbers,status flag list

status_flag.**GETfringejumps**(*pulse*, *FJC_dtypelist*)

Parameters

- **pulse** – pulse number
- **FJC_dtypelist** – string e.g. 'FC1','FC2',...

Returns array of integer, rapresenting fringe jumps corrections

status_flag.**main**(*pulse1*, *pulse2*, *FJthres*, *outputfilename*)

:param pulse1:intial pulse :param pulse2: final pulse :param FJthres: threshold to be used to check number of fringe jumps in pulse :param outputfilename: output database name :return: database containing all pulses inside the given interval whose median of fringe jumps exceed the given threshold and marks if there was a disruption or not

status_flag.**printdict**(*goodpulses_sorted*)

4.6 wv_denoise module

Module containg wv_densoise, a function to filter a signal using wavelet filtering. Determination of threshold from ncoeff and percent is done as in the idl function wv_denoise.pro.

wv_denoise.**wv_denoise**(*signal*, *family=None*, *nlevels=None*, *ncoeff=None*, *percent=None*)

Function to filter a signal using wavelet filtering. Determination of threshold from ncoeff and percent is done as in the idl function wv_denoise.pro. For more details on wavelet options see PyWavelet docs. If neither coeff or percent are specified then no filtering is done.

TO DO:

•Implement soft threshold

•Thresholding using variance or something?

•Think and check: divisible by 2 thing

•2D filter?

Parameters

- **signal** – signal to be filtered.
- **family** – Wavelet family and order to use (default is 'db1')
- **nlevels** – Number of levels of DWT to perform. If none is given then decomposition upto dwt_max_level is done, which is the maximum useful level as computed by pyWavelets.
- **ncoeff** – The number of coefficients to retain when filtering. If specified then percent is ignored.
- **percent** – The percentage of coefficients to retain when filtering. If specified then coeff is ignored.

Returns the filtered signal

4.7 wv_get_background module

Module for determining the background in a signal. Approximate implementation of the method described in Galloway et al. (2009), An iterative algorithm for background removal in spectroscopy by wavelet transforms,), which finds the background using an iterative wavelet filtering method .Applied Spectroscopy, 63, 1370

Depends on wv_denoise

It's a bit slow...

wv_get_background.**recursive_wv**(*data*, *ind_times*, *currentiter=0*, *niter=10*, *nlevels=12*)
Recursively find background of the data

Parameters

- **data** – data
- **ind_times** – time indices of background regions
- **currentiter** – current iteration number
- **niter** – total number of iterations
- **nlevels** – number of levels for the wavelet filtering

wv_get_background.**wv_get_background**(*time*, *data*, *start_time*, *end_time*, *nlevels=12*)
Use wavelet filtering to determine the background for a data set

Parameters

- **time** – array of time values
- **data** – array of data values
- **start_time** – time before which signal can be considered to be background
- **end_time** – time after which signal can be considered to be background
- **nlevels** – number of levels to use for the wavelet filtering

Returns background

CHAPTER

FIVE

CLASSES

5.1 SignalBase

Class for reading and storing a signal from the PPF or JPF system, with functionality for filtering, resampling, and calculating the differences between adjacent time points.

signal_base.**decimate_ZP**(*x*, *q*, *n=None*, *ftype='iir'*, *axis=-1*, *zero_phase=False*)

Downsample the signal by using a filter. By default, an order 8 Chebyshev type I filter is used. A 30 point FIR filter with hamming window is used if *ftype* is 'fir'. Parameters ———- x : ndarray

The signal to be downsampled, as an N-dimensional array.

q [int] The downsampling factor.

n [int, optional] The order of the filter (1 less than the length for 'fir').

ftype [str {'iir', 'fir'}, optional] The type of the lowpass filter.

axis [int, optional] The axis along which to decimate.

zero_phase [bool] Prevent phase shift by filtering with `filtfilt` instead of `lfilter`.

y [ndarray] The down-sampled signal.

resample

The `zero_phase` keyword was added in 0.17.0. The possibility to use instances of `lti` as `ftype` was added in 0.17.0.

signal_base.**gcd**(*a*, *b*)

Compute the greatest common divisor of a and b

signal_base.**lcm**(*a*, *b*)

Compute the lowest common multiple of a and b

class signal_base.**SignalBase**(*constants*)

Bases: `object`

read_data_ppf(*dda*, *dtype*, *shot_no*, *read_bad=False*, *read_uid='JETPPF'*, *seq=0*)

Read in and store PPF data :param dda: DDA :param dtype: DTYPE :param shot_no: shot number :param read_bad: If set to true, data is read in for all sequence numbers (even status == 4) :param read_uid: UID to use to read PPF data :param seq: sequence number to read in

read_data_jpf(*signal_name*, *shot_no*, *use_64bit=False*)

Read in and store JPF data :param signal_name: Node name for JPF :param shot_no: shot number :param use_64bit: If set to true the data is stored as 64 bit float

read_data_jpf_1D(*signal_name*, *shot_no*)
Read in JPF data with only one dimension :param signal_name: signal name :param shot_no: shot number

filter_signal(*family*, *ncoeff=None*, *percent=None*, *start_time=0*, *end_time=0*)
Filter the signal using wavelet filtering :param family: wavelet family to use for filtering :param ncoeff: number of coefficients to retain in the filtering :param percent: percentage of coefficients to retain in the filtering :param start_time: Time from which to start the filtering :param end_time: Time to finish the filtering :return: numpy array containing the filtered data from start_time - end_time

get_time_inds(*start_time*, *end_time*)
Get the index of the times corresponding to start_time and end_time :param start_time: start time :param end_time: end time :return: index of start_time, index of end_time

resample_signal(*resample_method*, *new_time*)
Resample the signal, to a different timebase, by -interpolation, -zeropadding :param resample_method: method to use, only "interp" implemented atm. :return: numpy array of resampled data

get_differences(*npoints*)
Get the difference between the data npoints apart. :param npoints: Number of points over which to calculate the difference :return: numpy array of difference

get_second_differences(*npoints*)
Get the second differential over npoints :param npoints: Number of points over which to calculate the difference :return: numpy array of second differential

delete_points(*ind_points*)
Delete points with indices ind_points :param ind_points: indices of points to delete

5.2 SignalAmp

Class for reading and storing KG1 amplitude signals.

Inherits from SignalBase (signal_base.py).

Additional functionality for finding bad points, and checking if the amplitude is valid in general

class signal_amp.**SignalAmp**(*constants*)
Bases: *signal_base.SignalBase*

KG1C_START_AMP_BAD = 0.2

KG1R_START_AMP_BAD = 3500

KG1V_START_AMP_BAD = 0.5

KG1R_TIME_AMP_CHECK = 38.5

KG1V_TIME_AMP_CHECK = 32.0

read_data_ppf(*dda*, *dtype*, *shot_no*, *signal_type*, *dcn_or_met*)
Overide read_data_ppf method, to include additional argument to set the signal type & dcn_or_met.

Parameters

- **dda** – DDA
- **dtype** – DTYPE
- **shot_no** – shot number
- **signal_type** – String to indicate if this is KG1R, KG1C or KG1V
- **dcn_or_met** – "dcn" or "met" depending on whether signal is from DCN or MET laser

read_data_jpf(*signal_name*, *shot_no*, *signal_type*, *dcn_or_met*)
Overide read_data_jpf method, to include additional argument to set the signal type and check the amplitude of the signal

Parameters

- **signal_name** – JPF signal name
- **shot_no** – shot number
- **signal_type** – Identifies whether data is KG1C or KG1R or KG1V
- **dcn_or_met** – "dcn" or "met", ie. signal is from DCN or MET laser

Returns 0: Amplitude was read in and is OK, 9: Error reading the JPF signal, 10: The amplitude is bad

find_bad_points()
Find points with a bad amplitude.

•For KG1C, CPRB should be within a valid range

•For KG1R & KG1V the amplitude should be above a certain value

:return indices of bad points

_check_amp_average(*time*, *threshold*)
Check the average amplitude at the start of the pulse is high enough. For use with KG1R or KG1V

Returns good_amp : True if the amplitude is good, false otherwise

_check_amp_kg1c()
Check the CPRB signal (the KG1C frequency, which acts as the amplitude here). The CPRB signal at the start of the pulse should be within 80% of cprb_mid. It should also not fall below this value by more than cprb_range too many times in the whole pulse. cprb_mid and cprb_range are different for DCN and MET lasers.

Returns good_amp : True if the amplitude is good, false otherwise

5.3 Kg1PPFData

Class to read and store KG1 PPF data for one channel. Reads in LIDX, FCX, MIRX, JXBX, TYPX Needs modifying so it would work with old KG1V & new KG1V dtypes

class kg1_ppf_data.**Kg1PPFData**(*constants*, *pulse*)
Bases: object

read_data(*shot_no*, *read_uid='JETPPF'*)
Read in PPF data for KG1V for a given channel :param shot_no: shot number :param chan: channel :param read_uid: read UID :return: True if data was read in successfully, False otherwise

set_status(*lid*, *new_status*, *time=None*, *index=None*)
Set time-dependent status flags for lid. If neither time or index are given, set status flags for all time points :param lid: LID number to set status for :param new_status: status to be set :param time: time, or time range in which to set status. :param index: index, or index range in which to set status.

uncorrect_fj(*corr*, *index*)
Uncorrect a fringe jump by corr, from the time corresponding to index onwards. Not used ATM. Will need more testing if we want to use it... Suspect isclose is wrong.

Parameters

- **corr** – Correction to add to the data
- **index** – Index from which to make the correction

correct_fj(*corr*, *time=None*, *index=None*, *store=True*, *correct_type=''*, *corr_dcn=None*, *corr_met=None*)

Shifts all data from time onwards, or index onwards, down by corr. Either time or index must be specified

Parameters

- **corr** – The correction to be subtracted
- **time** – The time from which to make the correction (if this is specified index is ignored)
- **index** – The index from which to make the correction
- **store** – To record the correction set to True
- **correct_type** – String describing which part of the code made the correction
- **corr_dcn** – Only for use with lateral channels. Stores the correction, in terms of the number of FJ in DCN laser (as opposed to in the combined density)
- **corr_met** – Only for use with lateral channels. Stores the correction, in terms of the number of FJ in the MET laser (as opposed to the correction in the vibration)

get_coord(*shot_no*)

Get vacuum vessel temperature & extract spatial coordinates of KG4 chords from text file. Function copied from A. Boboc's kg4r_py code.

Parameters **shot_no** – shot number

get_jpf_point(*shot_no*, *node*)

Get a single value from the JPF ie. Convert Nord data to real number Function copied from A. Boboc's kg4r_py code.

Parameters

- **shot_no** – shot number
- **node** – JPF node

5.4 ElmsData

Class to read Be-II signals, and detect ELMs.

The disruption time can be specified, in which case only ELMs before this time will be detected

Method used for ELM detection:

- Only use the Be-II signal up until the disruption time, if there is a disruption. This ensures that the filtering doesn't give us extra unwanted oscillations due to the large signal at the time of the disruption.
- Find the background using wavelet filtering (module wv_get_background).
- Find ELMs by studying the first derivative of the Be-II signal. We are looking for a positive derivative, followed by a negative derivative. Different thresholds can be set for the positive & negative derivative to account for the shape of the ELM signals (sharp rise, followed by shallower fall off).

Future improvements:

- Finding the background using wavelets is a bit slow: try a moving average

- a variable threshold. Sometimes the threshold is too low/high, resulting in incorrectly detected ELMs or missing ELMs.

class elms_data.**ElmsData**(*constants*, *shot_no*, *dis_time=0.0*)
Bases: object

Class to read Be-II signals, and detect ELMs.

WV_FAMILY = 'db15'

WV_PERC = 99.0

START_TIME = 40.0

END_TIME = 65.0

UP_THRESH = 0.5

DOWN_THRESH = -0.3

ELM_WIDTH_MAX = 0.1

BE_START_TIME = 35.0

_find_elms(*shot_no*, *dis_time*)
Read in ELMs signal and find elms

Parameters

- **shot_no** – shot number
- **dis_time** – Disruption time. Set to zero for no disruption. ELMs will only be detected before this time.

5.5 HRTSData

Class to read and store all hrts data

class hrts_data.**HRTSData**(*constants*)
Bases: object

read_data(*shot_no*, *read_uid='JETPPF'*)
Read in HRTX data

Parameters **shot_no** – shot number

5.6 LIDARData

Class to read and store all lidar data

class lidar_data.**LIDARData**(*constants*)
Bases: object

read_data(*shot_no*, *read_uid='JETPPF'*)
Read in lidar (LIDX)

Parameters **shot_no** – shot number

5.7 Kg4Data

Class to read and store all kg4 data

class kg4_data.**Kg4Data**(*constants*)
Bases: object

CIB = 51.6

MIN_FAR = 0.02

read_data(*mag*, *shot_no*)
Read in faraday angle & ellipticity, and convert to densities

Parameters

- **mag** – Instance of MagData, with data read in already. Needed for conversion to density
- **shot_no** – shot number

5.8 MagData

Class to read and store magnetics data

class mag_data.**MagData**(*constants*)
Bases: object

MIN_IP = 0.3

PER_IP_FLAT = 0.8

CBVAC = 5.1892e-05

MIN_BVAC = -1.5

read_data(*shot_no*)
Read in magnetics data

Parameters **shot_no** – shot number

Returns True if data was read successfully and there is ip False otherwise.

_find_ip_times()
Find the start and end time of the ip and the flat-top.

Returns False if there is no IP True otherwise

_find_bvac_times()
Find the start and end time of the Bvac > 1.5T

Returns False if there is no Bvac True otherwise

5.9 NBIData

Class to read in NBI data and store start & end of NBI power

class nbi_data.**NBIData**(*constants*)
Bases: object

NBI_MIN_POWER = 3.5

read_data(*shot_no*)
Read in nbi data

Parameters **shot_no** – shot number

5.10 PelletData

Class to read and store time of pellets.

Expecting to use PL/PTRK-ANA<PKM signal, which has a data point per pellet, the value of which is the mass of the pellet in mg, measured using a microwave cavity.

class pellet_data.**PelletData**(*constants*)
Bases: object

PELLET_THRESHOLD = 4.0

read_data(*shot_no*)
Read in pellets data

Parameters **shot_no** – Shot number

5.11 Canvas

class canvas.**Canvas**(*parent=None*)
Bases: matplotlib.backends.backend_qt4agg.FigureCanvasQTAgg

5.12 QPlainTextEditLogger

class texteditlogger.**QPlainTextEditLogger**(*parent*)
Bases: logging.Handler

class that defines a handler to write logging message inside the GUI the geometry and position of the TextEdit is defined here, not by QT designer

emit(*record*)

5.13 woop

woop.**_fromUtf8**(*s*)

woop.**_translate**(*context*, *text*, *disambig*)

class woop.**Ui_CORMAT_py**
Bases: object

setupUi(*CORMAT_py*)

retranslateUi(*CORMAT_py*)

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

Symbols

B

C

D

E

F

G

S

T

U

W