CORMATpy GUI Documentation

Release 2.2.3

Bruno Viola

Mar 05, 2020

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.

The code is stored in a git repository

from terminal

>> git clone git@git.ccfe.ac.uk:bviola/Cormat_py.git -b master /your/folder

Once this is done the user needs to checks if they have access to two Python libraries used at JET: ppf and getdata

From v2.0.0 this code works only with python > 3.7 and PyQt5. to swith to python3.7

From the Terminal

>> module unload python

>> module load python/3.7

Or you can create your own virtualenviroment running

>> python -v venv venv

From the Python console, run >> pip install -r repuirements.txt –user

to update the python packages to meet the requirements of the application.

Once this is donem the application is ready to be used

>> python Cormat_main.py

The code can run with different levels of verbosity. The defaul version is set to INFO (level = 0) Alternatively is possible to run the code specifying the debug level to increase verbosity and show debug/warning/error messages.

The user can change it

>> python Cormat_main.py -d level_name (with level_name an integer among 0,1,2,3,4)

Level 4 is the maximum level of verbosity possible and it is generally used for debugging purposes.

For a more comprehensive list of the command that the user can use:

>> python Cormat_main.py -h

will run an help

2.2 Running the code from Terminal

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

usage: Cormat_main.py [-h] [-d DEBUG] [-doc DOCUMENTATION] [-r RESTORE_CHANNEL] [-c CHANNEL] [-e ERASE_DATA]

Run CORMAT_py

optional arguments:

-h, --help show this help message and exit

-d DEBUG, --debug DEBUG Debug level. 0: Info, 1: Warning, 2: Debug, 3: Error, 4: Debug Plus; default level is INFO

-doc DOCUMENTATION, –documentation DOCUMENTATION Make documentation. yes/no

-r RESTORE_CHANNEL, --restore_channel RESTORE_CHANNEL restore channel to original value. yes/no

-c CHANNEL, --channel CHANNEL channel to be restored

-e ERASE_DATA, --erase_data ERASE_DATA remove stored data from scratch and saved folders. yes/no

Once run an itialization process begins:

The code will check if is running in an 64bit enviroment with Python3.7 It will also check what user is running the code and if it has proper authorisation to write KG1 public ppfs. If not, the user will be able to run the code but he/she will be limited in writing only private ppfs.

If the user has chosen so, during the initialisation process also the documentation can be produced (updated).

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

Window size will be scaled to dimension of screen resolution. It is advisable to scale a bit the dimension of the window to read messages on terminal even though all messages will be displayed as well inside the GUI.

2.3 Principles of operation

After chosing a pulse

The user can use the drop down menu in Figure to choose the read_uid.

The list that is shown here contains all user that are authorised to write KG1 public ppf. After clicking the load button, the code start running according to:

After loading a pulse the user now will have to choose a pulse to validate.

All figure windows come with a navigation toolbar, which can be used to navigate through the data set. Here is a description of each of the buttons at the bottom of the toolbar

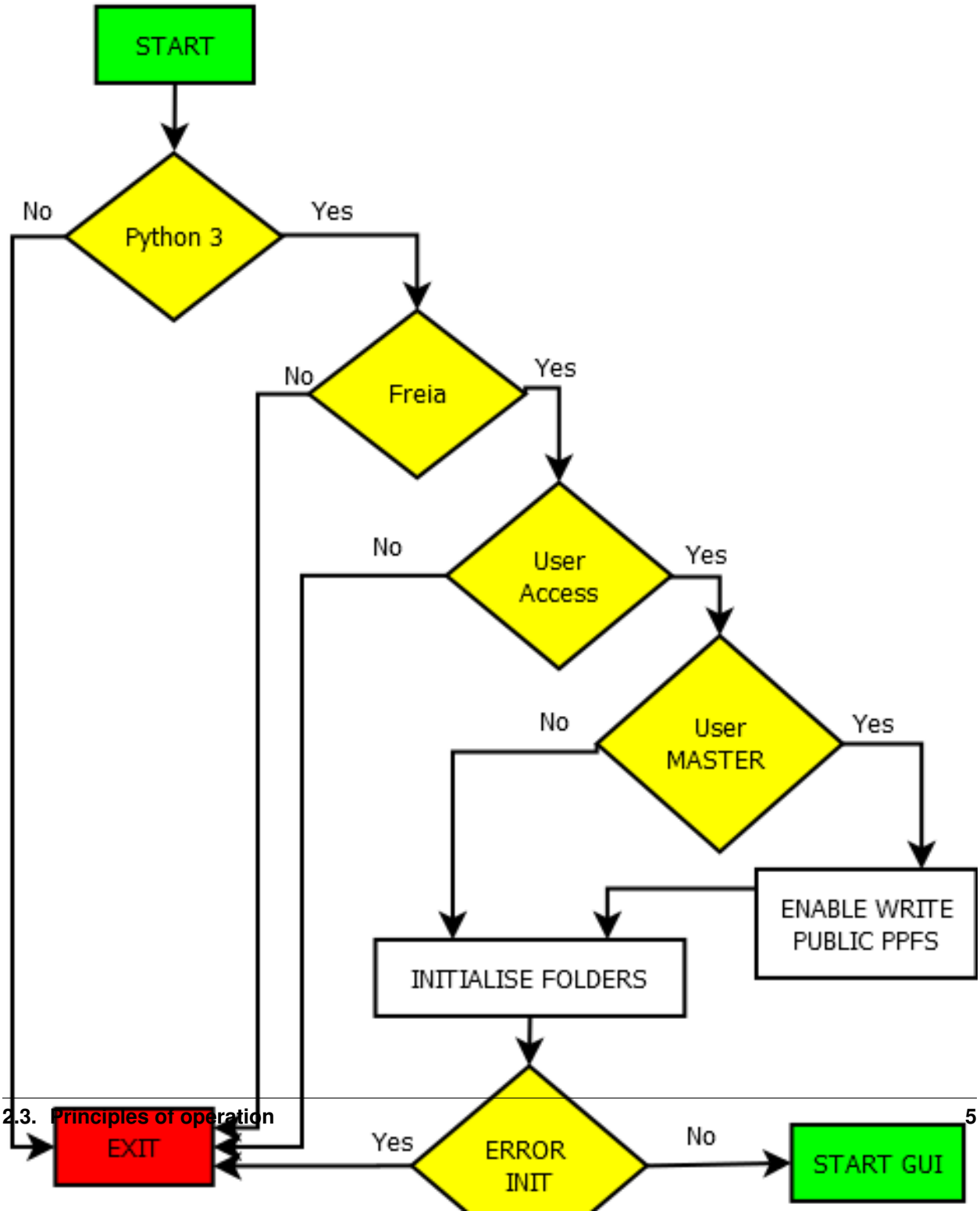
The Home, Forward and Back buttons

These are akin to a web browser's home, forward and back controls. Forward and Back are used to navigate back and forth between previously defined views. They have no meaning unless you have already navigated somewhere else using the pan and zoom buttons. This is analogous to trying to click Back on your web browser before visiting a new page or Forward before you have gone back to a page – nothing happens. Home always takes you to the first, default view of your data. Again, all of these buttons should feel very familiar to any user of a web browser.

The Pan/Zoom button

CORMAT_PY INITIALISATION

Author: A.Boboc
Version: V.1.0
Date: 31/01/2019



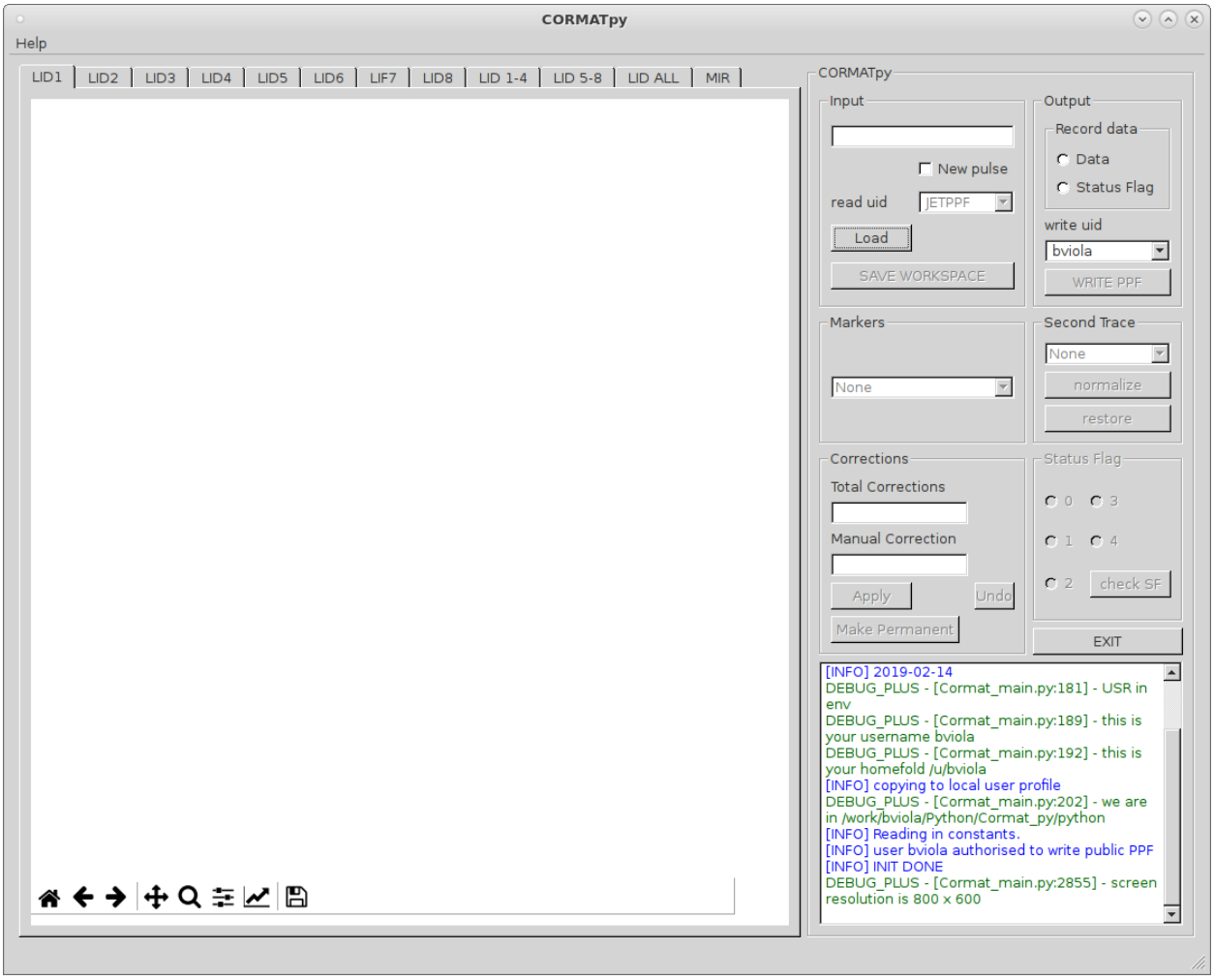


Fig. 2: Main GUI

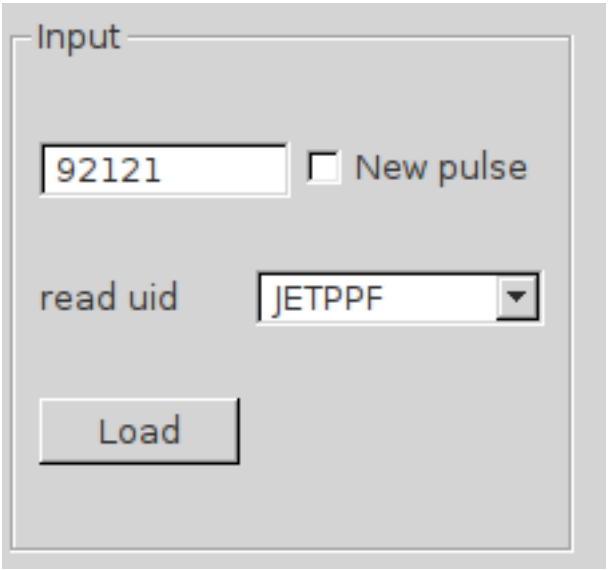


Fig. 3: Pulse selection

Fig. 4: list of authorised users (to write KG1 ppfs)

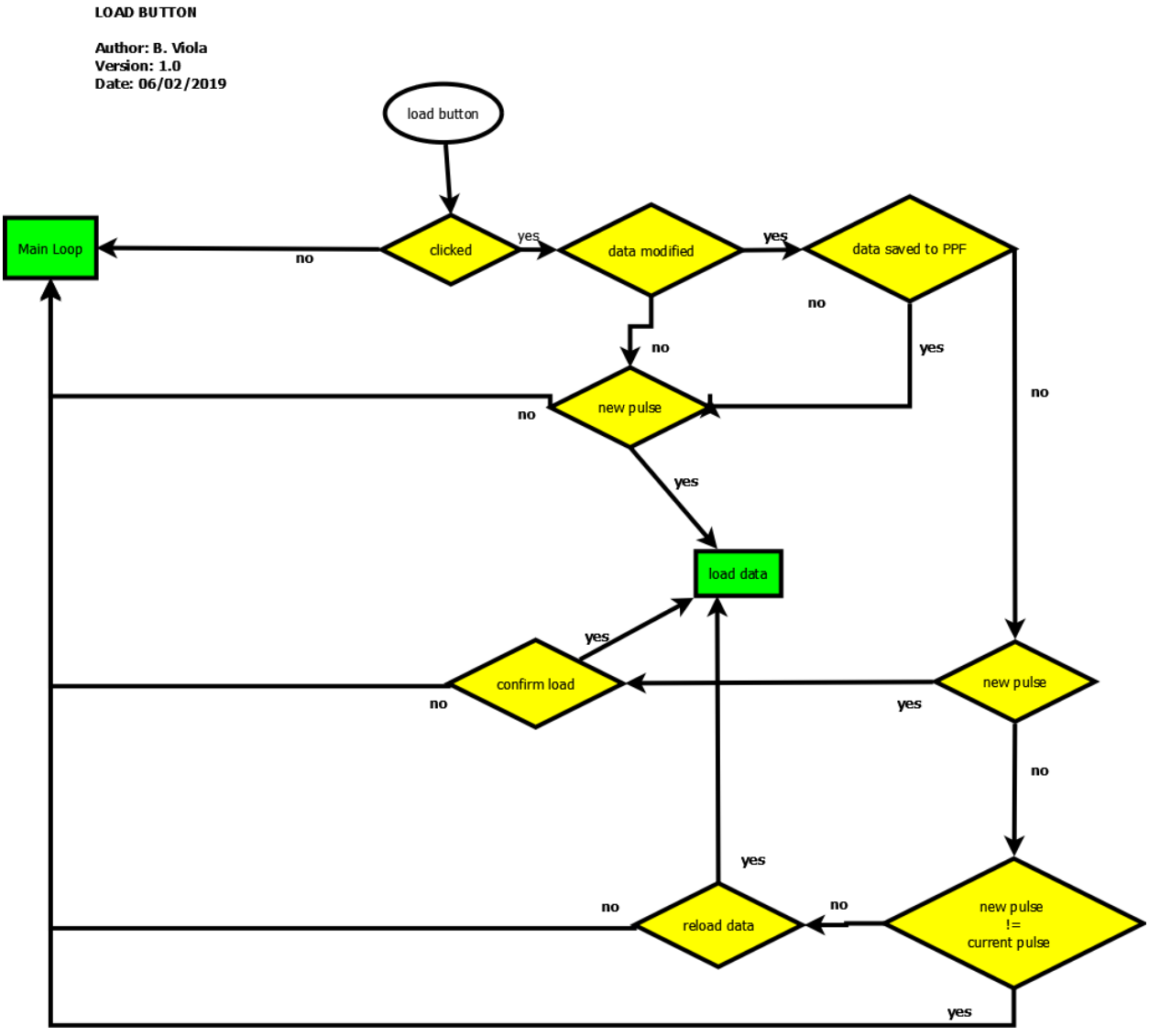


Fig. 5: Load button logic

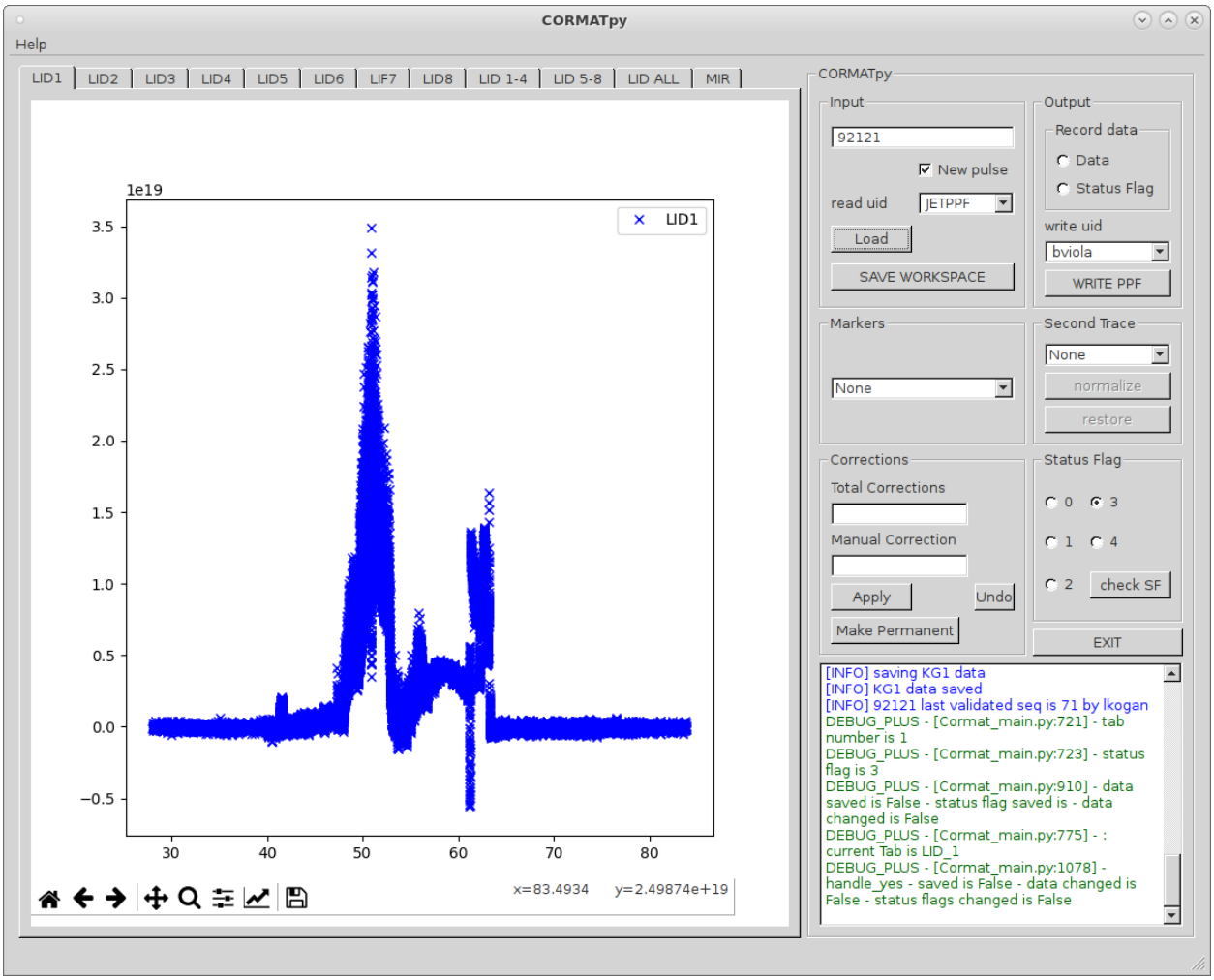


Fig. 6: Pulse selection

Fig. 7: Navigation toolbar

Fig. 8: Home, Forward and Back buttons

Fig. 9: Zoom button

This button has two modes: pan and zoom. Click the toolbar button to activate panning and zooming, then put your mouse somewhere over an axes. Press the left mouse button and hold it to pan the figure, dragging it to a new position. When you release it, the data under the point where you pressed will be moved to the point where you released. If you press 'x' or 'y' while panning the motion will be constrained to the x or y axis, respectively. Press the right mouse button to zoom, dragging it to a new position. The x axis will be zoomed in proportionately to the rightward movement and zoomed out proportionately to the leftward movement. The same is true for the y axis and up/down motions. The point under your mouse when you begin the zoom remains stationary, allowing you to zoom in or out around that point as much as you wish. You can use the modifier keys 'x', 'y' or 'CONTROL' to constrain the zoom to the x axis, the y axis, or aspect ratio preserve, respectively.

With polar plots, the pan and zoom functionality behaves differently. The radius axis labels can be dragged using the left mouse button. The radius scale can be zoomed in and out using the right mouse button.

The Zoom-to-rectangle button

Fig. 10: Zoom to rectangle

Click this toolbar button to activate this mode. Put your mouse somewhere over an axes and press a mouse button. Define a rectangular region by dragging the mouse while holding the button to a new location. When using the left mouse button, the axes view limits will be zoomed to the defined region. When using the right mouse button, the axes view limits will be zoomed out, placing the original axes in the defined region.

The Subplot-configuration button

Fig. 11: subplot preferences

Use this tool to configure the appearance of the subplot: you can stretch or compress the left, right, top, or bottom side of the subplot, or the space between the rows or space between the columns.

The Save button

Click this button to launch a file save dialog. You can save files with the following extensions: png, ps, eps, svg and pdf.

Action by user to make corrections are carried out by hitting a key and making mouse clicks.

The keyboard hits establish a "mode" with subsequent mouse click(s) expected. An overview of valid key strokes is given in the following table.

Fig. 12: save button

Key | Purpose	
.	single correction
M	multiple correction
N	neutralise corrections/s
S	suggest correction
T	zero LID (tail) data
Z	zero LID (interval) data

For further details see relative paragraph on topic/s.

Note that any action, valid or not, triggers a clear response, i.e. either the final result expected (user performed action correctly) or an error message with be displayed.

There is NO option so far to excape from an unwanted mode: the event has to finish.

2.4 How to define the Time value(s) of correction(s)

Before the value of a fringe jump correction in unit of fringes can be entered at a dedicated prompt in a Widget, the associated time point(s) have to be defined and, thus, the right correction mode must be raised.

2.4.1 Single correction

Hit the key **[.]** and then click between the two data points in question with the **right** mouse button. The vertical position of the pointer is unimportant.

A widget will be prompeted and the user will have to click the correction to be applied in fringe units

after selecting the correction to be applied, and pressing **apply** the canvas will be updated and a magenta line will appear on the selected data point.

the user will be promped with a widged asking if he/she wants to confirm that correction or, instead, apply the displayed suggested correction.

and then with another widget asking to confirm the correction and mark it as **permanent**, which means the correction will be stored.

2.4.2 Multiple correction

Hit the key **[M]** and you will be in the multicorrection mode.

You will have to click as many time points as you like and you then select in the widget the chosen correction to apply and that will be applied to all selected data points.

Note that the sign of a correction is given by the jump (+/-) to be removed.

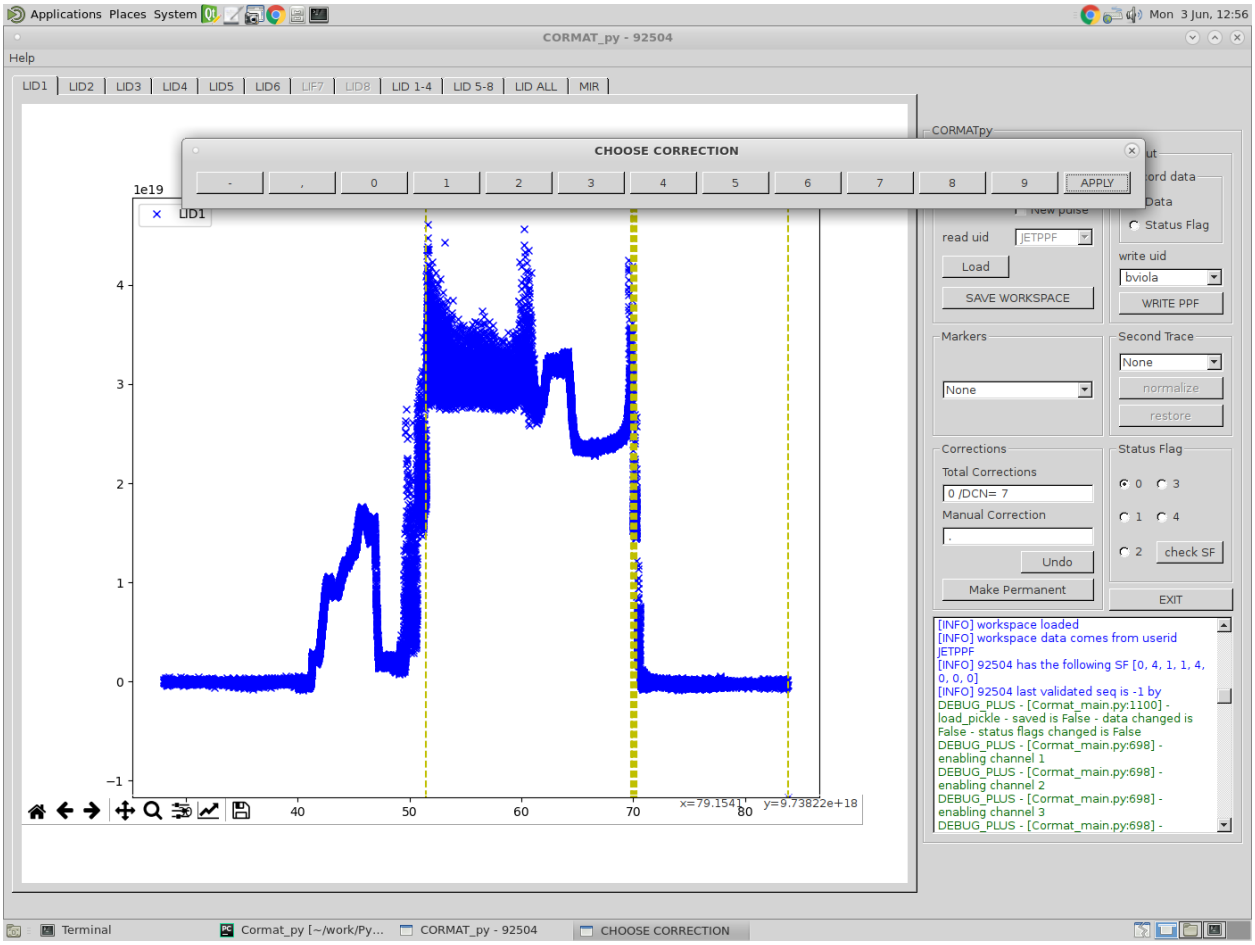


Fig. 13: select fringe correction to be applied

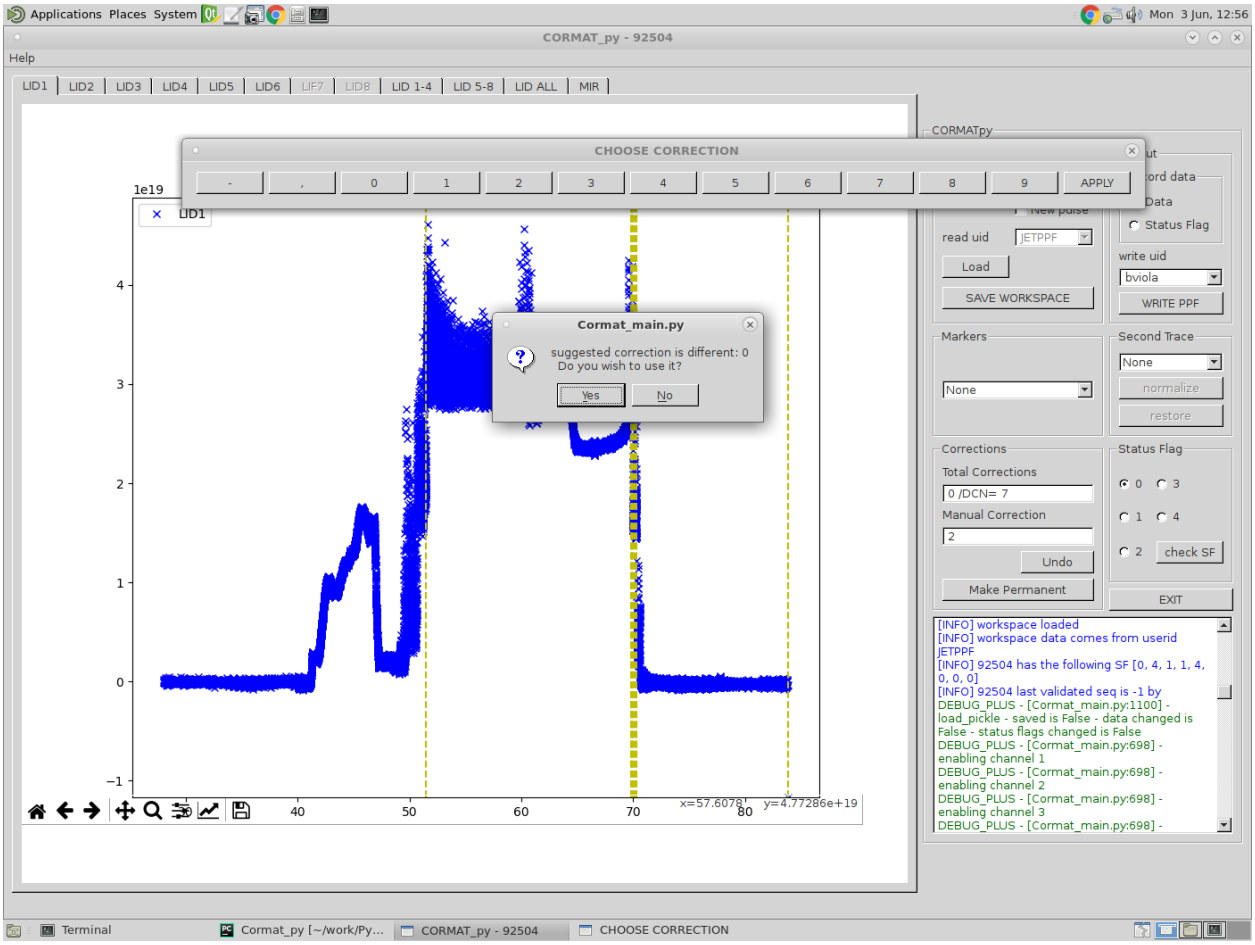


Fig. 14: select fringe correction to be applied

2.4.3 Neutralise correction(s)

Hit the key **[N]** and you will be in the neutralisation mode.

All corrections manual and/or intershot (made by hardware) can be removed by generating the corresponding neutralising new correction, all at once.

In order to do so:

1) make sure there are no pending corrections (press mark permanent button)
2) Hit the key **[N]** to invoke this mode
3) define time window by two mouse right clicks

First all manual corrections within the chose timw window will be neutralised and then intershot corrections.

2.4.4 Suggest correction(s)

Hit the key **[S]** and then click between the two data points in question with the **right** mouse button. The vertical position of the pointer is very important to get a precise response.

A widget will be prompeted and the user will have to click on the **[APPLY]** button. A message will be displayed telling the user what is the suggested correction to apply based on the distance between the point at the left of the click and the next one.

It is possible to tell the code to give a suggestion based on the distance between the point at the left of the click and X point on the right by selecting a number on the widget that appers once the event has started.

2.4.5 Zeroing LID data

Sometimes, expecially in the case of a disruption, it is desirable to bring everything down to zero for the uninteresting rest of the discharge (zeroing the tail) or just in a certain finite interval of very bad data.

To do so:

1) make sure there are no pending corrections (press mark permanent button).
2) Hit the key **[T]** to invoke zeroing mode of the tail.
3) define time window by one mouse right click.

or 2a) Hit the key **[Z]** to invoke zeroing mode of a selected interval. 3a)define time window by two mouse right clicks.

Permanent corrections will be generated and take effect such that:

i) all data points t>t1 or in the interval t1 < t < t2 are put as close to zero as possible and
ii) the sum of corrections is 0 if zeroing of the tail was invoked. Otherwise, the sum of corrections remains unchanged.

2.4.6 Undo correction(s)

All corrections can be undo if they have not been made permanent by clicking the **undo** button. If user want to undo corrections already marked as permanet he/she shall use the neutralise mode by selecting the time interval where the correction(s) to be undo lay.

corrections_1z

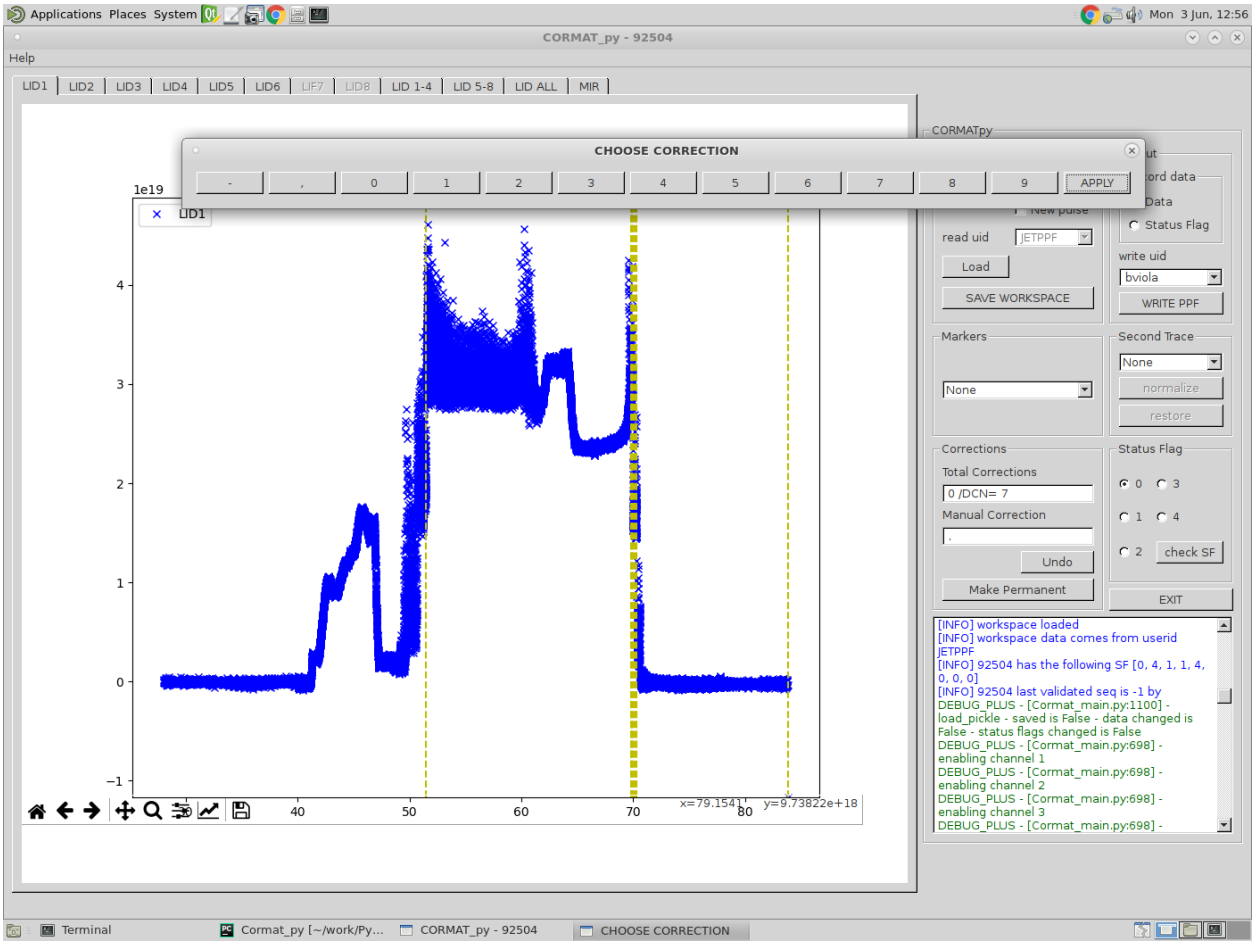


Fig. 15: select fringe correction to be applied

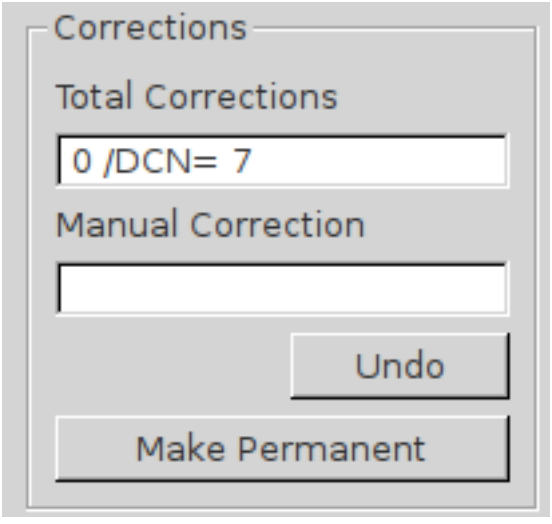


Fig. 16: Ttoal number of corrections, undo/mark permanent button

2.5 Plotting additional signals and plasma markers

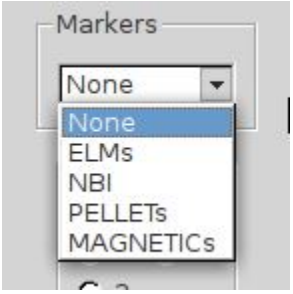


Fig. 17: Combo box selection for plasma marker

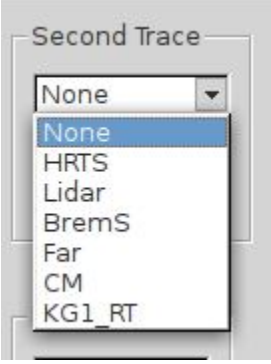


Fig. 18: Combo box selection for second trace

2.6 How to set the status flag of a channel

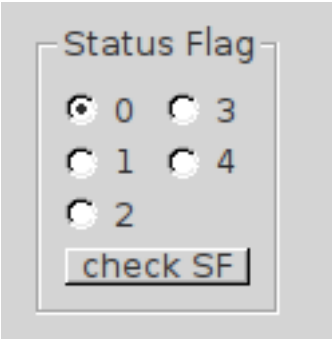


Fig. 19: change status flag to channel

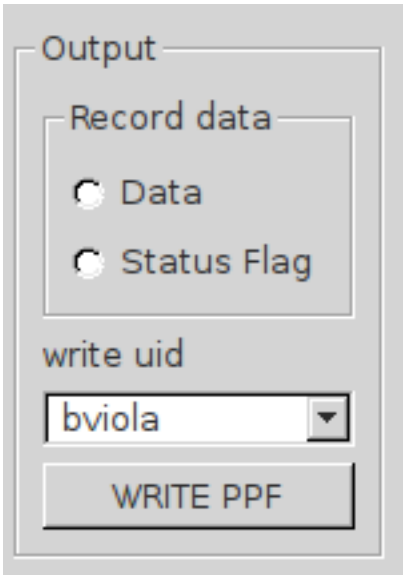
When the user is confident that the channel validation is complete can select a value for the status flag to apply to it.

The values for the System status flags are:

Value	Meaning
0	Data is unchecked - the default value.
1	Data is of above normal quality.
2	Data is of normal quality.
3	Data is below normal quality.
4	Data is unreliable or incorrect.

You will not normally be able to read any data that has been given a System Status Flag value of 4.

2.7 Saving data and storing Status flags



Once the validation request is complete the user can save to ppf the data.

They can choose to save privately or in a public ppf. Save just status flag (only available if data has not changed), or save both data and status flag.

At the moment of writing this guide v2.1.2 of the code still allow the user to downsample the data, suggested pratice if saving to public ppf.

This is due to still on going compatibility checks of KG1 data with the rest of the chain.

CHAPTER

THREE

MAIN CONTROL FUNCTION

3.1 Cormat_main module

Class that runs CORMAT_py GUI

Cormat_main.**myself**()

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

Bases: PyQt5.QtWidgets.QMainWindow, *CORMAT_GUI.Ui_CORMAT_py*, *custom_formatters.QPlainTextEditLogger*

Main control function for CORMATpy GUI.

handle_readbutton_master()

implemets what happen when clicking the load button on the GUI the logic is explained in the FLowDiagram ../FlowDiagram/load_button_new.dia

checkStatuFlags()

reads the list containing pulse status flag :return:

canvasselected(*arg=None*)

function that convert arg number into tab name it also sets the SF value when changing tabs :param arg: index of canvas :return:

setcoord(*chan*, *reset=False*)

connects selected tab to its own list of selected data points

Parameters **reset** –

Returns

checkstate(*button*)

connect tab number to LID channel and sets status flag it also performs a check if the user clicked a radio button to change status flag for current tab/channel :param button: :return:

set_status_flag_radio(*value*)

converts status flag integer value into boolean to check SF radio buttons in GUI :param value: status flag to be applied to selected channel :return:

load_scratch()

reloads data dumped to scratch

Returns

load_pickle(*kg1only=False*)

loads last saved data from non saved operations data are saved in pickle format (binary)

also to be used when reloading a pulse

save_to_pickle(*folder*)
saves to pickle experimental data :param folder: for now user can save either to saved or scratch folder :return:

save_kg1(*folder*)
module that saves just kg1 data to folder :param folder: :return:

dump_kg1()
temporary save kg1 data to scratch folder :return:

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

handle_yes_reload()
module that handles readloading already downloaded data :return:

readdata()
function that reads data as described in const.ini :return: True if success

False if error saves data to pickle files: one just for KG1 (kg1_data) and one for everything else (data.pkl)

plot_data()
handles widgets initialises canvas to blank sets grid and axes plots KG1 data

Returns

GUI_refresh()
updates GUI after reading data enables/disables buttons

connects tab to tabselected signal

finally run a check on status flag applied to selected channel :return:

update_channel(*chan*)
after a correction is applied this module updates kg1 data to new values and replots data on all tabs :param chan: :return:

set_xlimits(*lower*, *upper*)
Convenience method to canvas.axes.set_xlim as matplotlib autoupdate is bugged

Parameters

- **lower** – lower limit of data
- **upper** – upper limit of data

Returns

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_check_status(*button_newpulse*)
function used for debugging and control purposes - - checks if button "newpulse" is clicked :param button_newpulse: :return: disable/enable combobox

handle_saveppfbutton()
save data

user will save either Status Flags or ppf (and SF)

if user selects in GUI to save ppf a new PPF sequence will be written for dda='KG1V' if user selects in GUI to save SF only the SF will be written in the last sequence (no new sequence)

if data has not been modified it automatically switches to save status flags only

handle_save_data_statusflag()
function data handles the writing of a new PPF

Returns 67 if there has been an error in writing status flag 0 otherwise (success)

writeppf()
function that writes to ppf the KG1 data :return:

handle_save_statusflag()
function that uses ppfssf inside ppf library (ATTENTION this function is not listed yet in the ppf library python documentation (only C++ version) :return: 0 if success

68 if fails to write status flags

handle_normalizebutton()
function that normalises the second trace to KG1 for comparing purposes during validation and fringe correction :return:

handle_button_restore()
function that restores signals to original amplitude :return:

handle_makepermbutton()
function that makes permanent last correction store data into FC dtype only

Returns

check_current_tab()
returns current tab :return: current tab

keyPressEvent(*event*)
keyboard events that enable actions are:

. starts single correction mode m stats multiple correction mode starts zeroing n starts neutralisation mode

Parameters **event** – keyboard event

Returns

handle_help_menu()
opens firefox browser to read HTML documentation :return:

handle_pdf_open()

Returns opens pdf file of the guide

multiplecorrections()
this module applies multiple corrections (same fringe correction) to selected channel

Returns

changezerotail(*chan*, *lid*, *vib*, *index*)
this function enables the user to change the start time of zeroing of the tail of the data

it just restores previous data using

Parameters

- **lid** – density backup
- **vib** – vibration backup
- **index** – index of previous zeroing point

Returns

zeroingtail()
this module zeroes correction on selected channel

Permanent corrections will be generated and take effect such that all data points in the interval t>t1 are put as close to zero as possible and the sum of corrections is 0.

if user is unsatisfied it can be called again to amend previous tail point

red line will appear on all 8 tabs to show starting point of zeroed data

Returns

unzerotail()
once user has set a point to zero tail of data. action can be undo

This module restores data before zeroing of tail was invoked

works only after a tail event

Returns

remove_corrections_while_zeroing(*chan*, *index_start*, *index_stop=None*)
function that deals with removing the corrections (permanent and non) from the interval where zeroing is being applyed

data is dumped to a pickle file for easy restore in case the zeroing will be undo by user

Parameters

- **index_start** – starting index of zeroing
- **index_stop** – ending index of zeroing (if None is end of data)

Returns deletes corrections inside given interval

zeroinginterval()
this module zeroes data on selected channel inside a chosen interval

Permanent corrections will be generated and take effect such that all data points in the interval t2<t>t1 are put as close to zero as possible and the sum of corrections is 0.

it is possible to zero multiple interval

undo button click will undo just last action (i.e. last zeroed interval)

Returns

unzeroinginterval()
once user has zeroed an interval of data. action can be undo

works only after a zeroing interval event

Returns

neutralisatecorrections()
module that neutralises permanent corretions

first the code will process manual corrections not made permanent yet

then the code will process automatic corrections produced by hardware and stored inside KG1V/FCx

Returns

suggestcorrection()
module that suggests correction to apply on selected point :return:

suggest_corrections()
module that suggests correction to apply on selected point to be used for suggest correction events :return:

getcorrectionpointwidget()
action to perform when the single correction signal is emitted: each widget (canvas) is connected to the function that gets data point from canvas

and shows widget to apply correction :return:

getmultiplecorrectionpointswidget()
action to perform when the single correction signal is emitted: each widget (canvas) is connected to the function that gets data point from canvas

and shows widget to apply correction :return:

disconnnet_multiplecorrectionpointswidget()
action to perform when the single correction signal is emitted: each widget (canvas) is connected to the function that gets data point from canvas

and shows widget to apply correction :return:

zeroing_correction()
when zeroing add correction equal to total correction up to that moment to have total number of correction = 0 :param self: :return:

gettotalcorrections()
function that computes the total number of correction in a channel :return: sum of correction (vertical channels) sum of correction on density / vibration (lateral channels)

singlecorrection()
this module applies single correction to selected channel

Returns

get_point()
each tab as its own vector where input data points are stored in a list user later by correction modules to get where to apply corrections :return:

get_multiple_points()
each tab as its own vector where input data points are stored in a list user later by correction modules to get where to apply corrections :return:

discard_single_point()
after a single correction is applied this function allows the user to undo it

before it is permanently applied :return:

discard_neutralise_corrections()
after correction are neutralised this function allows the user to undo it

before it is permanently applied

Returns

discard_multiple_points()
after multiple corrections are applied this function allows the user to undo it

before it are permanently applied

Returns

which_tab()
function used to detect which tab is currently active and set ax and widget

Returns ax and widget name of current tab

show_kb()
function that shows or hide the widget used to select corrections so far max/min correction are +/- 5 fringes :return:

init_read()
function that runs initialization of the GUI each time a new pulse is loaded :return:

handle_exit_button()
Exit the application

static handle_yes_exit()
close application ::todo: check why using this function doens't allow to use profiling (what exit values does it need?)

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+1

CORR_VIB = array([-9.513e-05, 5.770e-05, -3.740e-05, 1.528e-04, 3.438e-04, 2.479e-04,

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.**test_logger**()

library.**find_duplicate_w_index**(*l*)

Parameters **l** – list

Returns value and indexes

library.**reconnect**(*signal*, *newhandler=None*, *oldhandler=None*)
reconnect a signal to a new handler after disconnection from old handler :param signal: :param newhandler: :param oldhandler: :return:

library.**is_empty**(*any_structure*)
check if structure is empty :param any_structure: :return:

library.**are_eq**(*a*, *b*)
checks if a and b are equal :param a: :param b: :return:

library.**autoscale_data**(*ax*, *data*)

library.**find_nearest**(*array*, *value*)

Parameters

- **array** –
- **value** –

Returns returns value and index of the closest element in the array to the value

library.**find_in_list_array**(*array*, *value*)

library.**order**(*lst*)
checks if a list is ordered in descending or ascending way :param lst: :return:

library.**find_listelements_in_otherlist2**(*list1*, *list2*, *tstep*)

Parameters

- **list1** –
- **list2** –
- **tstep** – minimum distance between two data points

Returns

library.**find_within_range**(*array*, *minvalue*, *maxvalue*)
returns elements of array within two values :param array: :param minvalue: :param maxvalue: :return:

library.**pyqt_set_trace**()
Set a tracepoint in the Python debugger that works with Qt

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

library.**normalise**(*signal*, *kg1_signal*, *dis_time*, *xlim1*, *xlim2*)

Parameters

- **signal** – second trace
- **kg1_signal** – KG1 signal
- **dis_time** – disruption time

Returns Use ratio of maximum of signal - kg1 as the normalisation factor. Exclude region around the disruption.

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

Parameters

- **shot_no** – pulse number
- **dda** –
- **read_uid** –

Returns get sequence of a ppf

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

Parameters

- **shot_no** –
- **dda** –
- **read_uid** –

Returns return min and max sequence for given pulse, dda and readuid

min is the unvalidated sequence max is the last validated sequence

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

Parameters

- **read_uid** –
- **pulse** –

Returns list of Status Flags

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*)

Parameters

- **firstFile** –
- **secondFile** –
- **blocksize** –

Returns returns True if files are the same,i.e. secondFile has same checksum as first

library.**copy_changed_kg1_to_save**(*src*, *dst*, *filename*)

Parameters

- **src** –
- **dst** –
- **filename** –

Returns copies file from src folder to dst

library.**delete_files_in_folder**(*folder*)

library.**on_xlims_change**(*ax*)

library.**on_ylims_change**(*ax*)

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=None*)

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 :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 Signalkg1

Class for reading and storing KG1 signals.

Inherits from SignalBase (signal_base.py).

Additional functionality for correcting fringe jumps and storing status flags

class signal_kg1.**SignalKg1**(*constants*, *shot_no*)
Bases: *signal_base.SignalBase*

uncorrect_fj(*corr*, *index*, *fringe_vib=None*)
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. 07mar2019 used this function instead of is close as there is an issue with types and the value we are looking for sometimes are not found

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*, *corr_dcn=None*, *corr_met=None*, *lid=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
- **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)

5.3 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

5.4 Kg1PPFData

Class to read and store KG1 PPF data for one channel. Reads in LIDX, FCX, MIRX, JXBX, TYPX

class kg1_ppf_data.**Kg1PPFData**(*constants*, *pulse*, *sequence*)
Bases: *signal_base.SignalBase*

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.

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

5.5 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.

5.6 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

5.7 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

5.8 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

5.9 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

5.10 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

5.11 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

5.12 Canvas

class canvas.**Canvas**(*parent=None*)
Bases: matplotlib.backends.backend_qt5agg.FigureCanvasQTAgg

CLASS used to convert widget into a matplotlib figure

contains mouse event (right click) that returns (xs,yx) when click is in axes

signal

5.13 SupportClasses

class support_classes.**MyLocator**
Bases: matplotlib.ticker.AutoLocator

view_limits(*vmin*, *vmax*)
Select a scale for the range from vmin to vmax.

Subclasses should override this method to change locator behaviour.

class support_classes.**LineEdit**
Bases: PyQt5.QtWidgets.QLineEdit

inherit from QLineEdit we use super so that child classes that may be using cooperative multiple inheritance will call the correct next parent class function in the Method Resolution Order (MRO).

uses mouse press event and emit signal when clicked

signal_evoke_kb

mousePressEvent(*self*, *QMouseEvent*)

class support_classes.**Key**(*name*, *event*, *receiver*)
Bases: PyQt5.QtWidgets.QPushButton

class support_classes.**KeyBoard**(*receiver*)
Bases: PyQt5.QtWidgets.QWidget

Keyboard class is a new widget that pops up when the QlineEdit is clicked to show keys that can be pressed to write in the QlineEdit defining the corrections the users wants to apply

apply_pressed_signal

key_pressed()

apply_pressed()

keyPressEvent(*self*, *QKeyEvent*)

5.14 Formatters

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

class to handle the logging formatting

PURPLE = '\x1b[95m'

CYAN = '\x1b[96m'

DARKCYAN = '\x1b[36m'

BLUE = '\x1b[94m'

GREEN = '\x1b[92m'

YELLOW = '\x1b[93m'

RED = '\x1b[91m'

BOLD = '\x1b[1m'

UNDERLINE = '\x1b[4m'

END = '\x1b[0m'

err_fmt = '[\x1b[91m%(levelname)-5s\x1b[0m] \x1b[91m%(message)s\x1b[0m'

dbg_fmt = '[\x1b[36m%(levelname)-4s\x1b[0m] [\x1b[36m%(filename)s\x1b[0m:\x1b[36m%(lin

dbgplus_fmt = '[\x1b[92m%(levelname)-8s\x1b[0m] (\x1b[92m%(filename)s:%(lineno)d\x1b[0

info_fmt = '[\x1b[94m%(levelname)-4s\x1b[0m] \x1b[94m%(message)s\x1b[0m'

warn_fmt = '[\x1b[93m%(levelname)-7s\x1b[0m] \x1b[93m%(message)s\x1b[0m'

format(*record*)
Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

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

class that defines a handler to write logging message inside the GUI

emit(*record*)
Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplementedError.

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

FORMATS = {5: ('%(levelname)s - [%(filename)s:%(lineno)d] - %(message)s', <PyQt5.QtGu

format(*record*)
Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

5.15 CORMAT_GUI

class CORMAT_GUI.**Ui_CORMAT_py**
Bases: object

setupUi(*CORMAT_py*)

retranslateUi(*CORMAT_py*)

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX