BEC Monitor Documentation

Release 0.0.1

Zachary Glassman

June 04, 2015

This software is used in the Lett Lab of the Laser Cooling and Trapping group at NIST/JQI for experiments on Spinor Na Bose-Einstein condensates.

CHAPTER

ONE

USAGE

Install the required packages

- numpy
- scipy
- pyqtgraph
- pyqt
- lmfit
- pandas

To use just run SpinorMonitor.py

Contents:

1.1 Fitobject

Contents:

1.1.1 fit_object

class `Fitobject.`**`fit_object`**(*index*, *params*, *type_of_fit*, *roi*, *data*)

fit object holds all the information for a single fit_sequence

Parameters

- **`index`** (*string*) – Shot number
- **`params`** (*dictionary*) – dictionary of Parameters objects containing fit parameters
- **`type_of_fit`** (*string*) – type of fit to be performed
- **`roi`** (*list*) – region of interest to crop data for fit
- **`data`** (*numpy array*) – numpy array of image to be analyzed

Methods

BEC_num(*scalex*, *scaley*)

get number of BEC atoms from fit from equation

$$N = \left(\frac{2\pi}{3\lambda^2}\right)\frac{2\pi A}{5}R_xR_y$$

Parameters

- **scalex** – x scale of pixel
- **scaley** – y scale of pixel

Variables

- **A** – fitted Thomas-Fermi amplitude
- **Rx** – fitted Thomas-Fermi x radius
- **Ry** – fitted Thomas-Fermi y radius
- **sigma** – optical density

Returns atom number

BEC_num_1(*scalex*, *scaley*, *A*, *dx*, *dy*)

helper function for BEC num get number of BEC atoms from fit from equation

$$N = \left(\frac{2\pi}{3\lambda^2}\right)\frac{2\pi A}{5}R_xR_y$$

Parameters

- **scalex** – x scale of pixel
- **scaley** – y scale of pixel
- **A** – fitted Thomas-Fermi amplitude
- **dx** – fitted Thomas-Fermi x radius
- **dy** – fitted Thomas-Fermi y radius

Variables **sigma** – optical density

Returns atom number

TF_2D()

two dimensional Thomas Fermi which is not normalized of the form:

$$TF = A\max\left\{\left[1-\left(\frac{x_c}{dx}\right)^2-\left(\frac{y_c}{dy}\right)^2\right],0\right\}^{3/2}$$

Variables

- **x0** – absolute x center
- **y0** – absolute y center
- **xc** – rotated x center
- **yc** – rotated y center
- **theta** – angle relative to x axis

- **A** – amplitude
- **dx** – Thomas-Fermi radius on rotated x axis
- **dy** – Thomas-Fermi radius on rotated y axis
- **off** – offsett

Therm_num(*scalex*, *scaley*)
get number of BEC atoms from fit from equation

$$N = \left(\frac{2\pi}{3\lambda^2}\right)\frac{2\pi A}{5}R_x R_y$$

Parameters

- **scalex** – x scale of pixel
- **scaley** – y scale of pixel

Variables

- **A** – fitted Gaussian amplitude
- **Rx** – fitted Gaussian x standard deviation
- **Ry** – fitted Gaussian y standard deviation
- **sigma** – optical density

Returns atom number

bimod2min(*params*)
function to minimize, need to subtract offset since included in both terms

create_vecs(*roi*)
create vectors scaled by pixel size

Parameters **roi** (*list*) – region of interest list

Return X x vector from meshgrid

Return Y y vector from meshgrid

fit_image()
fit corrected image with parameters from params

gauss_2D()
two dimensional Gaussian which is not normalized of the form:

$$G = A\exp\left(-\frac{(x-x_c)^2}{2dx^2} - \frac{(y-y_c)^2}{2dy^2}\right) + Off$$

Variables

- **x0** – absolute x center
- **y0** – absolute y center
- **xc** – rotated x center
- **yc** – rotated y center
- **theta** – angle relative to x axis
- **A** – amplitude
- **dx** – standard deviation on rotated x axis

- **dy** – standard deviation on rotated y axis
- **off** – offsett

get_angled_line(*x0*, *y0*, *theta*)
get angled line for angle theta with formulas

$$x_c = (x - x_0)\cos(\theta) - (y - y_0)\sin(\theta)$$
$$y_c = (x - x_0)\sin(\theta) - (y - y_0)\cos(\theta)$$

Parameters

- **x0** – absolute x center
- **y0** – absolute y center
- **xc** – rotated x center
- **yc** – rotated y center
- **theta** – angle relative to x axis

line_profile()
calculate line profile, with zeroes to make full image

Returns two-dimensional array which has padding outside of the region of

interest and can be summed for profiles.

multiple_fits()
function to fit sequentially with input defined from SpinorMonitor we may need to take parameters of previous fit!! do fit, update values, do next fit

partial_TF_2D(*xc*, *yc*, *A*, *dx*, *dy*)
two dimensional non-rotated Thomas Fermi which is not normalized of the form:

$$TF = A\max\left\{\left[1 - \left(\frac{x_c}{dx}\right)^2 - \left(\frac{y_c}{dy}\right)^2\right], 0\right\}^{3/2}$$

Parameters

- **xc** – absolute x center
- **yc** – absolute y center
- **A** – amplitude
- **dx** – Thomas-Fermi radius on rotated x axis
- **dy** – Thomas-Fermi radius on rotated y axis
- **off** – offsett

process_results(*scalex*, *scaley*)
process results of fit and allow output return dictonary scale with the appropriate pixel values after fit

sg2min(*params*)
stern gerlach function to minimize

stern_gerlach_2D()
2 dimensional three thomas fermi distributions

subtract_background()
Subtract background from image looking at first and last 20 rows of the inital image far away from experiment

1.2 Datatablewidget

Contents:

1.2.1 DataTable

class `Datatablewidget.`**`DataTable`**(*parent=None*)

tabbed tables to show system parameters and fitted parameters

Methods

`update_pandas_table`(*df*)

update tables, check if cols are different

1.3 Auxfuncwidget

Contents:

1.3.1 AuxillaryFunctionContainerWidget

class `Auxfuncwidget.`**`AuxillaryFunctionContainerWidget`**(*parent=None*)

class for displaying container of auxillary function widgets will hold a stacked layout of all auxillary functions

Methods

`add_element`(*name*)

convenicne function to create function widget and add to proper dictionaries

`re_import`()

1.3.2 AuxillaryFunctionWidget

class `Auxfuncwidget.`**`AuxillaryFunctionWidget`**(*func*, *parent=None*)

class holding function and entry information

Methods

`calculate`()

`generate_info_widgets`()

generate info sublayouts

`generate_params_widgets`()

generate parameter sublayout and return layout

`get_params`()

1.4 Ipython

Contents:

1.4.1 PlotObj

class `Ipython.`**`PlotObj`**

class to hold SpinorPlot objects

Methods

`add_plot`(*plot*, *name*)

Add plot to dictionary of plots to update

`update`(*var_dict*)

update all plots in dictionary

1.4.2 QIPythonWidget

class `Ipython.`**`QIPythonWidget`**(*customBanner=None*, **args*, ***kwargs*)

Convenience class for a live IPython console widget. This widget lives within the main GUI

Attributes

custom_control	
custom_page_control	

Methods

`clearTerminal`()

Clears the terminal

`executeCommand`(*command*)

Execute a command in the frame of the console widget

`printText`(*text*)

Prints some plain text to the console

`pushVariables`(*variableDict*)

Given a dictionary containing name / value pairs, push those variables to the IPython console widget

1.4.3 QIPythonWidgetContainer

class `Ipython.`**`QIPythonWidgetContainer`**(*parent=None*)

Ipython container class for multi-threading

Methods

1.4.4 SpinorPlot

class `Ipython.`**`SpinorPlot`**(*func*, *name=None*, *xaxis=None*, *yaxis=None*)
class to plot with updating stuff

Methods

`get_vars`(*var_dict*)

`set_axis`()

`update_plot`(*var_dict*)

1.5 Auxwidgets

Contents:

1.5.1 FingerTabBarWidget

class `Auxwidgets.`**`FingerTabBarWidget`**(*parent=None*, **args*, ***kwargs*)
Class to implement tabbed browsing for options

Methods

`paintEvent`(*event*)

`tabSizeHint`(*index*)

1.5.2 TextBox

class `Auxwidgets.`**`TextBox`**
custom textbox, mostly QTextEdit, with some added functions

Methods

`output`(*x*)

1.6 Subroutines

Contents:

1.7 Dataplots

Contents:

1.7.1 ImageWindow

class `Dataplots.`**`ImageWindow`**(*parent=None*)

Image View with custom ROI

Attributes

lastFileDir	

Methods

`add_lines`(*results*)

add lines to plot, input it numpy array which is then summed

`popup`()

function to start popup window object

`setImage`(*im*)

set image

`updatePlot`()

updates plot, can only be called once plot is initalized with image

1.7.2 PlotGrid

class `Dataplots.`**`PlotGrid`**(*parent=None*)

Methods

1.8 Fitmodels

Contents:

1.9 Image

Contents:

1.9.1 IncomingImage

class `Image.`**`IncomingImage`**

check for images, if found obtain image and send back to main GUI

Methods

`newImage`()

This function checks in directory for new image with proper name if found, it reads it in and then deletes it

run()
: every second search folder for new images, if found get image and emit back to main gui for processing

1.9.2 ProcessImage

class Image.**ProcessImage**(*data*, *exp_data*, *options*, *path*, *run*)
: Processing object for threading purposes @parameters

  data: numpy array options: list of options for fit parameters

  [params,

  type_of_fit, ROI, index

Methods

run()
: process results using methods from fit process and emit

1.10 SpinorMonitor

Contents:

1.10.1 MainWindow

class SpinorMonitor.**MainWindow**
: Main Window for the app, contains the graphs panel and the options panel. Executes main control of all other panels.

  Variables

  - **expData** – Pandas dataframe where all experiment information is kept
  - ***run*** – Run number for the day
  - **path** – Path to data storage folder
  - **processThreadPool** – Dictionary of running threads
  - **process** – Convenience dictionary to initialize objects
  - **ROI** – region of interest
  - **running** – Boolean if data collection thread is active
  - **index** – keeps track of shot internally
  - **image** – ImageWindow widget
  - **plots** – DataPlots widget
  - **options** – Options widget
  - **plot_options** – PlotOptions widget
  - **vis_plots** – VisualPlotter widget
  - **data_tables** – DataTable widget

- **`aux_funcs`** – AuxillaryFunctionContainerWidget widget
- **`tabs`** – QTabWidget, contains other widgets
- **`ipy`** – QIPythonWidget

Methods

`center`()
Centers Window

`change_state`()
start and stop data collection thread

`data_process`(*results_dict*)
process the data, including spawn a thread and increment index

`data_recieved`()
Send message that data was recieved

`end`()
function to stop listening Thread, writes out expData to csv in smae folder as data printing

`finish_thread`(*ind*)
pop the process should destroy it all I think/

`get_options`()
convenience function to return list of options note that function which recieves params must make deep copy or there will be problems!!

`get_roi`()
returns region of interest in list :returns: [xstart,xend,ystart,yend,angle] :rtype: list

`initUI`()
Iniitalize UI and name it. Creat all children widgets and place them in layout

`on_fit_name`(*data*)
Triggers the plots.change_key functions with argument data.

Params data name of fit

`on_message`(*data*)
Send message to output windows

Parameters **`data`** (*object*) – message to send

`set_up_ipy`()
setup the ipython console for use with useful functions

`start`()
Function to start listening thread, connect signals and :var imageThread: IncomingImage object listening for images

`to_ipy`()
push all variables to Ipython notebook

`update_data`(*results_passed*)
function to update plots and push data to ipython notebook

1.11 Optionswidgets

Contents:

1.11.1 FitInfo

class `Optionswidgets.`**`FitInfo`**(*params*, *parent=None*)
custom dialog for fit information

Methods

`close`()

`parse_params`(*tabs*)
populates the tables, row and column determined by run and parameter, so same for all table

1.11.2 Options

class `Optionswidgets.`**`Options`**(*parent=None*)
Panel which defines options for fitting and analyzing images

Methods

`create_fit_panel`()
create a fit panel

`fit_name`

`get_fit_info`()
popup window which has info of all fits

`make_key`(*index*)

`message`

`remove_fit_panel`()
remove fit panel

`save_params`()
update params

`set_current_fit`(*fit_name*)

1.11.3 ParameterEntry

class `Optionswidgets.`**`ParameterEntry`**(*params*, *first*, *parent=None*)
popup box to select parameters

Methods

`readout`()
function to return updated Parameters object

1.11.4 PlotOptions

class `Optionswidgets.`**`PlotOptions`**(*parent=None*)

Widget for Region of Interest Information and other plot options

Methods

`set_roi`(*vec*)

Generate roi strings and print coords

1.12 Auxfunctions

Contents:

1.13 Visualplotterwidget

Contents:

1.13.1 ParamEntry

class `Visualplotterwidget.`**`ParamEntry`**(*parent=None*)

convenience container widget to hold parameters

Methods

1.13.2 PopPlot

class `Visualplotterwidget.`**`PopPlot`**(*mod=None*, *params=None*, *do_fit=False*, *parent=None*)

popup class for plots both static and updating

Variables

- **`ax`** – matplotlib axis
- **`figure`** – matplotlib figure
- **`canvas`** – matplotlib canvas
- **`toolbar`** – matplotlib navigation toolbar

Parameters

- **`mod`** (*lmfit.Model*) – lmfit Model object for fitting
- **`do_fit`** (*Boolean*) – Boolean if fitting should occur
- **`params`** (*lmfit.Parameters*) – fit parameters

Methods

`plot` (*x*, *y*, *xl*, *yl*, *title*, *std*)
plot the data with a new fit if do_fit == True

Params x x vector of points

Params y y vector of points

Params xl x label

Params yl y label

Params title title of plot

Params std standard devation of points

`update` (*x*, *y*, *std=None*)
update the plots call the plot function

Params x x vector of points

Params y y vector of points

Params std standard devation of points

`update_init` (*xl*, *yl*, *title*, *ignore*, *start*)
update the parameters to start

Parameters

- **`title`** (*string*) – title of plot
- **`xl`** (*string*) – x label
- **`yl`** (*string*) – y label
- **`start`** (*int*) – starting index

1.13.3 VisualPlotter

class `Visualplotterwidget.`**`VisualPlotter`**(*parent=None*)
Class to choose plotting visually so it is easy. Will also automatically update plots for every shot. Can automatically fit on a single shot or updating shot basis.

Variables

- *`message`* – pyqtSignal which can be transmitted to main message box
- **`plots`** – Dictionary to hold all the plots
- **`data`** – local copy of entire pandas dataframe
- **`index`** – index of shot
- *`start`* – start of plot region
- *`end`* – end of plot region
- **`ignore_list`** – list of shots to ignore

Fit_models different models to fit too needs to be updated when models added

Methods

`add_fitting_widgets`()
: function populates stacked box for each type of fit

`avg_data`()
: average data and transform self.x_data and self.y_data this is a really crappy algorithm, but it does the trick

`do_fit`()
: do a fit

`filter_ignore`(*data*)
: filter data, list of indices to remove built list of indices not ignored

`ignore_update`()
: update the ignore list parse out test

`make_title_string`()
: make a title string

`make_updating_title_string`()
: make a title string

`message`

`plot_clicked`()
: function called when any plot option is called, sets the start and end values

`static_plot`()
: create new modal popup static plot

`test_fit`()
: do a fit on the test plto

`test_plot`()
: update the test plot

`update_plots`(*df*, *index*)
: Update the updating plots whose references are stored in self.plots

  Params df pandas dataframe holding data

  Params index index of shot

`updating_plot`()
: create an updating plot and fill it with parameters gathered from current state of widgets

`validate`(*el*)
: valid to make sure is a single integer or list comprehension and turn list comprehensions into their equivalent definee here since its an object method in QTGUI

`var_push`(*var_list*)
: add a list of variables to options

`verbose_avg`(*x*, *y*)
: average data and transform self.x_data and self.y_data this is a really crappy algorithm, but it does the trick

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX