Python4LSC - PlotStuff code

Siobhan Tobin

March 1, 2018

Contents

1 Overview **2**

2 User code **3**

3 The *PlotStuff4LSC_1* module as an example **6**

3.1 Setting up to plot 7

3.2 Extrapolation plots 9

3.3 Residual plots 11

Note: code line numbers in this document do not match the line numbers in the actual code. They are included here for easy in-text referencing of specific lines, and to indicate in what order the code generally appears.

1 Overview

PlotStuff4LSC is a companion to *pandas4LSC*. The idea is you run *pandas4LSC* and then you make nice pictures of your results with *PlotStuff4LSC*. Again, there are modules called from other modules. To get started, you need to have all these things in one working directory:

- *PlotStuff4LSC_user* – this is the code that you will edit each time you make pictures
- An xlsx file with the fit info in it from *pandas4LSC*, e.g. `D3LS1_rt200dt50_AllFits_doubles_TSD.xlsx`
- An xlsx file with the data points you actually want to plot (a 'regression data' file), e.g. `D3LS1_rt200dt50_RegData_doubles.xlsx`. Obviously this should match the fit info file!
- *PlotStuff4LSC_1*
- *PlotStuff4LSC_2*
- *PlotStuff4LSC_3*
- *PlotStuff4LSC_4*

The modules *PlotStuff4LSC_1,2,3,4* only need editing if you want extra customisability of the plots. **The code will replace any existing plot pictures of the same names. No warning is given.** When you run the *user* code, you will get twelve png files, three from each of the numbered modules (one extrapolation plots and two residual plots).

The numbered modules correspond to the following:

- *PlotStuff4LSC_1* plots least squares fits of β vs $1 - C/\gamma$
- *PlotStuff4LSC_2* plots least squares fits of $\beta\gamma/C$ vs $\gamma/C - 1$
- *PlotStuff4LSC_3* plots ODR fits of β vs $1 - C/\gamma$
- *PlotStuff4LSC_4* plots ODR fits of $\beta\gamma/C$ vs $\gamma/C - 1$.

2 User code

First we import the necessary modules. This code makes extensive use of the excellent and maleable *matplotlib* module.

```
import numpy as np
import pandas as pd
from scipy.optimize import curve_fit
import scipy.odr.odrpack as odear
import matplotlib.pyplot as plt
from matplotlib.offsetbox import AnchoredText
```

Then we point to the file with the data points to plot (these are the $\beta\gamma/C$ vs $\gamma/C-1$ and β vs $1-C/\gamma$ data used in the linear and cubic regression routines), and the file with all the fit info in it. We also input a source name, whether it is logical doubles or triples data, and whether the observed or theoretical standard deviation was used as the uncertainty on the data points. From hereonin, `BGC` refers to $\beta\gamma/C$ vs $\gamma/C-1$ data, and `B` to β vs $1-C/\gamma$ data. **If in doubt as to whether there are already pictures out there that you want to keep that might have the same filename, change the** `sourcename`.

```
filename="D3LS1_rt200dt50_RegData_doubles.xlsx"
fitsfilename="D3LS1_rt200dt50_AllFits_doubles_OSD.xlsx"
sourcename="D3LS1"
branchingratio=1
DorT='Doubles'
SD='OSD' # or 'TSD' for weighting by theoretical standard deviation
```

The plots are quite customisable from the *user* code alone. Here we specify the minimum and maximum for the relevant axes (we always want to see the intercept at 100% efficiency, hence it is not necessary to specify a horizontal axis minimum). `R` refers to residual plot.

```
#CHANGE DOMAIN (plot horizontal axis maximum)
xMaxBGC=4
#CHANGE RANGE (plot vertical axis maximum)
yMinBGC=10000
yMaxBGC=48000
#change domain of resid plot
xMinRBGC=0
xMaxRBGC=4
#change range resid plot
yMinRBGC=-2000
yMaxRBGC=2000

#CHANGE DOMAIN (plot horizontal axis maximum)
xMaxB=1
#CHANGE RANGE (plot vertical axis maximum)
yMinB=6000
yMaxB=14000
#change domain of resid plot
xMinRB=0.3
xMaxRB=0.85
```

```
#change range resid plot
yMinRB=-600
yMaxRB=600
```

The font size is adjustable from the *user* code for the extrapolation plots, as well as the height and width of the extrapolation plot images. If you wish to adjust the sizes for the residual plots, simply manually adjust the relevant *PlotStuff1,2,3,4* code.

```
# FONT SIZE (not for residual plots, only for extrapolation plots)
fsize=10 # 10 recommended for papers, 12 recommended for powerpoints
# PLOT SIZE (not for residual plots, only for extrapolation plots)
pwidth=6 # 6 recommended for papers, 8 recommended for powerpoints
pheight=4 # 4 recommended for papers and powerpoints
```

Depending on the font size and the exact data set, the legend and the extra text on the plots may look better in different places. Using standard *matplotlib* position code, the extra text (referring to the activity concentration), and the plot legend can be moved to different corners.

```
# POSITION OF LEGEND AND TEXT ON PLOTS
# 'upper right'  : 1
#'upper left'    : 2
#'lower left'    : 3
#'lower right'   : 4
# BGC vs GC-1
BGC_legpos = 4 # 4 is default
BGC_txtpos = 2 # 2 is default
# B vs 1-CG
B_legpos = 3 # 3 is default
B_txtpos = 1 # 1 is default
```

The csv file containing all the data is read in. The data, which should already be sorted, are sorted again!

```
datfileref = filename[:-5]
print()

data = pd.read_excel(filename)
data.head()
datatofit=data.sort_values('1-Co/Ga wm')
```

The code points to the correct columns for the relevant data points. This should not need to be changed if you have just run the *pandas4LSC* code suite. The columns that are used to weight the data will depend on whether the observed standard deviation or the theoretical standard deviation is used.

```
#In which column of the spreadsheet are the x data/y data/yunc located
xdataBGC = np.array(datatofit)[:,0]
ydataBGC = np.array(datatofit)[:,4]

xdataB = np.array(datatofit)[:,11]
ydataB = np.array(datatofit)[:,15]

if SD=='OSD':
```

```
    xuncBGC = np.array(datatofit)[:,3]
    yuncBGC = np.array(datatofit)[:,8]
    xuncB = np.array(datatofit)[:,14]
    yuncB = np.array(datatofit)[:,19]
else: #TSD
    xuncBGC = np.array(datatofit)[:,1]
    yuncBGC = np.array(datatofit)[:,5]
    xuncB = np.array(datatofit)[:,12]
    yuncB = np.array(datatofit)[:,16]
```

Each of the four *PlotStuff4LSC* modules is summoned one at a time. Each module will produce three plots: an extrapolation plot, an absolute residuals plot, and a residuals-as-percentages plot. Enjoy your plots!

```
print()
print("Plotting the least squares fits of B vs 1-C/G")
import PlotStuff4LSC_1
print()
print("Plotting the least squares fits of BG/C vs G/C-1")
import PlotStuff4LSC_2
print()
print("Plotting the ODR fits of B vs 1-C/G")
import PlotStuff4LSC_3
print()
print("Plotting the ODR fits of BG/C vs G/C-1")
import PlotStuff4LSC_4
print()
print("Finished plotting stuff!")
```

3 The *PlotStuff4LSC_1* module as an example

Each of the four *PlotStuff4LSC* modules is more alike than different. The differences are in whether the module uses `xdataB` or `xdataBGC`, and whether least squares or orthogonal distance regression fits are shown. Extra customisability as to the appearance of the plots is easily obtained with some tweaks in the same places in all four modules. So many options are out there to be discovered!

Here the *PlotStuff4LSC_1* module is shown as an example. *PlotStuff4LSC_1* is for least squares regression with B vs $1 - C/\gamma$ data.

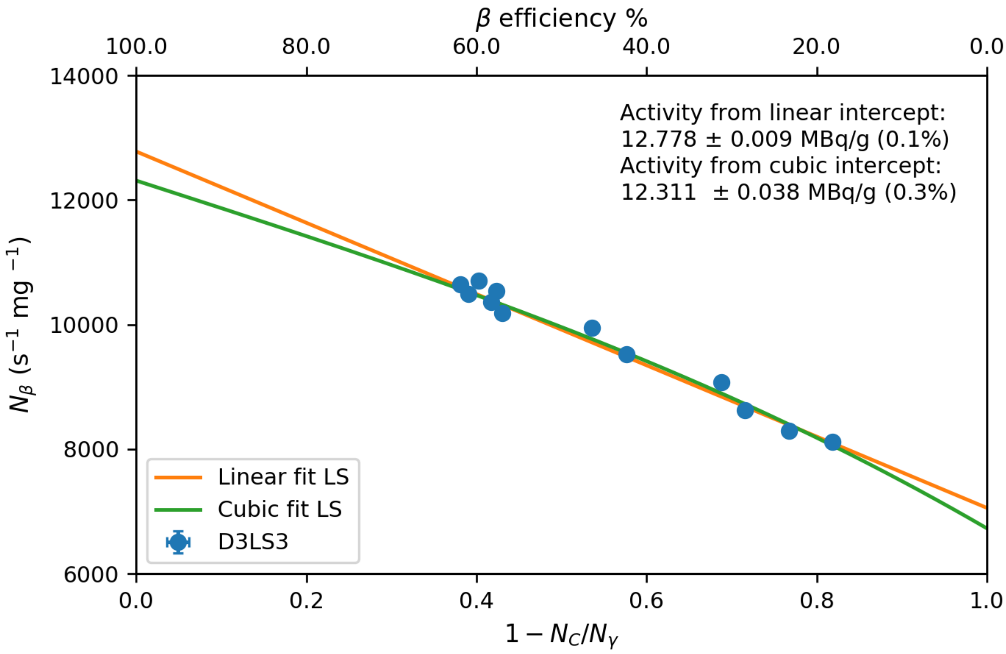

Figure 1: An example of an extrapolation plot showing activity concentration calculated from the intercept. This is output of the *PlotStuff4LSC_1* module.

An example of the extrapolation plot generated by this specific module is given in Figure 1. The default *matplotlib* colour scheme is used in order. β efficiency (as a percentage) is plotted as a second horizontal axis on the top of the plot. The activity concentrations as calculated from the vertical axis intercepts are printed in one corner, along with uncertainties. The uncertainties printed are purely from the estimated uncertainty in the intercept parameters, and not to be taken as the final uncertainty in the activity concentration! The acronym *LS* appears alongside the fits in the legend to indicate the fits are from least squares regression (the equivalent ODR module *PlotStuff4LSC_3* has *ODR* instead). The source name *D3LS3* appears as the legend entry for the data points. The information that the observed standard deviation was used in the weightings for this set of plotted data is apparent from the file name of the plot *D3LS3_B_vs_ineff_LeastSq_PLOT_DoublesOSD.png*. Also obvious from the file name is which values were plotted, the source, the regression method, and whether it is logical doubles or triples data.

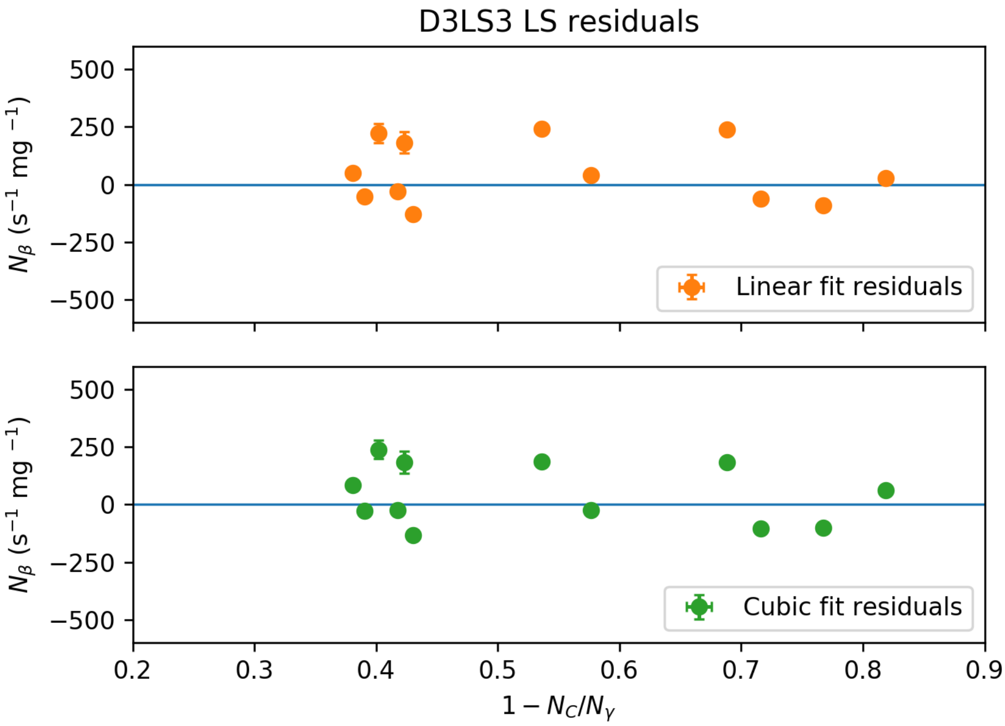

Figure 2: An example of an absolute residuals plot showing residuals from both linear and cubic fits. This is output of the *PlotStuff4LSC_1* module.

3.1 Setting up to plot

It is necessary to import all names from the main working namespace. Then linear and cubic functions are defined.

```
from __main__ import *

def f1(x, m, c):
    """ The linear function y= m*x + c """
    return m*x + c

def f3(x, a, c, d):
    """ The cubic function y= a*x^3 + c*x + d """
    return a*x**3  + c*x + d
```

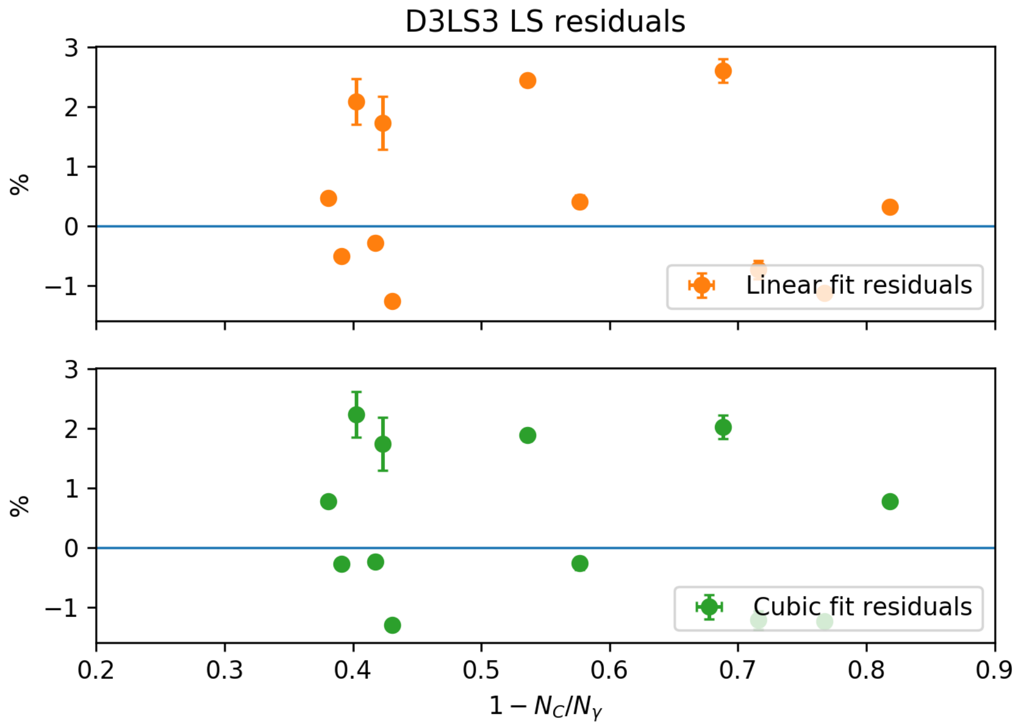

Figure 3: An example of a plot showing the residuals calculated as percentages. This is output of the *PlotStuff4LSC_1* module.

It seemed easier at the time to rerun the regression routine rather than have the code read the contents of some xlsx file. This section looks different depending on whether it is least squares (*PlotStuff4LSC_1* or *PlotStuff4LSC_2*) versus ODR (*PlotStuff4LSC_3* or *PlotStuff4LSC_4*).

```
# LINEAR B vs 1-C/G
poptLW, pcovLW = curve_fit(f1, xdataB, ydataB, p0=None, sigma=yuncB,
         absolute_sigma=True)
xtp = np.linspace(0, xMaxB, 200)
ylinfitW = f1(xtp, *poptLW)

# CUBIC B vs 1-C/G
poptcW, pcovcW = curve_fit(f3, xdataB, ydataB, p0=None, sigma=yuncB,
        absolute_sigma=True)
ycfitW = f3(xtp, *poptcW)
```

Residuals of a function $y = f(x)$ are defined thus:

$$\text{residual}_x = y \text{ data}_x - \text{fit} f(x)$$

Residuals are also expressed in terms of relative percentage:

$$\text{residual}_x\% = \frac{\text{residual}_x}{y\ \text{data}_x} \times 100$$

```
residLT = ydataB - f1(xdataB, *poptLW)
residCT = ydataB - f3(xdataB, *poptcW)
residLP = (ydataB - f1(xdataB, *poptLW))/ydataB*100
residCP = (ydataB - f3(xdataB, *poptcW))/ydataB*100
```

Now we define the uncertainties – this is for the activity concentration caption on the plot. Relative uncertainties are all expressed in terms of percentages.

```
percyunc = yuncB / ydataB *100
unc_intL=np.sqrt(pcovLW[1][1])
unc_intC=np.sqrt(pcovcW[2][2])
reluncL=unc_intL/poptLW[1]*100
reluncC=unc_intC/poptcW[2]*100
```

The caption text is constructed below. The `actfromL` and `actfromC` are the activity concentrations from the linear and cubic fits. `uncL` and `uncC` are the absolute uncertainties in these values. In the `cornertext`, `\n` signifies a new line, `r" ----"` signifies math mode (some LaTeXsymbols are available), with `{0:.3f}` an example of substitution of the first element of the `.format` list to three decimal places (`.3f`).

```
actfromL=poptLW[1]/(branchingratio*1000)
actfromC=poptcW[2]/(branchingratio*1000)
uncL=unc_intL/poptLW[1]*actfromL
uncC=unc_intC/poptcW[2]*actfromC
cornertext="Activity from linear intercept: \n"+r"{0:.3f} $\pm$ {1:.3f}
        MBq/g ({2:.1f}%)".format(actfromL,uncL,reluncL)+"\n"+"Activity from
        cubic intercept: \n"+r"{0:.3f}  $\pm$ {1:.3f} MBq/g ({2:.1f}%)"
        .format(actfromC,uncC,reluncC)
```

3.2 Extrapolation plots

More options are available if you define your plot `f` as being one subplot, one column across, and one row down. `ax` represents all the things you will add to this subplot on the one set of axes.

```
#PLOT
f, ax = plt.subplots(1,1)
```

`ax.errorbar` produces a scatterplot of co-ordinates `xdataB`, `ydataB`. The error bars are of size `xerr=0` and `yerr=yuncB` respectively. For the ODR plot modules, `xerr` $\neq 0$. The scatterplot points are circles `fmt='o'`, and the errorbars do have caps on them `capsize =2`. The `label` for the legend is given by `sourcename`. *matplotlib* handles plotting lines very poorly, so to show the fitted curves, we plot many points (`xtp`, `ylinfitW`) joined by a line. This is done with `ax.plot(xtp, ylinfitW, label='Linear fit LS'`, and similarly for the cubic fit.

```
ax.errorbar(xdataB, ydataB, xerr=0, yerr=yuncB, fmt= 'o', capsize=2,
        label='{0}'.format(sourcename))
ax.plot(xtp, ylinfitW, label='Linear␣fit␣LS')
ax.plot(xtp, ycfitW, label='Cubic␣fit␣LS')
```

The maximum number of ticks on each axis is controlled by changing `nbins` in the `locator_params` class. The font size of the tick labels and whether the labels are rotated or not is controlled with `plt.xticks(fontsize=fontsize-1, rotation=0`, and similarly for the `yticks`. The font size is set to be one less than the default user-specified `fsize` fontsize.

```
plt.locator_params(axis='y', nbins=6)
plt.locator_params(axis='x', nbins=6)
plt.xticks(fontsize=fsize-1, rotation=0)
plt.yticks(fontsize=fsize-1, rotation=0)
```

The maximum and minimum shown on the horizontal and vertical axes is set using `plt.xlim` and `plt.ylim`. These values are specified in the *user* code. Details of the legend are specified using `ax.legend`, with the location controlled by the variable `loc`, and legend font size with `fontsize`.

```
plt.xlim(0,xMaxB)
plt.ylim(yMinB,yMaxB)
ax.legend(loc=B_legpos, fontsize=fsize-1)
```

The caption text referring to the activity concentrations (`cornertext`) is added to the plot with the `add_artist` functionality. There is some gap between the text and the boundary of the shape in which the text sits (`borderpad=1`) but you can't actually see the border of the shape as `frameon=false`. The location is specified through `loc`, and sizing through `prop`. The labels for the horizontal and vertical axes are set using the `ax.set_xlabel` and `ax.set_ylabel` classes.

```
ax.add_artist(AnchoredText(cornertext,prop=dict(size=fsize-1),loc=B_txtpos,
        frameon=False,borderpad=1))
ax.set_xlabel(r"$1␣-␣N_C␣/N_\gamma$",fontsize=fsize)
ax.set_ylabel(r"$N_\beta␣$␣(s$^{-1}$␣mg␣$^{-1}$)",fontsize=fsize)
```

Now a second, top horizontal axis is added. By saying `ax2=ax.twiny`, we are saying to add another set of axes, but make the vertical axis of this second set a twin of the existing vertical (y) axis. There will be 6 ticks between 0 and the `xMaxB` on the second x axis (`new_tick_locations`).

```
ax2 = ax.twiny()
new_tick_locations = np.linspace(0, xMaxB, 6)
```

We define a function `tick_function` to convert x co-ordinates into the new axis format (i.e. from $1 - C/\gamma$ [in the case of *PlotStuff4LSC_1*] to C/γ as a percentage).

```
def tick_function(X):
    V = (1-X)*100
    return ["%.1f" % z for z in V]
```

Now the second set of axes `ax2` can be added to the plot. The horizontal limit of `ax2` will clearly be whatever the largest tick value is, so we summon this with `get_xlim`. The ticks are placed at `new_tick_locations` in the original ordinates through `set_xticks`. The tick labels are calculated as the efficiencies of the new tick locations in the original ordinates using the `tick_function`. Then this second horizontal axis is labelled. All of the stuff about the number, size, and rotation of ticks/bins is repeated to keep *matplotlib* happy, along with the domain and range. `plt.tight_layout` means any white space deemed extra is squashed (e.g. space between the two horizontal axes). Then the overall size of the plot image is set using `plt.gcf().set_size_inches` with user input dimensions `pwidth` and `pheight`. Finally the plot is saved as a png file at 240 dpi, and shown as output to whoever is running the module from the *user* code.

```
ax2.set_xlim(ax.get_xlim())
ax2.set_xticks(new_tick_locations)
ax2.set_xticklabels(tick_function(new_tick_locations))
ax2.set_xlabel(r"$\beta$ efficiency %",fontsize=fsize)
plt.locator_params(axis='y', nbins=6)
plt.locator_params(axis='x', nbins=6)
plt.xticks(fontsize=fsize-1, rotation=0)
plt.yticks(fontsize=fsize, rotation=0)
plt.xlim(0,xMaxB)
plt.ylim(yMinB,yMaxB)
plt.tight_layout()
plt.gcf().set_size_inches(pwidth,pheight)
plt.savefig('{0}_B_vs_ineff_LeastSq_PLOT_{1}{2}.png'.format(sourcename,
        DorT,SD),dpi=240)
plt.show()
```

3.3 Residual plots

The set-up of the residual plots largely mirrors that of the extrapolation plots. Here a few differences are explained. There are two subplots in each residual plot `f`. One subplot has axes `ax1`, the other `ax3`. `sharex` and `sharey` are both `True`, since we want these axes to be aligned for both subplots. In order to keep the colours similar to the extrapolation plots, it is necessary to specify the default color scheme colours with their hex codes (e.g. orange is `'#ff7f0e'`). The blue line at $y = 0$ is added with `ax1.axhline(y=0,color='#1f77b4',linewidth=1)`. Horizontal offset between the subplots is set to zero with `f.subplots_adjust(hspace=0)`. Finally, we do not need two lots of horizontal axis labels (line 19).

```
1  #RESIDUAL PLOTS
2  f, (ax1,ax3) = plt.subplots(2, sharex=True, sharey=True)
3  ax1.errorbar(xdataB, residLT, xerr=0, yerr=yuncB, fmt= 'o', capsize=2,
4          color='#ff7f0e', label='Linear fit residuals')
5  ax1.axhline(y=0,color='#1f77b4',linewidth=1)
6  ax1.legend(loc='lower right', fontsize=10)
7  ax1.set_title('{0} LS residuals'.format(sourcename))
8  ax1.set_ylabel(r"$N_\beta $ (s$^{-1}$ mg $^{-1}$)")
9  ax3.errorbar(xdataB, residCT, xerr=0, yerr=yuncB, fmt= 'o', capsize=2,
10          color='#2ca02c',label='Cubic fit residuals')
11 ax3.axhline(y=0,color='#1f77b4',linewidth=1)
12 ax3.legend(loc='lower right',fontsize=10)
```

```python
ax3.set_xlabel(r"$1 - N_C / N_\gamma $")
ax3.set_ylabel(r"$N_\beta $ (s$^{-1}$ mg $^{-1}$)")
f.subplots_adjust(hspace=0)
plt.xlim(xMinRB,xMaxRB)
plt.ylim(yMinRB,yMaxRB)
plt.tight_layout()
plt.setp([a.get_xticklabels() for a in f.axes[:-1]], visible=False)
plt.gcf().set_size_inches(6,4.5)
plt.savefig('{0}Resid_B_vs_ineff_LeastSq{1}{2}.png'.format(sourcename,
        DorT,SD),dpi=240)
plt.show()
```

```python
f, (ax1,ax3) = plt.subplots(2, sharex=True, sharey=True)
ax1.errorbar(xdataB, residLP, xerr=0, yerr=percyunc, fmt= 'o', capsize=2,
        color='#ff7f0e', label='Linear fit residuals')
ax1.axhline(y=0,color='#1f77b4',linewidth=1)
ax1.legend(loc='lower right', fontsize=10)
ax1.set_title('{0} LS residuals'.format(sourcename))
ax1.set_ylabel("%")
ax3.errorbar(xdataB, residCP, xerr=0, yerr=percyunc, fmt= 'o', capsize=2,
         color='#2ca02c',label='Cubic fit residuals')
ax3.axhline(y=0,color='#1f77b4',linewidth=1)
ax3.legend(loc='lower right',fontsize=10)
ax3.set_xlabel(r"$1 - N_C / N_\gamma $")
ax3.set_ylabel("%")
f.subplots_adjust(hspace=0)
plt.xlim(xMinRB,xMaxRB)
plt.tight_layout()
plt.setp([a.get_xticklabels() for a in f.axes[:-1]], visible=False)
plt.gcf().set_size_inches(6,4.5)
plt.savefig('{0}ResidP_B_vs_ineff_LeastSq{1}{2}.png'.format(sourcename,
        DorT,SD),dpi=240)
plt.show()
```