How to use the python coulter counter fitting script

Updated 21/08/2022

Based on initial script from Amy Briffa, however it has since been significantly modified by Rose McNelly

The script will be published in an upcoming publication, McNelly et al. 2023.

What does the script do?

- The script fits 3 different models to bimodal coulter counter traces:

1) Two normal curves (double normal)
2) Two lognormal curves (double lognormal)
3) Lognormal for B granule peak and normal for A granule peak (lognormal-normal)

- It compares the different fittings using parameter uncertainties and a standard error of regression test. It produces estimates of A and B granule diameter and B granule content (as a ratio) for each model.

What are the scripts?

2) Batch_running_of_coulter_counter_script – this script will take each coulter counter csv file in turn and feed them into the coulter_counter_fitting_script. It also saves all the parameters from the different fittings and produces summary excel sheets (see later for info about outputs)

Batch_running_of_coulter_counter_fitting_script	21/08/2022 10:51	Jupyter Source File	9 KB
coulter_counter_fitting_script	21/08/2022 11:06	Jupyter Source File	52 KB
Running script	21/08/2022 16:43	Jupyter Source File	3 KB

1) Running script – this script is used to change our working directory and submit the other two scripts

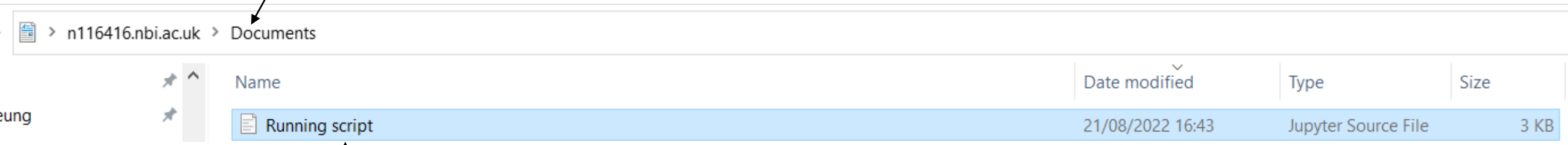
3) coulter_counter_fitting_script – this script does the actual fittings. It takes one input file and fits the double normal, double lognormal and lognormal-normal curves to the data. It produces a PDF with graphs of the 3 different fittings and a comparison of the 3.

Why can't we just use the coulter_counter_fitting_script by itself? Technically we could but in order to get excel sheets with all the parameters in we need to use script 2 (batch_running_of_coulter_counter_script). It also means that if we have multiple coulter counter files to analyse at once we only need to submit the script once and not for every input file. The running script also makes it easier so that we can tell the computer where are input files are (so that we don't have to copy them into our Jupyter lab directory – see later for info about Jupyter lab).

How should my data be organised?

I always place the Running script (only this has to be here) in my Documents folder as this is easy to navigate to using Jupyter notebook (see later)

Documents folder on your computer



Only the Running script needs to be here

How should my data be organised?

Everything else can be in a different folder, this can be anywhere and doesn't have to be within your Documents folder, but if you want it in your Documents folder this is fine too. It should be organised as shown here:

data_inputs folder – contains all the csv files from the coulter counter (if it contains other file types it should run ok as the script should only read in csv files)

data_inputs	21/08/2022 16:57	File folder	
output	21/08/2022 17:41	File folder	
Batch_running_of_coulter_counter_fitting_...	21/08/2022 17:00	Jupyter Source File	2,892 KB
coulter_counter_fitting_script	21/08/2022 11:06	Jupyter Source File	52 KB

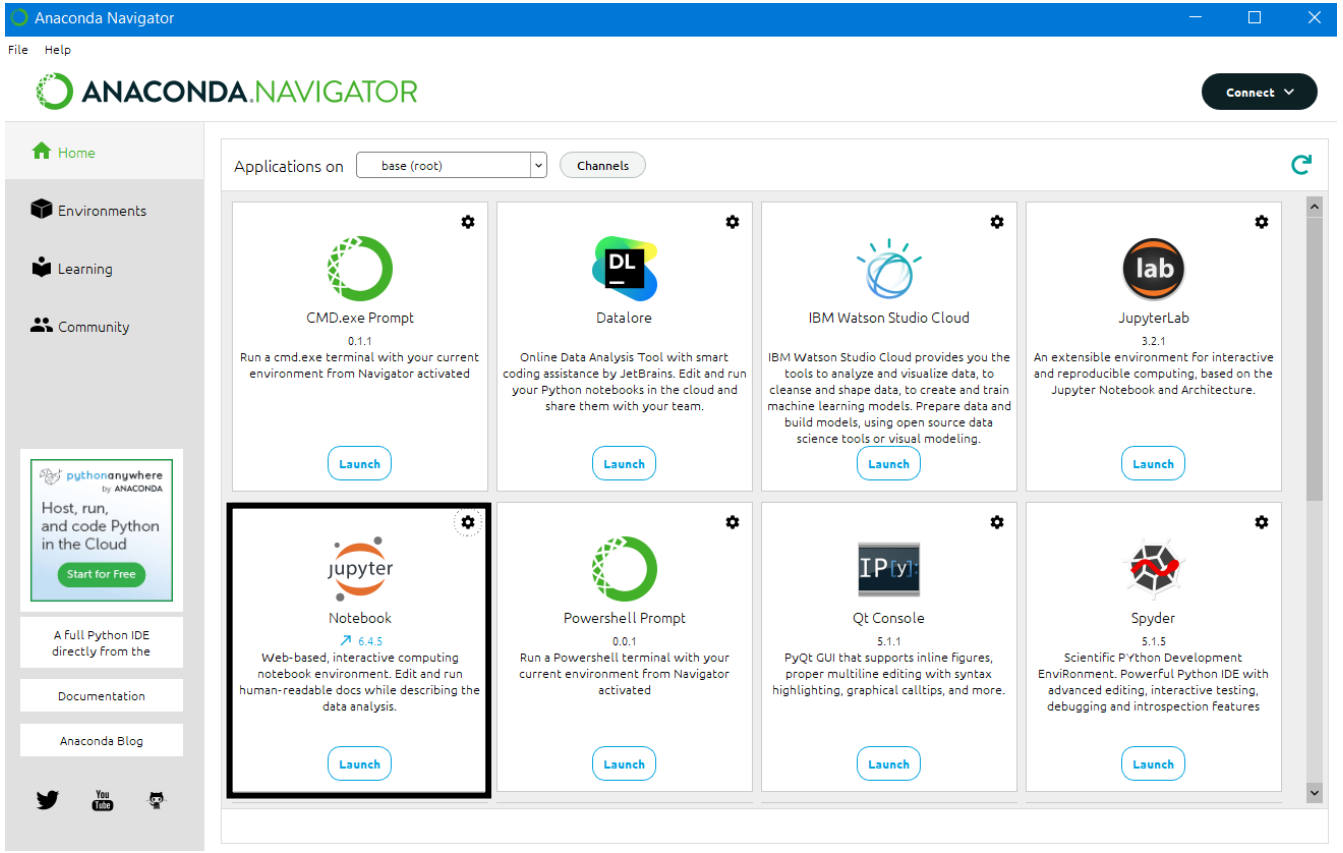
Empty folder called output – the output files (see later) will be added to this folder

The batch_running_of_coulter_counter_fitting_script and the coulter_counter_fitting_script

The folder which is organised like this is you **working directory,** take note of the name of this folder and where it is (the full path) as you'll need it

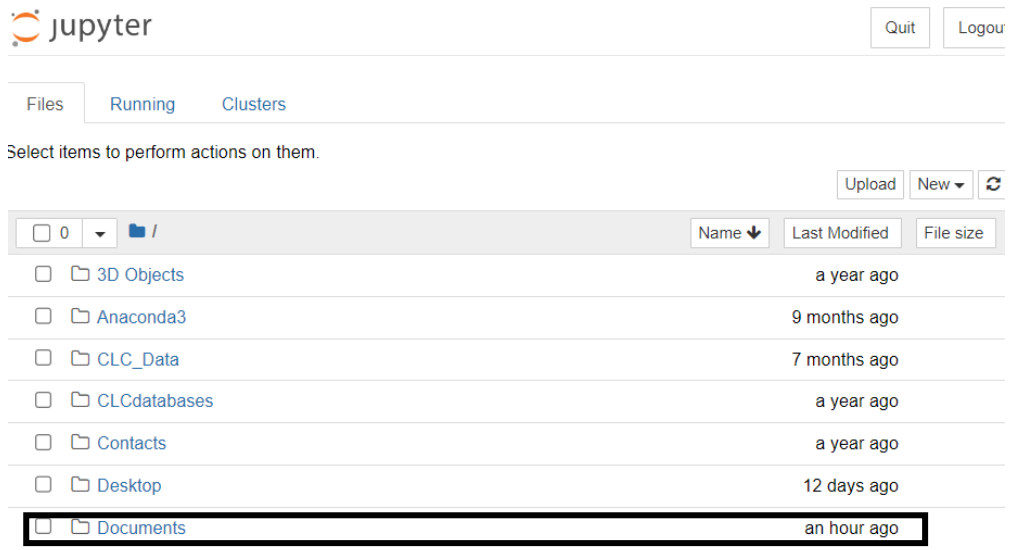
How do I run the script?

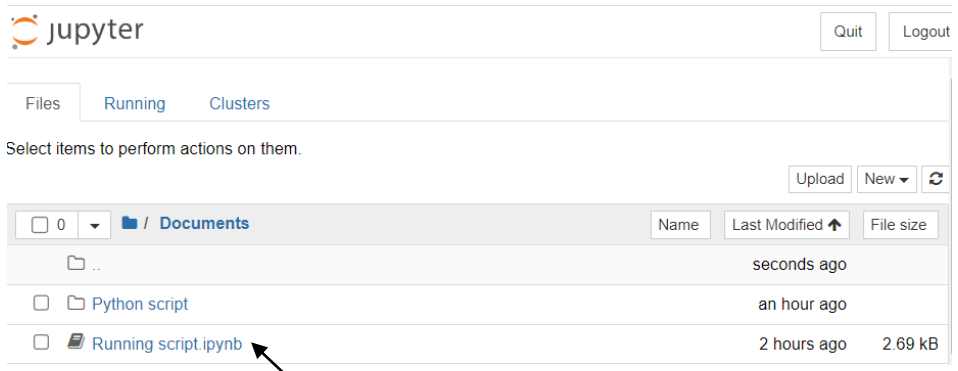
1) Download Anaconda Navigator (https://docs.anaconda.com/anaconda/install/) and open Jupyter Notebook (this will open in an internet browser)



How do I run the script?

2) Navigate to your Documents folder





You'll see the Running script here

3) Open the Running script

How do I run the script?

jupyter Running script Last Checkpoint: 2 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

Chunk 1

```
In [ ]: # Formatting notebook to fit the browser size

        from IPython.core.display import display, HTML
        display(HTML("<style>.container { width:100% !important; }</style>"))
```

Chunk 2

```
In [ ]: #Loading libraries and packages

        import os
```

```
In [ ]: #Changing working directory
        #Note - in your working directory you need folders called:
            #1)data_inputs - which contains all the csv files from the coulter counter
            #2)output - which is empty which your files will be placed in


        os.chdir(FILL YOUR WORKING DIRECTORY IN HERE)
        os.getcwd()
```

```
In [ ]: #Run script

        %run Batch_running_of_coulter_counter_fitting_script.ipynb

        #This will execute the Batch_running_of_coulter_counter_fitting_script which will then run the coulter_counter_fitting_script in
        #This will produce an output folder containing:
            #1) PDF files for each individual trace which compare the fittings from all 3 models
            #2) Excel file called 'Parameters_from_fittings' which contains all the parameters from the different models
            #3) Excel file called 'Samples_which_fitted_successfully_and_samples_which_failed' which shows if the models couldn't be fiti
            #4) Excel file called 'Samples_where_the_script_failed' which shows where there was an error during data import, e.g. maybe
```

4) Run the first two chunks (shown as grey boxes), by clicking in each chunk and pressing Ctrl and Enter.

If it has run successfully a small number should appear in the square brackets on the left of the chunk

How do I run the script?

jupyter Running script Last Checkpoint: 2 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [ ]:
```
# Formatting notebook to fit the browser size

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [ ]:
```
#Loading libraries and packages

import os
```

In [ ]:
```
#Changing working directory
#Note - in your working directory you need folders called:
    #1)data_inputs - which contains all the csv files from the coulter counter
    #2)output - which is empty which your files will be placed in


os.chdir(FILL YOUR WORKING DIRECTORY IN HERE)
os.getcwd()
```

Chunk 3

In [ ]:
```
#Run script

%run Batch_running_of_coulter_counter_fitting_script.ipynb

#This will execute the Batch_running_of_coulter_counter_fitting_script which will then run the coulter_counter_fitting_script in
#This will produce an output folder containing:
    #1) PDF files for each individual trace which compare the fittings from all 3 models
    #2) Excel file called 'Parameters_from_fittings' which contains all the parameters from the different models
    #3) Excel file called 'Samples_which_fitted_successfully_and_samples_which_failed' which shows if the models couldn't be fiti
    #4) Excel file called 'Samples_where_the_script_failed' which shows where there was an error during data import, e.g. maybe
```

5) In chunk 3, change your working directory to the location which contains the other scripts, your data_inputs and output folder. You need to put in the full path, not just the folder name, and put it within quotation marks, e.g. ‘U:\Working_Directory_Folder’

Run the chunk, if it has worked then it should print the working directory folder path directly underneath the chunk.

How do I run the script?

jupyter Running script Last Checkpoint: 2 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [ ]:
```
# Formatting notebook to fit the browser size

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [ ]:
```
#Loading libraries and packages

import os
```

In [ ]:
```
#Changing working directory
#Note - in your working directory you need folders called:
    #1)data_inputs - which contains all the csv files from the coulter counter
    #2)output - which is empty which your files will be placed in


os.chdir(FILL YOUR WORKING DIRECTORY IN HERE)
os.getcwd()
```

In [ ]:
```
#Run script

%run Batch_running_of_coulter_counter_fitting_script.ipynb

#This will execute the Batch_running_of_coulter_counter_fitting_script which will then run the coulter_counter_fitting_script in
#This will produce an output folder containing:
    #1) PDF files for each individual trace which compare the fittings from all 3 models
    #2) Excel file called 'Parameters_from_fittings' which contains all the parameters from the different models
    #3) Excel file called 'Samples_which_fitted_successfully_and_samples_which_failed' which shows if the models couldn't be fiti
    #4) Excel file called 'Samples_where_the_script_failed' which shows where there was an error during data import, e.g. maybe t
```

Chunk 4

6) Run chunk 4 – this submits the Batch_running_of_coulter_counter_script which itself will submit the coulter_counter_fitting_script (you won't have to do anything)

You'll start to see the outputs appear in the output folder

If you want to know how the script works see later

What are the outputs?

Parameters_from_fittings – contains all the parameters from the 3 different fittings and also contains the uncertainties and the standard error of regression

Samples_where_the_script_failed – contains a list of filenames where the script failed (most likely during data import), if a filename appears in this spreadsheet it could be because the coulter counter CSV file is corrupted or not in the correct format

Samples_which_fitted_successful_and_samples_which_failed – contains a list of filenames for each of the different fitting methods for samples which worked and samples where the fitting failed. The fitting might have failed if the data is not bimodal in shape.

Parameters_from_fittings	21/08/2022 16:58	Microsoft Excel W...	8 KB
Samples_where_the_script_failed	21/08/2022 16:58	Microsoft Excel W...	6 KB
Samples_which_fitted_successfully_and_samples_which_failed	21/08/2022 16:58	Microsoft Excel W...	6 KB
TOWWC138_rep1_26 Jan 2021_104.#m4	21/08/2022 16:58	Adobe Acrobat D...	49 KB
TOWWC139_rep1_26 Jan 2021_107.#m4	21/08/2022 16:58	Adobe Acrobat D...	49 KB
TOWWC140_rep1_26 Jan 2021_111.#m4	21/08/2022 16:58	Adobe Acrobat D...	48 KB
TOWWC141_rep1_26 Jan 2021_112.#m4	21/08/2022 16:58	Adobe Acrobat D...	49 KB
TOWWC142_rep1_26 Jan 2021_113.#m4	21/08/2022 16:58	Adobe Acrobat D...	48 KB
TOWWC143_rep1_26 Jan 2021_114.#m4	21/08/2022 16:58	Adobe Acrobat D...	49 KB
TOWWC183_rep1_19 Jan 2021_02.#m4	21/08/2022 16:58	Adobe Acrobat D...	49 KB

PDF files – contain graphs of all the different fittings

See subsequent slides for more info about all the different outputs

Parameters_from_fittings

Filename

Double normal fitting

Double lognormal fitting

Lognormal-normal fitting

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1		Sample_ name	B granule diameter double normal fitting	A granule diameter double normal fitting	B granule content double normal fitting	Uncertai nity for the double normal fitting	Standard error of regressio n for the double normal fitting	B granule diameter double lognorma l fitting	A granule diameter double lognorma l fitting	B granule content double lognorma l fitting	Uncertai nity for the double lognorma l fitting	Standard error of regressio n for the double lognorma l fitting	B granule diameter lognorma l-normal fitting	A granule diameter lognorma l-normal fitting	B granule content lognorma l-normal fitting	Uncertai nity for the lognorma l-normal fitting	Standard error of regressio n for the lognorma l-normal fitting
2	0	TOWWC1	6.231614	20.92452	0.137802	1.9286	0.002115	10.37982	22.65257	0.272935	3.218637	0.002184	7.940557	21.28468	0.188094	2.487674	0.001966
3	1	TOWWC1	6.686771	20.78239	0.110641	2.01302	0.001934	13.42058	22.88176	0.335764	5.647152	0.001968	8.778745	21.25482	0.173613	3.168227	0.001791
4	2	TOWWC1	6.461777	22.97916	0.19153	2.188529	0.00225	9.048968	24.82652	0.289721	2.42644	0.002248	7.995676	23.44914	0.246104	2.439573	0.002076
5	3	TOWWC1	5.974037	21.61401	0.228324	3.679256	0.002897	7.057384	25.84657	0.317692	3.045525	0.002552	6.626907	22.34325	0.269839	3.416541	0.002459
6	4	TOWWC1	5.156727	17.29375	0.117164	1.900296	0.002091	7.356125	19.47067	0.242194	2.264614	0.001995	5.815544	17.50307	0.140694	2.354822	0.002062
7	5	TOWWC1	5.868601	22.5642	0.156516	2.82146	0.002494	8.837488	25.42819	0.281812	3.171382	0.002474	7.415623	23.21856	0.218214	3.349011	0.002327
8	6	TOWWC1	5.932095	21.95082	0.108861	2.112591	0.002243	10.66762	23.55213	0.213071	3.649782	0.002362	8.021781	22.20665	0.146492	2.770785	0.002114
9																	

Each row contains the parameters from a different input file

Uncertainty is a measure of how sure we can be about the parameters. Higher uncertainties mean that the parameters in the models could take lots of different values and still result in the same fitting. Therefore **we want the uncertainty to be as low as possible** as this means we are more certain that the model parameters are 'correct'. Standard error of regression is a form of R^2 for non-linear models. It measures how well the predicted model fits the data. Higher standard errors of regression means that the data deviates from the model more. So **we want the standard error of regression to be as low as possible**.

Samples_where_the_script_failed

	A	B
1		**Samples where the script failed - most likely during data import, so check coulter_counter file**
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		

If any of the input files failed to load into the script properly they will be listed in this column.

Here this column is empty so all the files were inputted correctly.

NOTE – this does NOT mean that the fitting won't work as it has failed before the fitting has been attempted.

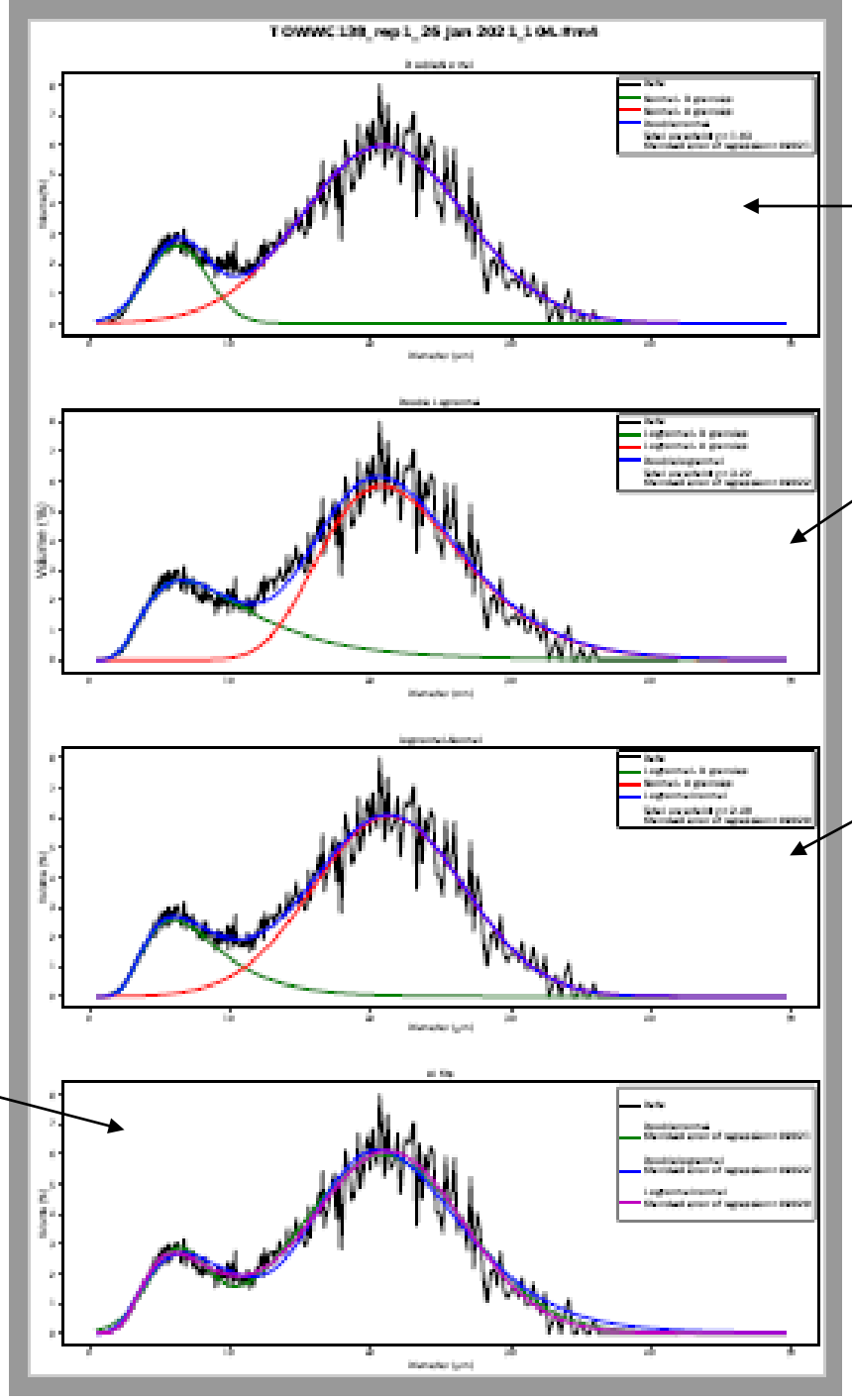
Samples_which_fitted_successful_and_samples _which_failed

	A	B	C	D	E	F	G
1		**Successful samples for the double normal fitting**	**Failed samples for the double normal fitting**	**Successful samples for the double lognormal fitting**	**Failed samples for the double lognormal fitting**	**Successful samples for the lognormal-normal fitting**	**Failed samples for the lognormal-normal fitting**
2	**0**	TOWWC138_rep1_26 Jan 2021_104.#m4	All_successful	TOWWC138_rep1_26 Jan 2021_104.#m4	All_successful	TOWWC138_rep1_26 Jan 2021_104.#m4	All_successful
3	**1**	TOWWC139_rep1_26 Jan 2021_107.#m4		TOWWC139_rep1_26 Jan 2021_107.#m4		TOWWC139_rep1_26 Jan 2021_107.#m4	
4	**2**	TOWWC140_rep1_26 Jan 2021_111.#m4		TOWWC140_rep1_26 Jan 2021_111.#m4		TOWWC140_rep1_26 Jan 2021_111.#m4	
5	**3**	TOWWC141_rep1_26 Jan 2021_112.#m4		TOWWC141_rep1_26 Jan 2021_112.#m4		TOWWC141_rep1_26 Jan 2021_112.#m4	
6	**4**	TOWWC142_rep1_26 Jan 2021_113.#m4		TOWWC142_rep1_26 Jan 2021_113.#m4		TOWWC142_rep1_26 Jan 2021_113.#m4	
7	**5**	TOWWC143_rep1_26 Jan 2021_114.#m4		TOWWC143_rep1_26 Jan 2021_114.#m4		TOWWC143_rep1_26 Jan 2021_114.#m4	
8	**6**	TOWWC183_rep1_19 Jan 2021_02.#m4		TOWWC183_rep1_19 Jan 2021_02.#m4		TOWWC183_rep1_19 Jan 2021_02.#m4	
9							

Tells us if the fitting failed for the double normal (column C), double lognormal (column E) and lognormal-normal (column G), if any of the fittings failed then the filename will be listed in these columns. This means that the model couldn't be fitted to the data most likely as it significantly deviates from being bimodally distributed. It could be the case that one model might not fit but the other two might.

What should I do if the fitting is failing quite often? You could go into the coulter_counter_fitting_script and adjust the initial parameter values to fit your data more closely (HINT – use the visualisation of the initial parameter values to help). If not this could suggest that this is not an appropriate model for your data and you might need to use something else, e.g. is it more unimodal?

PDF files



Double normal fitting

Double lognormal fitting

Lognormal-normal fitting

Comparison of double normal (green), double lognormal (blue) and lognormal-normal (purple) fittings with the raw data (black) and the standard error of regression for each model

Black = coulter counter data, green = B granules, red = A granules, blue = overall fitting. Also includes the uncertainty and standard error of regression measurements.

How does the coulter_counter_fitting_script work?

These 2 chunks format the script and import all the libraries and packages which are needed.

You'll probably never need to alter these.

```
In [ ]: # Formatting notebook to fit the browser size

        from IPython.core.display import display, HTML
        display(HTML("<style>.container { width:100% !important; }</style>"))
```

```
In [ ]: # Loading libraries and packages

        %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib as mpl
        import matplotlib.ticker as mtick
        import scipy.optimize
        import scipy.stats
        import scipy.integrate as integrate
        import scipy.integrate as special
        import pandas as pd
        import os
        import linecache
        from statistics import mean

        plt.style.use('ggplot') # use ggplot style for graps
```

This is only intended to provide an overview, for more info look at the comments in the script. It is also more for interest/if you might need to alter the script and you can run the script easily without understanding this.

How does the coulter_counter_fitting_script work?

The next chunk imports the data (a CSV file) from the data_inputs folder. It only imports one file at a time.

It sorts and converts all the data to the correct format. Crucially it adjusts the volume for the bin width (column called Diff_volume_density).

```
In [ ]: # Import data

# path of input file(s) - note that the file(s) has to be in a folder called data_inputs
input_data_file = os.path.join('data_inputs', filename) #note - this uses a variable called filename as the input file, if you ar

#getting the base name of the file which will be used to name the output file in the same way
name = ((os.path.splitext(input_data_file))[0]).lstrip('data_inputs\\') #This takes the file name, splits it at the extension (e.

# define column headings for the dataframe
headings_list = ['Bin_number','Bin_diameter_lower', 'Diff_volume', 'Diff_number_perc', 'Diff_number_gram', 'Diff_surface_area', '

# read in dataframe using pandas library
input_data_df = pd.read_csv(input_data_file, sep='\,', engine='python', skiprows=55,
                            names=headings_list)

# calculate bin width in a new column
input_data_df['Bin_width'] = input_data_df['Bin_diameter_lower'].shift(-1) - input_data_df['Bin_diameter_lower']

# read off the final bin number into a variable, called input_data_bin_max, and then drop this row
input_data_bin_max = input_data_df.iloc[-1,1]
input_data_df = input_data_df.drop(input_data_df.index[input_data_df.tail(1).index])

# set 'Bin_number' column to be the index
input_data_df = input_data_df.set_index('Bin_number')

# change datatype of index from float to integer
input_data_df.index = input_data_df.index.astype('int')

# Calculate mid-point of bins
input_data_df['Bin_midpoint'] = input_data_df['Bin_diameter_lower']+input_data_df['Bin_width']/2

# Calc Diff_volume_density using: Diff_volume/Bin_width
input_data_df['Diff_volume_density'] = input_data_df['Diff_volume']/input_data_df['Bin_width']

# convert input data to lists
bin_midpoint_input_list = input_data_df['Bin_midpoint'].tolist()
diff_volume_density_input_list = input_data_df['Diff_volume_density'].tolist()
```

How does the coulter_counter_fitting_script work?

The script defines the lognormal and normal functions mathematically:

```
In [ ]: # Define the lognormal and normal functions

        def lognormal_func(x_, A_1_, mu_1_, sigma_1_):
            return (  (A_1_/(x_*sigma_1_*np.sqrt(2.*np.pi)))*np.exp( -((np.log(x_)-mu_1_)/sigma_1_)**2/2. )  )

        def normal_func(x_, A_2_, mu_2_, sigma_2_):
            return (  (A_2_/(sigma_2_*np.sqrt(2.*np.pi)))*np.exp( -( (x_ - mu_2_)/sigma_2_ )**2/2. )  )
```

How does the coulter_counter_fitting_script work?

The script starts with the lognormal-normal fitting:

Defines the lognormal-normal fitting mathematically and makes it into a function, this means that it can be easily used. A_1_, mu_1_ ….. are the parameters which are in the mathematical equation

```
In [ ]: ## LOGNORMAL-NORMAL FUNCTION
```

```
In [ ]: # Define the lognormal-normal function

        def lognormal_normal_func(x_, A_1_, mu_1_, sigma_1_, A_2_, mu_2_, sigma_2_):
            return ( lognormal_func(x_, A_1_, mu_1_, sigma_1_) + normal_func(x_, A_2_, mu_2_, sigma_2_) )
```

Uses the scipy.optimise.curve_fit package to create a function which we will use later to fit the lognormal-normal curve to the input data

```
In [ ]: # Define a function to fit the lognogrmal-normal function

        def func_lognormal_normal_fit(bin_midpoint_input_, diff_volume_density_input_):
            #initial parameter values:
            lognormal_initial_params_ = [20.0, 1.8, 0.4]
            normal_initial_params_ = [50.0, 20.0, 6.0]
            fit_params_, fit_cov_ = scipy.optimize.curve_fit(lognormal_normal_func,bin_midpoint_input_, diff_volume_density_input_,
                                                             p0 = lognormal_initial_params_+normal_initial_params_, bounds = ([0,0,0,0,0,0
            fit_params_err_ = np.sqrt(np.diag(fit_cov_))
            return fit_params_, fit_params_err_

        #This function will return: {list of fit parameters}, {list of fit uncertanties}
```

The initial parameters which are used in the fitting

We restrict the values of the parameters (e.g. so we don't get negative diameters) using bounds

How does scipy.optimisie.curve_fit actually perform the fitting? It uses a non-linear least squares method

How does the coulter_counter_fitting_script work?

```
# #Plots the initial parameters to visualise how close the initial parameters fit
# #This can be useful if your curves are very different from normal and the initial parameters need to be adjusted
# #Commented out so not produced when running in batch mode

# #Initial parameter estimates - ensure these are the same as shown above
# lognormal_initial_params = [20.0, 1.8, 0.4]
# normal_initial_params = [50.0, 20.0, 6.0]

# # define x_values to plot
# x_vals_plot = np.arange(0.5,50,0.5)

# # define curves to fit
# lognormal_vals_curve = [lognormal_func(i_, lognormal_initial_params[0],lo
#                                        lognormal_initial_params[2]) for i_ in x_vals_plot]
# normal_vals_curve = [normal_func(i_, normal_initial_params[0],normal_initial_params[1],normal_initial_params[2])
#                      for i_ in x_vals_plot]
# lognormal_normal_vals_curve = [lognormal_normal_func(i_, lognormal_initial_params[0],lognormal_initial_params[1],lognormal_init
#                                        normal_initial_params[0],normal_initial_params[1],normal_initial_params[2])
#                         for i_ in x_vals_plot]

# #plot graphs of initial parameters
# fig, ax = plt.subplots(1,1,figsize=(15,8))
# for spine in ['left','right','top','bottom']:
#     ax.spines[spine].set_color('k')
# ax.set_facecolor('white')
# ax.plot(input_data_df['Bin_midpoint'].tolist(), input_data_df['Diff_volume_density'].tolist(),
#         linewidth=2, color='k',label='Data' )
# ax.plot(x_vals_plot, lognormal_vals_curve,
#         linewidth=2, color='green',label='Lognormal' )
# ax.plot(x_vals_plot, normal_vals_curve,
#         linewidth=2, color='r',label='Normal' )
# ax.plot(x_vals_plot, lognormal_normal_vals_curve,
#         linewidth=2, color='b',label='Lognormal-normal' )
# fig.suptitle('Initial parameter guess: '+name, fontsize=16)
# ax.set_xlabel("Diameter ($\mu $m)", fontsize=18)
# ax.xaxis.set_major_locator(plt.MaxNLocator(6))
# ax.set_ylabel("Volume (%)", fontsize=18)
# ax.tick_params(axis='both', which='major', labelsize=16)
# leg_00 = ax.legend(fontsize=16, loc='upper right', ncol=1,facecolor='white', framealpha=1)
# leg_00.get_frame().set_edgecolor('k')
# plt.tight_layout()
# fig.subplots_adjust(top=0.9)
```

Make sure these match the initial parameters in the function in the chunk above

Optional - Sometimes we might want to visualise how the initial curve (using the initial parameters) fits the data. This chunk will plot the initial curve over the input_data

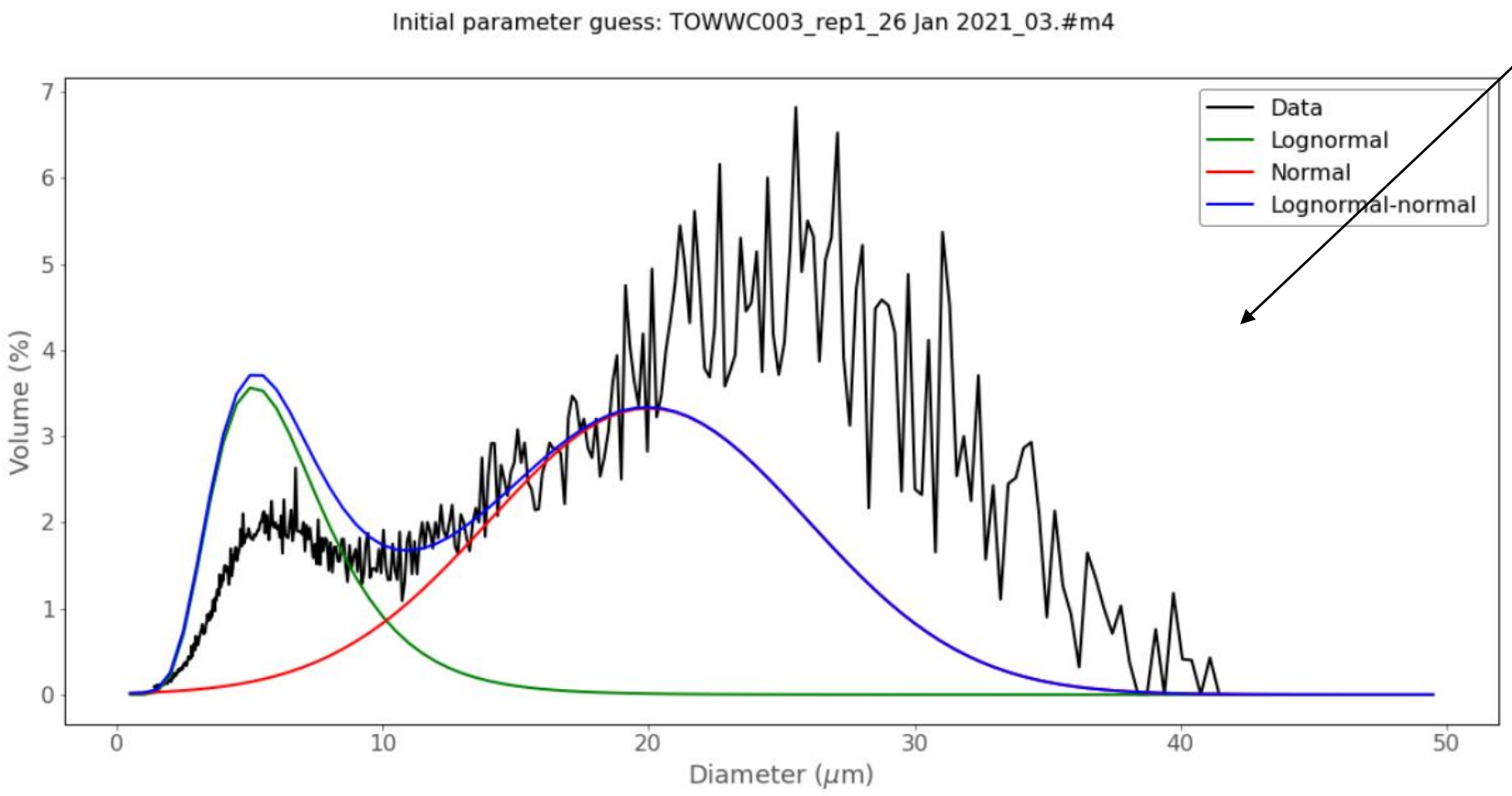
This script has already been optimised for bimodal distributions, but this chunk could be useful if you want to change the initial parameters to fit traces which look substantially different (if you've already checked to make sure the current model isn't fitting properly).

At the moment this chunk won't run as it has been commented out, to uncomment it select all the text and press CTRL and /

The output graph will be produced at the bottom of the Jupyter notebook (in the Running_script) and NOT as a PDF.

How does the coulter_counter_fitting_script work?

Example output from the initial parameter plot:



The initial curve doesn't have to be that close in shape to the final curve, this example still produces a successful fitting.

Should I bother to plot the initial parameters? Personally I wouldn't bother doing this as I've found that the initial parameters are fine and allow the model to be fitted to a wide range of bimodal coulter counter traces.

How does the coulter_counter_fitting_script work?

The next chunk tries to fit the lognormal-normal fitting to the input data:

```
In [ ]: # Run the lognormal-normal fitting

        #create empty lists which the successful and non-successful fittings will be stored in
        failed_lognormal_normal = []
        successful_lognormal_normal = []
        fitting_failed_lognormal_normal = []

        #run the lognormal-normal fit with the exception catcher
        try:
            output_lognormal_normal_fit_params_list, output_lognormal_normal_fit_err = func_lognormal_normal_fit(bin_midpoint_input_list,

        exc
                                                                                  get \n included in the sample names when we make a l

            #this sets the parameters to fitting failed so in the output file we get this message
            B_granule_diameter_lognormal_normal = "fitting failed"
            A_granule_diameter_lognormal_normal = "fitting failed"
            B_granule_area_lognormal_normal = "fitting failed"
            A_granule_area_lognormal_normal = "fitting failed"
            B_granule_content_lognormal_normal = "fitting failed"
            lognormal_normal_uncertainity = "fitting failed"
            lognormal_normal_standard_error_of_regression = "fitting failed"

            #this is used in an if statement during plotting so that if failed we don't get a graph
            fitting_failed_lognormal_normal = "yes"
```

There is an exception catcher – this means that if the model can't be fitted to the input data the error won't cause the script to crash. Instead all the parameters will be saved as 'fitting failed'

We are using the function we defined earlier with the scipy.optimise.curve_fit package

How does the coulter_counter_fitting_script work?

Also in this chunk we calculate the granule diameters, granule content, uncertainty and standard error of regression:

```
else:
    successful_lognormal_normal = name.split('\n') #we need the \n if not we get \n included in the sample names when we make a l

    output_lognormal_fit_params_list = output_lognormal_normal_fit_params_list[:3]
    output_normal_fit_params_list = output_lognormal_normal_fit_params_list[3:]

    #Getting the parameters from the models:

    #lognorm mean is equal to exp(mu + (sigma^2/2))
    B_granule_diameter_lognormal_normal = np.exp(output_lognormal_normal_fit_params_list[1] + ((output_lognormal_normal_fit_paran

    #norm mean is equal to mu
    A_granule_diameter_lognormal_normal = output_lognormal_normal_fit_params_list[4]

    #areas under curves - with the scipy integrate function: the individual functions have been redefined using the output parame
    #and the integrate function is used, the output is a tuple with an estimated value of the integral first and the second value
    #B granules - lognorm fitting
    def lognorm_func_B_granules(x_):
        return (  (output_lognormal_normal_fit_params_list[0]/(x_*output_lognormal_normal_fit_params_list[2]*np.sqrt(2.*np.pi)))*
    B_granule_area_lognormal_normal = scipy.integrate.quad(lognorm_func_B_granules,0, np.inf) #integrate between 0 and infinity
    #A granules - normal fitting
    def normal_func_A_granules(x_):
        return (  (output_lognormal_normal_fit_params_list[3]/(output_lognormal_normal_fit_params_list[5]*np.sqrt(2.*np.pi)))*np.
    A_granule_area_lognormal_normal = scipy.integrate.quad(normal_func_A_granules, 0, np.inf) #integrate between 0 and infinity

    Total_area_lognormal_normal = A_granule_area_lognormal_normal[0] + B_granule_area_lognormal_normal[0]
    B_granule_content_lognormal_normal = B_granule_area_lognormal_normal[0]/Total_area_lognormal_normal

    #reduced uncertainity
    lognormal_normal_uncertainity = np.sum(output_lognormal_normal_fit_err)

    #standard error of regression
    lognormal_normal_vals_list = [lognormal_normal_func(input_data_df['Bin_midpoint'].tolist()[i_], *output_lognormal_normal_fit
                        for i_ in range(len(input_data_df['Diff_volume_density']))]
    lognormal_normal_standard_error_of_regression = np.sqrt( (1/len(input_data_df['Diff_volume_density']-2)) * (sum((np.array(inp
```

The diameters are calculated from the means of the mathematical equations

The scipy.integrate.quad package is used to integrate the area under the curves and the relative areas are used to calculate the granule content

The uncertainity is the sum of the errors of all the individual parameters

Standard error of regression, uses this equation:

$$s(b_1) = \sqrt{\frac{1}{n-2} * \frac{\sum (y_i - \hat{y}_i)^2}{\sum (x_i - \bar{x})^2}}$$

- **n**: total sample size
- $\mathbf{y_i}$: actual value of response variable
- $\hat{\mathbf{y}}_\mathbf{i}$: predicted value of response variable
- $\mathbf{x_i}$: actual value of predictor variable
- $\bar{\mathbf{x}}$: mean value of predictor variable

How does the coulter_counter_fitting_script work?

Plotting the final fitting:

This is commented out as we want to save it in a PDF and not just display the final fitting graph here.

In [ ]:
```
# # Plot optimized fit of lognormal-normal fitting (if it was successful)
# # Commented out so not produced when running in batch mode

# if fitting_failed_lognormal_normal == "yes":
#     pass
# else:
#     # define x_values to plot
#     x_vals_plot = np.arange(0.5,50,0.5)

#     # define curves to fit
#     lognormal_vals_curve = [lognormal_func(i_, *output_lognormal_fit_params_list) for i_ in x_vals_plot]
#     normal_vals_curve = [normal_func(i_, *output_normal_fit_params_list) for i_ in x_vals_plot]
#     lognormal_normal_vals_curve = [lognormal_normal_func(i_, *output_lognormal_normal_fit_params_list) for i_ in x_vals_plot]

#     #plot graphs of initial parameter
#     fig, ax = plt.subplots(1,1,figsize=(15,8))
#     for spine in ['left','right','top','bottom']:
#         ax.spines[spine].set_color('k')
#     ax.set_facecolor('white')
#     ax.plot(input_data_df['Bin_midpoint'].tolist(), input_data_df['Diff_volume_density'].tolist(),
#             linewidth=2, color='k',label='Data' )
#     ax.plot(x_vals_plot, lognormal_vals_curve,
#             linewidth=2, color='green',label='Lognormal - B granules' )
#     ax.plot(x_vals_plot, normal_vals_curve,
#            linewidth=2, color='r',label='Normal - A granules' )
#     ax.plot(x_vals_plot, lognormal_normal_vals_curve,
#             linewidth=2, color='b',label='Lognormal-normal')
#     ax.plot(x_vals_plot, lognormal_vals_curve,
#             linewidth=2, color='w', alpha=0, label='Total uncertainty = %.2f\nStandard error of regression =%.4f' %
#            (lognormal_normal_uncertainity, lognormal_normal_standard_error_of_regression))
#     fig.suptitle('Lognormal-normal fit: '+name, fontsize=16)
#     ax.set_xlabel("Diameter ($\mu $m)", fontsize=18)
#     ax.xaxis.set_major_locator(plt.MaxNLocator(6))
#     ax.set_ylabel("Volume (%)", fontsize=18)
#     ax.tick_params(axis='both', which='major', labelsize=16)
#     leg_00 = ax.legend(fontsize=16, loc='upper right', ncol=1,facecolor='white', framealpha=1)
#     leg_00.get_frame().set_edgecolor('k')
#     plt.tight_layout()
#     fig.subplots_adjust(top=0.9)
```

How does the coulter_counter_fitting_script work?

- The script goes through the same process with the double normal and then the double lognormal fittings
- It then combines the fits together in one graph:

```
In [ ]: # COMBINING THE OUTPUT OF THE DIFFERENT FITS
```

```
In [ ]: # # Plotting all fits
        # # Commented out so not produced when running in batch mode

        # #defining x values
        # x_vals_plot = np.arange(0.5,50,0.5)
        # fig, ax = plt.subplots(1,1,figsize=(15,8))
        # for spine in ['left','right','top','bottom']:
        #     ax.spines[spine].set_color('k')
        # ax.set_facecolor('white')
        # ax.plot(input_data_df['Bin_midpoint'].tolist(), input_data_df['Diff_volume_density'].tolist(),
        #         linewidth=2, color='k',label='\nData\n' )
        # if fitting_failed_double_normal == "yes":
        #     pass
        # else:
        #     ax.plot(x_vals_plot, double_normal_vals_curve,
        #         linewidth=2, color='g',label='Double normal\nStandard error of regression =%.4f\n' % (double_normal_standard_error_of_r
        # if fitting_failed_double_lognormal == "yes":
        #     pass
        # else:
        #     ax.plot(x_vals_plot, double_lognormal_vals_curve,
        #         linewidth=2, color='b',label='Double lognormal\nStandard error of regression =%.4f\n' % (double_lognormal_standard_erro
        # if fitting_failed_lognormal_normal == "yes":
        #     pass
        # else:
        #     ax.plot(x_vals_plot, lognormal_normal_vals_curve,
        #         linewidth=2, color='m',label='Lognormal-normal\nStandard error of regression =%.4f\n' % (lognormal_normal_standard_erro
        # ax.set_xlabel("Diameter ($\mu $m)", fontsize=18)
        # ax.xaxis.set_major_locator(plt.MaxNLocator(6))
        # ax.set_ylabel("Volume (%)", fontsize=18)
        # ax.tick_params(axis='both', which='major', labelsize=16)
        # ax.set_title('All fits', fontsize=18)
        # leg_00 = ax.legend(fontsize=16, loc='upper right', ncol=1,facecolor='white', framealpha=1)
        # leg_00.get_frame().set_edgecolor('k')
```

This is commented out as we want to save it in a PDF and not just display the final fitting graph here.

How does the coulter_counter_fitting_script work?

This chunk then plots four graphs 1) Double normal fitting, 2) Double lognormal fitting, 3) Lognormal-Normal fitting, 4) Overlay of all 3 fittings and saves it as a PDF

```
# Combining all plots together in one pdf file
# Note - the if-else loops are used so that if the fitting failed then the graph is blank and says fitting failed
# Note - saves the pdf in a folder called output

#defining x values
x_vals_plot = np.arange(0.5,50,0.5)

fig, ax = plt.subplots(4,1,figsize=(15,25))
fig.suptitle(name, fontsize=20, y=1.01, fontweight="bold")
fig.subplots_adjust(top=0.95)
fig.tight_layout(h_pad=8)

#first graph - double normal graph
```

↓

Long chunk so not all shown here

↓

```
#Saving plots in a folder called output
plot_name = (name.strip('\n')+"_output.pdf")
fig.savefig('output/'+name+".pdf", bbox_inches = 'tight', dpi=300)
```

How does the coulter_counter_fitting_script work?

The final chunk saves all the parameters from the different fittings and saves them to a variable called ‘fitting_parameters’ which the Batch_running_of_coulter_counter_script uses

```
#Getting the parameters from the different fits
Double_normal_parameters = (B_granule_diameter_double_normal, A_granule_diameter_double_normal, B_granule_content_double_normal,
Double_lognormal_parameters = (B_granule_diameter_double_lognormal, A_granule_diameter_double_lognormal, B_granule_content_double
Lognormal_normal_parameters = (B_granule_diameter_lognormal_normal, A_granule_diameter_lognormal_normal, B_granule_content_lognor

#Joining together the sample name and fitting parameters
name_str = (name,) #converts the sample name into a tuple so it is in the same format as the parameters
fitting_parameters = (name_str + Double_normal_parameters + Double_lognormal_parameters + Lognormal_normal_parameters)
```

How does the Batch_running_of_coulter_counter_script work?

Starts the same as the coulter_counter_fitting script

```
In [1]: # Formatting notebook to fit the browser size

        from IPython.core.display import display, HTML
        display(HTML("<style>.container { width:100% !important; }</style>"))
```

```
In [2]: # Loading libraries and packages

        import os
```

How does the Batch_running_of_coulter_counter_script work?

```
#Running in batch mode
#This needs a folder called data_inputs with csv files to use and a folder called output which it will put the fitting parameters

parameters_from_fittings = [] #need to make an empty data frame with which I can add the fitting parameters to

#empty lists which I will fill with sample names which have either worked or failed
did_not_run = []
failed_lognormal_normal_fit_list = []
successful_lognormal_normal_fit_list = []
failed_double_normal_fit_list=[]
successful_double_normal_fit_list=[]
failed_double_lognormal_fit_list=[]
successful_double_lognormal_fit_list=[]


#this is a for loop which reads input files in a directory called data_inputs and uses them as input for the coulter_counter_fit_
for filename in os.listdir('data_inputs/'): #runs with all files in the data_inputs folder

    if filename.endswith(".csv") or filename.endswith(".CSV"): #will only work on csv files, so means you can have other file typ
        file_name = filename #file_name is the variable which the run_coulter_count_fit_rose_using_python_fitting script uses as
        try:
            %run coulter_counter_fitting_script.ipynb #this line runs the script

        except IndexError or RuntimeError:
            did_not_run.append(filename)

        else:
            parameters_from_fittings.append(fitting_parameters) #takes the fitting parameters from all the samples and stores tog
            #creating lists of samples which have been successful and those which have failed
            successful_lognormal_normal_fit_list.append(successful_lognormal_normal)
            failed_lognormal_normal_fit_list.append(failed_lognormal_normal)
            successful_double_normal_fit_list.append(successful_double_normal)
            failed_double_normal_fit_list.append(failed_double_normal)
            successful_double_lognormal_fit_list.append(successful_double_lognormal)
            failed_double_lognormal_fit_list.append(failed_double_lognormal)

    continue
```

It reads through all CSV files in the data_inputs folder and one by one runs them with the coulter_counter_fitting_script

If there is an error it stores the filename but it shouldn't cause the script to crash

The fitting_parameters variable (which contains all the parameters from the different fittings) are saved in one large data frame

How does the Batch_running_of_coulter_counter_script work?

```
#concatenates the fitting parameters into an excel file
#produces an output called 'Parameters_from_fittings.xlsx'
parameters_from_fittings_data_frame = pd.DataFrame(parameters_from_fittings, columns=['Sample_name','B granule diameter double no
                                                                                       'B granule content double normal fitting',
                                                                                       'Standard error of regression for the doubl
                                                                                       'A granule diameter double lognormal fittir
                                                                                       'Uncertainity for the double lognormal fitt
                                                                                       'B granule diameter lognormal-normal fittir
                                                                                       'B granule content lognormal-normal fitting
                                                                                       'Standard error of regression for the logno
parameters_from_fittings_data_frame.to_excel("output/Parameters_from_fittings.xlsx")


#creating data frames with the fittings which were successful and those which failed
successful_lognormal_normal_fit_dataframe = pd.DataFrame(successful_lognormal_normal_fit_list)
failed_lognormal_normal_fit_dataframe = pd.DataFrame(failed_lognormal_normal_fit_list)
successful_double_lognormal_fit_dataframe = pd.DataFrame(successful_double_lognormal_fit_list)
failed_double_lognormal_fit_dataframe = pd.DataFrame(failed_double_lognormal_fit_list)
successful_double_normal_fit_dataframe = pd.DataFrame(successful_double_normal_fit_list)
failed_double_normal_fit_dataframe = pd.DataFrame(failed_double_normal_fit_list)


#if loops which (if the dataframes are empty) fills it with either all successful or all failed as appropriate
All_successful="All successful"
All_failed = "All failed"

if failed_lognormal_normal_fit_dataframe.empty:
    failed_lognormal_normal_fit_dataframe = pd.DataFrame(['All_successful'])
if failed_double_lognormal_fit_dataframe.empty:
    failed_double_lognormal_fit_dataframe = pd.DataFrame(['All_successful'])
if failed_double_normal_fit_dataframe.empty:
    failed_double_normal_fit_dataframe = pd.DataFrame(['All_successful'])

if successful_lognormal_normal_fit_dataframe.empty:
    successful_lognormal_normal_fit_dataframe = pd.DataFrame(['All_failed'])
if successful_double_lognormal_fit_dataframe.empty:
    successful_double_lognormal_fit_dataframe = pd.DataFrame(['All_failed'])
if successful_double_normal_fit_dataframe.empty:
    successful_double_normal_fit_dataframe = pd.DataFrame(['All_failed'])

#merging the data frames together and saving the results in excel
#produces an output called 'Samples_which_fitted_successfully_and_samples_which_failed.xlsx'
successful_and_failed_fittings = pd.concat([successful_double_normal_fit_dataframe[0], failed_double_normal_fit_dataframe[0], suc
                    failed_double_lognormal_fit_dataframe[0], successful_lognormal_normal_fit_dataframe[0], failed_lognormal_norm
successful_and_failed_fittings_results = successful_and_failed_fittings.set_axis(["Successful samples for the double normal fitt
                                         "Successful samples for the double lognormal fitting", "Failed samples for the double lo
                                         "Successful samples for the lognormal-normal fitting", "Failed samples for the lognormal
successful_and_failed_fittings_results.to_excel("output/Samples_which_fitted_successfully_and_samples_which_failed.xlsx")
```

The fitting parameters are all saved together in one Excel file

Creates an Excel file which contains which fittings worked/didn't work for each file – called "Samples_which_fitted_successfully_and _samples_which_failed"

How does the Batch_running_of_coulter_counter_script work?

Creates an Excel file which contains the list of filenames which didn't import properly, this is called "Samples_where_the_script_failed"

```
#creating a file of samples which failed - most likely during data import
did_not_run_dataframe = pd.DataFrame(did_not_run, columns=["Samples where the script failed - most likely during data import, so
did_not_run_dataframe.to_excel("output/Samples_where_the_script_failed.xlsx")
```

Overview

1) You submit it using the Running script

You only need to do this first step, everything else is automated

2) The Running script starts the Batch_running_of_coulter_counter_script

3) The Batch_running_of_coulter_counter_script takes every csv file in the data_inputs folder and submits it to the coulter_counter_fitting_script (one at a time)

4) The coulter_counter_fitting_script performs the fitting