Curvalyser and Paramalyser

v1.10/v1.10

The Curvalyser is intended for the analysis of force-extension curves obtained by atomic force spectroscopy measurements with biological cells. It provides an objective means to evaluate a large batch of recorded datasets automatically. This includes baseline correction, noise reduction using a novel wavelet-based technique, contact point calibration, step detection, fitting procedures, extraction of characteristic curve parameters, plotting, and a quality check to sort out erroneous data. The extracted parameters can then be filtered, statistically analysed and plotted by the Paramalyser. This manual explains how to use these programs. Basic Python skills are required to customise some configuration settings, such as the user-defined callback functions.

1. LICENSE

The software included in this distribution was developed by Jan Opfer, j [at] opfer [dot] net.

2. INSTALLATION

The software runs on different platforms, including Windows, Linux and Mac OS. It was tested with Python 2.7.1, NumPy 1.5.1 and matplotlib 1.0.1, but should work with newer versions as well. Please follow these instructions to install the required components:

- download and install Python 2.7.1 (or higher): http://www.python.org/download/
- download and install NumPy 1.5.1 (or higher): http://sourceforge.net/projects/numpy/files/NumPy/
- download and install matplotlib 1.0.1 (or higher): http://sourceforge.net/projects/matplotlib/files/matplotlib/
- unpack the contents of the ZIP file
- open a command line window and change to the directory containing the unpacked files

Note: It is essential to select Numpy and matplotlib versions matching your Python version.

On Linux it might be easier to use the package manager of your distribution. If you run into trouble or want to install the software on Mac OS, we recommend using the Enthought Python Distribution instead. It contains all required components and is available for Windows, Linux and Mac OS.

Optional: To speed up the wavelet-based noise reduction algorithm, we provide a Python module, which needs to be compiled before use. A compatible C compiler must be set up properly. For details see http://docs.python.org/install/index.html. To install the module, execute the following command in the directory *C library (optional)*:

```
> python setup.py install
```

After successful installation the module will be detected automatically.

3. CURVALYSER

3.1. Basic configuration

Before first use, the format and location of the input files must be specified. To this end, open the configuration file *config-curvalyser* in a text editor. If the data is saved in text files, set the variable `file_format` to `'text'` and `columns` to the column numbers of the extension and force records, respectively (note that counting starts with 0). The columns can either be separated by any whitespace (default) or by the string given in `column_delimiter`. This example assumes that the data is contained in the comma-separated columns 2 and 3:

```
file_format = 'text'
columns = (1,2)
column_delimiter = ','
```

In case of Curvalyser (**.crv*) and old JPK files (**.out*), a single line is sufficient, as the correct columns are determined automatically:

```
file_format = 'crv'
```

```
file_format = 'jpk-old'
```

New JPK files (**.jpk-force*) can contain multiple "segments". Therefore, `JPK_segments` must be defined to select the number of the trace and of the retrace segment, respectively:

```
file_format = 'jpk'
JPK_segments = (0,1)
```

Extension and force can optionally be scaled to a decent order of magnitude:

```
multiplier_x = 1e6
multiplier_y = 1e12
```

Negative values may be needed if the extension does not increase with the distance from the surface or if indentation does not correspond to a positive sign of the force. All program output is based on the units of the values in the input files, which are multiplied by these factors.

A separate configuration file should be created for each set of force spectra ("experiment") and stored in the directory *config* (e.g. *001*, *002* and so on). This allows distinguishing the corresponding input and output files. Common configuration parameters can still be defined in *config-curvalyser* and included in each experiment-specific configuration file (such as *config/001*):

```
execfile('config-curvalyser')
file_pattern = 'data/001/*.txt'
```

Here, only a different set of input files is selected by the variable `file_pattern`, which points to the absolute or relative directory containing the force spectra to be analysed.

By default, the name of the configuration file (e.g. *001*) is used as output directory below `base_output_dir` (here: *output/001*). This can be overridden by the parameter `output_dir`:

```
output_dir = 'output2/001'
```

Further parameters are listed in section 3.10 (e.g. to select a certain range of input files or to define the limits of experimental settings to be checked).

3.2. Program execution

The Curvalyser is run using the configuration file *config/001* by typing

```
> python Curvalyser.py 001
```

By default, configuration files are searched in the folder *config*. If no configuration file is given, all files in this directory will be processed. Multiple configuration files can be selected individually

```
> python Curvalyser.py 001 002 003 config2/010 config2/020
```

or by file masks:

```
> python Curvalyser.py 00* config2/0?0
```

The following command line options control the tasks performed by the program:

Option	Example	Description
`-h`		show a help message
`-l`	`-l output/log.txt`	set log file
`-c`	`-c"exp_id=='001'"`	select experiments by a Python expression
`-e`	`-e WT`	select only experiments belonging to the given experiment type
`-O`		overwrite existing output files
`-f`	`-f data/*.txt`	same as configuration parameter `file_pattern`
`-r`	`-r 100:200:10`	same as configuration parameter `file_range`
`-o`	`-o output`	same as configuration parameter `base_output_dir`
`-v, -vv`		be verbose (`-v`) or very verbose (`-vv`)
`-d`	`-d 4.5`	set the primary de-noising parameter
`-i`	`-i 5`	set the relative indicator threshold
`-p`		create force plots (`-pf`), indicator plots (`-pi`) or both (`-pfi`)
`-s`		show plots in a graphical user interface
`-V`		display program version

A very flexible way to pick out experiments according to some rule is provided by the option "-c". It takes a Python expression as parameter returning either true or false, depending on whether the file is to be included or not. The condition can be attached to the consecutive configuration file number (`config_file_no`), the file name (`config_file`), the experiment ID (`exp_id`) or any configuration parameter (stored in the dictionary `config`). In this example, only experiments whose ID start with the prefix "WT" are chosen:

```
> python Curvalyser.py -c"exp_id[:2]=='WT'"
```

To select experiments of a certain type (specified by the configuration parameter `experiment_type`), it is more convenient to use the option "-e".

3.3. Baseline correction

Force curves can be baseline-corrected to compensate instrumental drift and to allow for correct calibration of the contact point as well as of the zero-force level. Depending on the configuration parameter `fit_baseline`, either a linear (`1`) or a quadratic fit (`2`) is subtracted from the retrace curve before any further evaluation is done. Fitting is performed in a smooth part at the end of the retrace curve where the distance from the surface is maximal and only background noise is present. This range must be free of steps, tip-surface interactions or other sample-specific effects and sufficiently long to obtain an accurate fit. Therefore, the length of the retrace extension should not be too short. A baseline fit is calculated iteratively every `baseline_step_len` data points, starting from the end of the curve. It stops automatically if the residual sum of squares (RSS) locally exceeds the expected value by the factor `baseline_max_rel_local_RSS`. The end point of the fit interval is reverted to the last local minimum of the RSS if `baseline_return_to_local_min` is set to `1`. In case the length of the fit falls below `baseline_fit_min_width`, the curve is excluded from further analysis. If the baseline fit stops too early, a higher threshold `baseline_max_rel_local_RSS` or a lower step interval `baseline_step_len` should be chosen (and vice versa).

3.4. Noise reduction

Noise reduction is crucial for reliable calculation of step heights (if `step_measuring_lt`/`rt` is `0` or `1`), contact point calibration, and for accurate determination of some characteristics of the force curves, such as the peak force. Different noise reduction methods can be selected: ReNoiR [1] (a wavelet-based noise filter), Gaussian smoothing [2] and the Savitzky-Golay filter [3]. The first is preferable if the force spectra contain sharp features (such as spikes or steep, narrow steps) and the second is ideal for curves with more or less constant plateaus separated by smooth transitions.

The filter strength of ReNoiR (`denoising_method = 'renoir'`) depends on the parameters T_0 (`denoising_param`) and T_1 (`denoising_param2`). The higher the values the more noise is reduced and the more details of the signal are lost. See reference [1] for a description of the algorithm. The number of recursions (`denoising_recursions`) and levels (`denoising_levels`) usually do not need to be changed. To minimize border distortion caused by the wavelet filter, the signal can be extended by point reflection at both ends. `denoising_padding_lt` and `denoising_padding_rt` are the numbers of padded data points on the left and right, respectively. The resulting number of samples should be a power of two.

If Gaussian smoothing (`denoising_method = 'gauss'`) is performed, `denoising_param` represents the standard deviation of the smoothing kernel (see [2]).

In case of Savitzky-Golay filtering (`denoising_method = 'savgol'`), the same parameter determines the half window size and `savgol_order` the order of the polynomial kernel (see [3]).

`denoising_param` and `denoising_param2` can be user-defined callback functions. They must take three parameters: `noise_level` (the estimated standard deviation of the noise), `rows` (the number of samples), and `meta`. The latter is a dictionary containing metadata, such as the headers of JPK force spectra (**.out*, **.jpk-force*) or the additional information comprised by Curvalyser files (**.crv*). Example:

```
denoising_param = lambda noise_level, rows, meta: noise_level * 3
```

3.5. Contact point calibration

The extension where the indentation force becomes zero for the first time during retraction is referred to as “contact point”. Its determination is necessary to calibrate the zero point of the extension, e.g. of detected

steps (see section 3.6) or of the peak force. Although most subsequent calculations are influenced by the calibration, it is only performed if `find_contact_pos` is `1`. Otherwise, the extension at the reversal point between trace and retrace curve defines the zero point.

If no contact point was found in the retrace curve, if it was found further away from the reversal point than stipulated by `max_contact_pos`, or if the maximum indentation force is lower than `min_contact_force`, the trace curve is used instead. In case this also fails, the calibration and all parameters depending on it will be omitted.

3.6. Step detection

A moving step fit is deployed to detect upward steps in the retrace curve (see [4] for a description of the algorithm). Briefly, it produces an indicator that is correlated to sharp transitions within the signal. The sharper a transition is and the longer its range is the higher will be the indicator. Therefore, local maxima correspond to possible step positions. The indicator can be smoothed by convolution with a Gaussian kernel of standard deviation `MSF_sigma`. Choose higher values if too many local maxima appear and lower values if neighbouring steps are not detected separately. The half window size `MSF_window` governs the influence of both criteria (sharpness and range of the transition). It should be set to `1` if the steps are narrow or sharp (i.e. showing steep flanks) or to a higher value if the steps are smooth (try e.g. `10` or `100`). Higher values increase detection sensitivity for low, wide steps and lower values increase lateral resolution (important for the detection of hardly separated, narrow steps); each at the expense of the other. An absolute threshold (`indicator_threshold`) and a relative one that is multiplied by the noise level of the indicator (`indicator_relative_threshold`), define the detection sensitivity (the greater of the two values counts). Increase/decrease the thresholds to reduce false-positives/false-negatives. The step positions are marked by vertical yellow lines in the force curve and indicator diagrams and the effective threshold by a black horizontal line in the indicator diagrams (see section 3.8).

Detected steps can be filtered out by defining the minimum step height (`step_min_height`), the minimum and maximum step width (`step_min_width` and `step_max_width`), or the minimum average amplitude of the indicator (`step_min_slope`).

Linear fits of the flanks left and right of the detected steps are performed if the distance to the next step (or to the beginning/end of the force curve) is at least `step_fit_min_len` data points. Then, the slope and interception are recorded. The lengths of the fit intervals are optimized automatically within `step_fit_min_len` and `step_fit_max_len` by minimizing the root mean square errors. In any case, the intervals are terminated at the nearest neighbouring step (or at the beginning/end of the force curve). As most steps are not perfectly sharp, the intervals usually should not start right at the detected position, but at the beginning/end of the transition. These points can be located by finding the next local minimum in the denoised retrace curve to the left and the next local maximum to the right of the step position (these will be referred to as “lower/upper edge”). Alternatively, the inner limits of the fit intervals can be specified in terms of a distance to the left and to the right where the indicator curve falls below values determined by the parameters `step_fit_proximity_lt` and `step_fit_proximity_rt`, respectively. Both parameters are related to the height of the indicator peak at the step position and can take values between `0` and `1`. Higher values correspond to a smaller distance (and vice versa). A value in the middle (e.g. `0.5`, the width at half height) is usually a good starting point. `1` means no distance, which only makes sense for extremely sharp transitions. In the force curve diagrams, the fits are represented by red and green lines. Additionally, the intervals are highlighted in red and green on the retrace curve and the edges are marked by red and green circles in the interactive force plots. The latter also appear in the indicator diagrams.

Step heights can be determined by two different methods: If `step_measuring_lt`/`rt = 0`, the difference in force at the edges is calculated from the denoised retrace curve. If `step_measuring_lt`/`rt = 1` or `2`, the linear fits of the step flanks are extrapolated and the difference of their ordinates at the detected step positions is calculated. In both cases, the step heights are indicated by red and green crosses in the force curve diagrams. If the fit cannot be performed (because the interval would be smaller than `step_fit_min_len` data points) and `step_measuring_lt`/`rt = 1`, the first height calculation method is deployed as a fallback method. If the height cannot be calculated, a detected step is not completely ignored, but its height is not recorded.

To exclude the beginning or end of the retrace curve, `indicator_margin_lt` or `indicator_margin_rt` can be set to a positive value.

3.7. Fitting procedures

The slope of the trace or retrace curve is fitted to the left and to the right of the point corresponding to an indentation force given by `indentation_fit` over intervals specified by `indentation_lt_fit_width` and `indentation_rt_fit_width`, respectively. If `indentation_curve = 1`, the trace curve is used, otherwise the retrace curve.

The indentation part of the trace curve (from the contact point to the left) can be fitted with the user-defined callback function `trace_fit_function`, which is initialized with the parameters defined in the Python sequence `trace_fit_init_params`. Analogously, the section of the retrace curve between the contact point and the first step can be fitted using `retrace_fit_function` and `retrace_fit_init_params`.

3.8. Plotting

Two types of diagrams can be plotted: Force curves and indicator curves. The indicator curves are useful to adjust and check the step detection algorithm (see section 3.6). To determine if these diagrams are plotted into files of the format `plot_format`, set `plot_force_curves` and `plot_indicators` to either `0` or `1`. The plots will be written to the subdirectory *plots*. Additionally, they are displayed in an interactive graphical user interface if `show_plots` is `1`. The horizontal and vertical data range is specified by `plot_xmin`, `plot_xmax`, `plot_ymin`, and `plot_ymax`. The resolution of the output files can be changed by setting their width and height in `plot_size_force_curves` and `plot_size_indicator_curves`. `plot_step_fit_width` defines the extension of the lines drawn into the plots to depict the fitted slopes on the left and right edge of the steps. A list of elements to be plotted into the force curves can be specified by the parameter `plot_features`.

3.9. Output files

All extracted information about the force curves is saved into two files in the output directory (see section 3.1): *curves.txt* contains all parameters specific to the whole curve and *steps.txt* all data related to the steps. The first line of the files is a header shortly describing the meaning of the columns. Basically, the units found in the input files are used, but may be scaled by multiplication factors (`multiplier_x` and `multiplier_y`).

Plots are written to the subdirectory *plots*. The file names contain the names of the corresponding input files. “(failed)” is appended if a curve could not be analysed (e.g. because the baseline correction failed).

3.10. Overview of important configuration parameters

Parameter	Values / example	Description
Input and output		
`file_pattern`	`'data/*.txt'`	file mask for force curves to be analysed
`file_format`	`'txt'`	text format (also set `columns`!)
	`'jpk'`	new JPK format (**.jpk-force*)
	`'jpk-old'`	old JPK format (**.out*)
	`'crv'`	Curvalyser format; default
`file_range`	`'100:200:10'`	select a range of files (format: 'start:stop:step,start:stop:step,...'; counting starts with 1; negative value: count from the end)
`columns`	`(1,2)`	specify the columns containing extension and force data, respectively (counting starts with 0!)
`column_delimiter`	`','`	string used to separate values; default: any whitespace
`JPK_segments`	`(0,2)`	segment numbers of the trace and retrace data, respectively
`multiplier_x`	`1e6`	multiplier for rescaling the extension data
`multiplier_y`	`1e12`	multiplier for rescaling the force data
`output_dir`	`'output2/001'`	output directory (automatically determined by `base_output_dir` and `exp_id` if omitted)
`exp_id`	`'123'`	ID used to distinguish multiple sets of force spectra (experiments); default: name of the configuration file
`base_output_dir`	`'output'`	base output directory; only used if `output_dir` is omitted
`nominal_values`	`{'sensitivity': (50e-9,10e-9)}`	Python dictionary; first field: nominal value; second field: maximum deviation
`assert_nominal_values`	`0`	warn only
	`1`	exclude force curves conflicting with the limits defined in `nominal_values`; default
`unit_x`	`'um'`	unit of the extension data (only used for plots)
`unit_y`	`'pN'`	unit of the force data (only used for plots)
Baseline correction		
`fit_baseline`	`0`	do not correct baseline
	`1`	subtract a linear baseline; default
	`2`	subtract a quadratic baseline
`baseline_max_rel_local_RSS`	`2.5`	relative threshold for the termination of the baseline fit; default: `2.0`
`baseline_fit_min_width`	`4.0`	minimum baseline fit length for a curve to be further analysed (in units of the extension data)
`baseline_step_len`	`100`	step length for the iterative baseline fit (default: automatic)
`baseline_return_to_local_min`	`0`	the baseline fit is performed up to the point, where it is terminated
	`1`	the baseline fit is performed up to the last local minimum of the RSS after termination; default
De-noising		
`denoising_method`	`'renoir'`	use ReNoiR for noise reduction
	`'gauss'`	use Gaussian smoothing; default
	`'savgol'`	use the Savitzky-Golay filter
`denoising_param`	`4.1`	primary parameter for the noise reduction filter

		or user-defined Python function
denoising_param2	20.8	secondary parameter for the noise reduction filter or user-defined Python function
denoising_recursions	1	number of recursions (ReNoiR only); default: 1
denoising_wavelet	'haar'	name of the wavelet used; default: 'haar'
denoising_levels	0	number of levels (ReNoiR only); default: 0 (automatic)
savgol_order	5	order of the polynomial kernel (Savitzky-Golay filter only)
Contact point calibration		
find_contact_pos	0	do not detect the contact point
	1	detect the contact point; default
max_contact_pos	2.0	maximum distance of the contact point from the beginning of the curve; default: do not check
min_contact_force	0	minimum force at the contact point; default: 0
Step detection		
MSF_sigma		Gaussian smoothing of the retrace curve
MSF_window	100	width of the moving fit window (number of samples); default: 1
indicator_margin_lt	100	left margin of the indicator; default: 0
indicator_margin_rt	100	right margin of the indicator; default: 0
indicator_smoothing_sigma	2.5	Gaussian smoothing of the indicator
indicator_threshold	40	step detection sensitivity (absolute value)
indicator_relative_threshold	10	step detection sensitivity (relative to noise level after de-noising); default: 5
max_steps	20	maximum number of detected steps
step_fit_proximity_lt	None	fitting of left step flank starts at next local minimum in the denoised retrace curve; default
	0 – 1	proximity between step and left fit window
step_fit_proximity_rt	None	fitting of right step flank starts at the next local maximum in the denoised retrace curve; default
	0 – 1	proximity between step and right fit window
step_fit_min_len	10	minimum size of the window for step fits
step_fit_max_len	500	maximum size of the window for step fits; default: no limit
step_measuring_lt	0	use denoised retrace curve to determine left step flank; default
	1	use extrapolated linear fit of the left step flank if possible and otherwise denoised retrace curve
	2	only use fits
step_measuring_rt	0	use denoised retrace curve to determine right step flank; default
	1	use extrapolated linear fit of the right step flank if possible and otherwise denoised retrace curve
	2	only use fits
step_min_height	15	minimum step height; default: do not check
step_min_width	10	minimum step width; default: do not check
step_max_width	10	maximum step width; default: do not check
step_min_slope	250	minimum average step slope (height / width); default: do not check
Fitting		
indentation_fit	None	do not perform fit; default
	0	fit indentation slopes at contact point

	> 0	fit slopes at specified indentation force
indentation_curve	0	use retrace curve; default
	1	use trace curve (never denoised)
indentation_lt_fit_width	0.02	width of the indentation fit to the left; 0 means fit up to beginning of the force curve
indentation_rt_fit_width	0.01	width of the indentation fit to the right
indentation_fit_avg_window	3	half size of the averaging window for finding a specified indentation force; default: 0
trace_fit_function		fit function for trace curve from beginning to contact point
trace_fit_init_params		initial fit parameters for trace fit
retrace_fit_function		fit function for retrace curve from contact point to first step (if number of steps is 1)
retrace_fit_init_params		initial fit parameters for retrace fit
single_step_fit_function		fit function for retrace curve from contact point to first and only step (number of steps must be 1); only used if retrace_fit_function is None
single_step_init_params		initial fit parameters for first tether fit
Plotting		
plot_force_curves	0	do not plot force curve diagrams
	1	plot force curve diagrams; default
plot_indicators	0	do not plot indicator diagrams
	1	plot indicator diagrams; default
show_plots	0	do not show plots; default
	1	show plots in a graphical user interface
plot_format	'pdf'	file format for plots; default: 'png'
plot_xmin	-1	minimum extension; default: auto-scale
plot_xmax	20	maximum extension; default: auto-scale
plot_ymin	-500	minimum force; default: auto-scale
plot_ymax	500	maximum force; default: auto-scale
plot_size_force_curves	(640,480)	plot size (pixels)
plot_size_indicator_curves	(640,480)	plot size (pixels)
plot_features	['retrace', 'steps']	list of features to plot into the force curve diagrams (trace, retrace, denoised, baseline, indentation_fit, contact_pos, trace_fit, retrace_fit, steps, step_fits, markers, aux); default: plot all
plot_step_fit_width	0.01	width of the step slope fits; default: use actual fit lengths
Miscellaneous		
verbose	2	level of verbosity (0-2); default: 0
spring_constant	0.1	spring constant used for force calibration if not specified in input file
length_correction	0	do not convert extension to distance; default
	1	convert extension to distance
experiment_type	'WT'	string used to categorise experiments by a user-defined experiment type

4. PARAMALYSER

4.1. Configuration

The Paramalyser uses the same configuration files as the Curvalyser (see section 3.1) and an additional one, which defines the settings needed for statistical evaluation. It is specified by the command line option "-C" (default: *config-paramalyser*). The following configuration parameters are evaluated:

Parameter	Values / example	Description
Input and output		
`fixed_input_dir`	`'output2'`	input directory; default: Curvalyser output directory
`fixed_output_dir`	`'output2/stats'`	output directory; default: `statistics_dir` in Curvalyser output directory
`statistics_dir`	`'stats'`	relative output directory; default: *statistics*
`multiplier_x`	`1e6`	multiplier for rescaling the extension data
`multiplier_y`	`1e12`	multiplier for rescaling the force data
`experiment_types`	`['WT','A2','B1']`	selection and order of experiment types to be analysed
`analyse_all`	`0`	analysed only experiment types specified by `experiment_types` in the given order
	`1`	analyse all experiment types; the order specified by `experiment_types` is still regarded
`ignore_ids`	`['001','002']`	experiment IDs to ignore
`curves_range_start`	`100`	first curve to include (counting starts with 1; negative value: count from the end); can be overwritten by `paramalyser_curves_range_start` in Curvalyser configuration file
`curves_range_stop`	`200`	last curve to include (counting starts with 1; negative value: count from the end); can be overwritten by `paramalyser_curves_range_stop` in Curvalyser configuration file
`curves_range_step`	`10`	step size (negative value: go backwards); default: auto +1/-1; can be overwritten by `paramalyser_curves_range_step` in Curvalyser configuration file
`autosplit_size`	`10`	split curves into chunks of the given size
`autosplit_shift`	`10`	shift between successive auto-split chunks
Tasks		
`tasks`	`['CDF','CDF_cum']`	list of tasks to perform (see section 4.3); default: all
`plot_params`	`['force','work']`	list of parameters to analyse within each task (see section 4.4); default: all
`plot_steps`	`['first','last']`	list of subset plots to create (`all`: all curves, `single`: only curves with exactly one step, `first`: only first step, `last`: only last step); default: all
Filters		
`step_filter_limits`	`{'height': (10,None), 'work': (0,100)}`	Python dictionary of criteria to filter steps (tuple of minimum and maximum for each parameter to be checked); all criteria must be met
`curve_filter_limits`	`{'noise_sigma': (None,10), 'steps_ctr': (1,1)}`	Python dictionary of criteria to filter curves and steps (tuple of minimum and maximum for each parameter to be checked); all criteria must be met; applied after `step_filter_limits`

Plotting		
`plot_ranges`	`{'work': (0,100)}`	Python dictionary of default plot ranges (tuple of minimum and maximum for each parameter)
`histogram_ranges`		like `plot_ranges`, but for histograms
`boxplot_ranges`		like `plot_ranges`, but for boxplots
`scatterplot_ranges`		like `plot_ranges`, but for scatterplots
`plot_size`	`(800,600)`	default plot size (tuple of width and height)
`plot_size_histogram`		like `plot_size`, but for histograms
`plot_size_CDF`		like `plot_size`, but for CDFs
`plot_size_scatter`		like `plot_size`, but for scatterplots
`plot_size_boxplot`		like `plot_size`, but for boxplots
`plot_size_errorbar`		like `plot_size`, but for errorbar plots
`plot_size_errorbar_cum`		like `plot_size`, but for cumulated errorbar plots
`plot_size_errorbar_corr`		like `plot_size`, but for correlated errorbar plots
`histogram_bins`	`100`	number of histogram bins; default: `50`
`histogram_ylim`	`(0,1)`	vertical range of histograms (tuple of minimum and maximum)
`histogram_norm_curves`	`0`	do not normalise histograms over curve-specific parameters; default
	`1`	normalise to one
	`2`	normalise to average number of steps
	`3`	normalise to adhesion rate
`histogram_norm_steps`	`0`	do not normalise histograms over step-specific parameters; default
	`1`	normalise to one
	`2`	normalise to average number of steps
	`3`	normalise to adhesion rate
`CDF_histogram_bins`	`100`	number of histogram bins for CDF plots (default: sum of counts)
`experiment_type_labels`	`{'WT': 'wild-type'}`	Python dictionary defining custom labels for experiment types
`experiment_type_colors`	`{'WT': 'blue'}`	Python dictionary defining custom colours for experiment types
Miscellaneous		
`cumulation_mode`	`0`	cumulate data for errorbar plots globally
	`1`	cumulate and calculate averages/medians/modals by experiment type; default
`steps_ctr_averaging`	`0`	consider all curves to calculate the average number of steps
	`1`	consider only adhesive curves; default
`output_format`	`'text'`	save data files in standard text format; default
	`'igor'`	save data files in an Igor-compatible text format
`custom_script`	`'custom.py'`	custom script to be executed
`curvalyser_config`		Python dictionary of Curvalyser configuration parameters (overwrites other settings)

4.2. Program execution

The Paramalyser is invoked the same way as the Curvalyser (see section 3.2). Example:

```
> python Paramalyser.py -e WT
```

It understands these command line options:

Option	Example	Description
-h		show a help message
-C	-C pconfig.txt	set the configuration file to be used; default: *config-paramalyser*
-c	-c"exp_id=='001'"	select experiments by a Python expression
-e	-e WT,A2,B1	selection and order of experiment types (same as configuration parameter experiment_types)
-a		analyse all experiment types (same as configuration parameter analyse_all); default: only specified types
-i	-i output2	set the input directory (same as configuration parameter fixed_input_dir); default: Curvalyser output directory
-o	-o output2/stats	set the output directory (same as configuration parameter fixed_output_dir); default: statistics_dir in Curvalyser output directory
-t	-t CDF,CDF_cum	same as configuration parameter tasks
-p	-p force,work	same as configuration parameter plot_params
-s	-s first,last	same as configuration parameter plot_steps
-V		display program version

4.3. Tasks

Data can either be analysed per-experiment or cumulated over all experiments of the same type. The following tasks can be performed (specified by tasks, see section 4.1):

Task	Description	Output directory
data	save parameters to text files	*data*
data_cum	save cumulated parameters to text files	*cumulated data*
histos	create histograms	*histograms/**
histos_cum	create histograms of cumulated data	*histograms*
CDF	plot cumulative distribution function (CDF)	*CDFs*
CDF_cum	plot cumulative distribution function (CDF) of cumulated data	*cumulated CDFs*
CDF_data_cum	save the 50% values of the cumulated CDF plots to text files	*cumulated data*
scatter	create scatter plots	*scatter plots/**
scatter_cum	create cumulated scatter plots	*scatter plots*
boxplots	create boxplots	*boxplots*
boxplots_cum	create cumulated boxplots	*cumulated boxplots*
averages	plot average values	*averages*
averages_cum	plot cumulated average values	*cumulated averages*
averages_data	save averages to text files	*data*
averages_data_cum	save cumulated averages to text files	*cumulated data*
averages_boxplots	create boxplots of average values	*boxplots of averages*
medians	plot medians	*medians*
medians_cum	plot cumulated medians	*cumulated medians*
medians_data	save medians to text files	*data*
medians_data_cum	save cumulated medians to text files	*cumulated data*

medians_boxplots	create boxplots of medians	*boxplots of medians*
modals	plot modals	*modals*
modals_cum	plot cumulated modals	*cumulated modals*
modals_data	save modals to text files	*data*
modals_data_cum	save cumulated modals to text files	*cumulated data*
modal_boxplots	create boxplots of modals	*boxplots of modals*
other	plot adhesion rates and average number of steps	.
tests	calculate Mann-Whitney and Kruskal-Wallis tests	.
custom	execute the custom script defined by custom_script	

4.4. Parameters

The following parameters can be analysed (specified by plot_params, see section 4.1):

Parameter	Description
Curve-specific	
steps_ctr	number of steps
peak_pos	sample number of the peak force (global minimum/maximum force)
peak_extension	extension of the peak force
peak_force	peak force
indent_force	indentation force
work	work (area between the baseline and the retrace curve)
bl_itcpt	interception of the baseline
bl_slope	slope of the baseline
bl_crvtr	curvature of the baseline (in case of a quadratic fit)
bl_fit_len	length of the baseline fit (number of samples)
bl_avg_RSS	average residual sum of squares of the baseline fit
contact_pos	sample number of the contact point
indent_slope_l	left indentation slope of the retrace curve
indent_slope_r	right indentation slope of the retrace curve
indent_slope_r_normed	like indent_slope_r, but divided by the number of steps
trace_fit1	parameters obtained by the custom fit function trace_fit_function
trace_fit2	
trace_fit3	
retrace_fit1	parameters obtained by the custom fit function retrace_fit_function
retrace_fit2	
retrace_fit3	
noise_sigma	estimated standard deviation of the noise
denoising_param	value of the actually used primary de-noising parameter
denoising_param2	value of the actually used secondary de-noising parameter
indicator_thld	value of the actually used indicator threshold
Step-specific	
lt_edge	sample number of the left (lower) edge of the step
lmax_pos	sample number of the local maximum position
rt_edge	sample number of the right (upper) edge of the step
extension	extension of the left (lower) edge of the step
force	force at the left (lower) edge of the step
height	relative height of the step (difference of the forces at the upper and lower edge)
avg_slope	average value of the indicator
max_slope	maximum value of the indicator
plateau_slope_l	fitted slope on the left of the step

`plateau_slope_r`	fitted slope on the right of the step
`stiffness`	`height` / `extension`

5. ADDITIONAL TOOLS

plot_force_curves.py creates plots of force curves similar to the Curvalyser, but considers the Paramalyser filter settings. Removed steps are drawn as small symbols and the ones passing all filters as big symbols. This allows for a visual control of the filtering process. Usage is similar to the other programs:

```
> python plot_force_curves.py 001
```

convert_jpk_to_crv.py converts force curves in the old JPK format (**.out*) to space-saving Curvalyser files (**.crv*). The output directory and the input files are passed as parameters:

```
> python convert_jpk_to_crv.py force_curves data/*.out
```

6. REFERENCES

1. **Opfer, J.** Single-molecule force spectroscopy studies of integrin-mediated cell signaling. *PhD thesis.* 2012, 2.

2. **Gonzalez, R C and Woods, R E.** *Digital Image Processing, 2nd ed.* s.l. : Addison-Wesley Longman Publishing, 1992.

3. **Savitzky, A and Golay, M J E.** Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Analytical Chemistry.* 1964, 36:1627.

4. **Opfer, J and Gottschalk, K.** Identifying discrete states of a biological system by a novel step detection algorithm. *PLoS ONE.* 2012, Vol. 7, 11, p. e45896.