
SBpipe documentation

Release 3.13.0

Piero Dalle Pezze and Nicolas Le Novère

Mar 24, 2017

CONTENTS

1	Source code	1
1.1	Python modules	1
1.1.1	sbpipe package	1
2	Indices	19
	Python Module Index	21

SOURCE CODE

Python modules

sbpipe package

Subpackages

sbpipe.pl package

Subpackages

sbpipe.pl.create package

Submodules

sbpipe.pl.create.newproj module

```
class sbpipe.pl.create.newproj.NewProj (models_folder='Models',          work-
                                         ing_folder='Results')
    Bases: sbpipe.pl.pipeline.Pipeline (page 7)
```

This module initialises the folder tree for a new project.

Parameters

- **models_folder** – the folder containing the models
- **working_folder** – the folder to store the results

run (*project_name*)

Create a project directory tree.

Parameters **project_name** – the name of the project

Returns 0

Module contents

sbpipe.pl.pe package

Submodules

sbpipe.pl.pe.parest module

```
class sbpipe.pl.pe.parest.ParEst (models_folder='Models',          working_folder='Results',
                                  sim_data_folder='param_estim_data',
                                  sim_plots_folder='param_estim_plots')
```

Bases: *sbpipe.pl.pipeline.Pipeline* (page 7)

This module provides the user with a complete pipeline of scripts for running model parameter estimations

```
classmethod analyse_data (simulator, model, inputdir, outputdir, fileout_final_estims,
                          fileout_all_estims, fileout_param_estim_details, file-
                          out_param_estim_summary, sim_plots_dir, best_fits_percent,
                          data_point_num, cluster='local', plot_2d_66cl_corr=False,
                          plot_2d_95cl_corr=False, plot_2d_99cl_corr=False,
                          logspace=True, scientific_notation=True)
```

The second pipeline step: data analysis.

Parameters

- **simulator** – the name of the simulator (e.g. Copasi)
- **model** – the model name
- **inputdir** – the directory containing the simulation data
- **outputdir** – the directory to store the results
- **fileout_final_estims** – the name of the file containing final parameter sets with the objective value
- **fileout_all_estims** – the name of the file containing all the parameter sets with the objective value
- **fileout_param_estim_details** – the name of the file containing the detailed statistics for the estimated parameters
- **fileout_param_estim_summary** – the name of the file containing the summary for the parameter estimation
- **sim_plots_dir** – the directory of the simulation plots
- **best_fits_percent** – the percent to consider for the best fits
- **data_point_num** – the number of data points
- **cluster** – local, lsf for Load Sharing Facility, sge for Sun Grid Engine.
- **plot_2d_66cl_corr** – True if 2 dim plots for the parameter sets within 66% should be plotted
- **plot_2d_95cl_corr** – True if 2 dim plots for the parameter sets within 95% should be plotted
- **plot_2d_99cl_corr** – True if 2 dim plots for the parameter sets within 99% should be plotted
- **logspace** – True if parameters should be plotted in log space
- **scientific_notation** – True if axis labels should be plotted in scientific notation

Returns True if the task was completed successfully, False otherwise.

```
classmethod generate_data (simulator, model, inputdir, cluster, local_cpus, runs, outputdir,
                           sim_data_dir)
```

The first pipeline step: data generation.

Parameters

- **simulator** – the name of the simulator (e.g. Copasi)

- **model** – the model to process
- **inputdir** – the directory containing the model
- **cluster** – local, lsf for load sharing facility, sge for sun grid engine
- **local_cpus** – the number of cpu
- **runs** – the number of fits to perform
- **outputdir** – the directory to store the results
- **sim_data_dir** – the directory containing the simulation data sets

Returns True if the task was completed successfully, False otherwise.

classmethod generate_report (*model, outputdir, sim_plots_folder*)

The third pipeline step: report generation.

Parameters

- **model** – the model name
- **outputdir** – the directory to store the report
- **sim_plots_folder** – the folder containing the plots

Returns True if the task was completed successfully, False otherwise.

parse (*my_dict*)

run (*config_file*)

Module contents

sbpipe.pl.ps1 package

Submodules

sbpipe.pl.ps1.parscan1 module

```
class sbpipe.pl.ps1.parscan1.ParScan1 (models_folder='Models', working_folder='Results',
                                         sim_data_folder='single_param_scan_data',
                                         sim_plots_folder='single_param_scan_plots')
```

Bases: *sbpipe.pl.pipeline.Pipeline* (page 7)

This module provides the user with a complete pipeline of scripts for computing single parameter scans.

```
classmethod analyse_data (model, knock_down_only, outputdir, sim_data_folder,
                          sim_plots_folder, runs, local_cpus, percent_levels, min_level,
                          max_level, levels_number, homogeneous_lines, cluster='local',
                          xaxis_label='', yaxis_label='')
```

The second pipeline step: data analysis.

Parameters

- **model** – the model name
- **knock_down_only** – True for knock down simulation, false if also scanning over expression.
- **outputdir** – the directory containing the results
- **sim_data_folder** – the folder containing the simulated data sets
- **sim_plots_folder** – the folder containing the generated plots
- **runs** – the number of simulations

- **local_cpus** – the number of cpus
- **percent_levels** – True if the levels are percents.
- **min_level** – the minimum level
- **max_level** – the maximum level
- **levels_number** – the number of levels
- **homogeneous_lines** – True if generated line style should be homogeneous
- **cluster** – local, lsf for Load Sharing Facility, sge for Sun Grid Engine.
- **xaxis_label** – the name of the x axis (e.g. Time [min])
- **yaxis_label** – the name of the y axis (e.g. Level [a.u.])

Returns True if the task was completed successfully, False otherwise.

classmethod generate_data (*simulator, model, scanned_par, cluster, local_cpus, runs, simulate_intervals, single_param_scan_intervals, inputdir, outputdir*)

The first pipeline step: data generation.

Parameters

- **simulator** – the name of the simulator (e.g. Copasi)
- **model** – the model to process
- **scanned_par** – the scanned parameter
- **cluster** – local, lsf for Load Sharing Facility, sge for Sun Grid Engine.
- **local_cpus** – the number of CPU.
- **runs** – the number of model simulation
- **simulate_intervals** – the time step of each simulation
- **single_param_scan_intervals** – the number of scans to perform
- **inputdir** – the directory containing the model
- **outputdir** – the directory to store the results

Returns True if the task was completed successfully, False otherwise.

classmethod generate_report (*model, scanned_par, outputdir, sim_plots_folder*)

The third pipeline step: report generation.

Parameters

- **model** – the model name
- **scanned_par** – the scanned parameter
- **outputdir** – the directory containing the report
- **sim_plots_folder** – the folder containing the plots

Returns True if the task was completed successfully, False otherwise.

parse (*my_dict*)

run (*config_file*)

Module contents

sbpipe.pl.ps2 package

Submodules

sbpipe.pl.ps2.parscan2 module

```
class sbpipe.pl.ps2.parscan2.ParScan2(models_folder='Models', working_folder='Results',
                                       sim_data_folder='double_param_scan_data',
                                       sim_plots_folder='double_param_scan_plots')
```

Bases: [sbpipe.pl.pipeline.Pipeline](#) (page 7)

This module provides the user with a complete pipeline of scripts for computing double parameter scans.

```
classmethod analyse_data(model, scanned_par1, scanned_par2, inputdir, outputdir, cluster='local', local_cpus=1, runs=1)
```

The second pipeline step: data analysis.

Parameters

- **model** – the model name
- **scanned_par1** – the first scanned parameter
- **scanned_par2** – the second scanned parameter
- **inputdir** – the directory containing the simulated data sets to process
- **outputdir** – the directory to store the performed analysis
- **cluster** – local, lsf for Load Sharing Facility, sge for Sun Grid Engine.
- **local_cpus** – the number of CPU.
- **runs** – the number of model simulation

Returns True if the task was completed successfully, False otherwise.

```
classmethod generate_data(simulator, model, sim_length, inputdir, outputdir, cluster, local_cpus, runs)
```

The first pipeline step: data generation.

Parameters

- **simulator** – the name of the simulator (e.g. Copasi)
- **model** – the model to process
- **sim_length** – the length of the simulation
- **inputdir** – the directory containing the model
- **outputdir** – the directory to store the results
- **cluster** – local, lsf for Load Sharing Facility, sge for Sun Grid Engine.
- **local_cpus** – the number of CPU.
- **runs** – the number of model simulation

Returns True if the task was completed successfully, False otherwise.

```
classmethod generate_report(model, scanned_par1, scanned_par2, outputdir, sim_plots_folder)
```

The third pipeline step: report generation.

Parameters

- **model** – the model name
- **scanned_par1** – the first scanned parameter
- **scanned_par2** – the second scanned parameter
- **outputdir** – the directory containing the report

- **sim_plots_folder** – the folder containing the plots.

Returns True if the task was completed successfully, False otherwise.

parse (*my_dict*)

run (*config_file*)

Module contents

sbpipe.pl.sim package

Submodules

sbpipe.pl.sim.sim module

class sbpipe.pl.sim.sim.**Sim** (*models_folder='Models', working_folder='Results',
sim_data_folder='simulate_data', sim_plots_folder='simulate_plots'*)
Bases: *sbpipe.pl.pipeline.Pipeline* (page 7)

This module provides the user with a complete pipeline of scripts for running model simulations

classmethod **analyse_data** (*model, inputdir, outputdir, sim_plots_dir, exp_dataset,
plot_exp_dataset, cluster='local', xaxis_label='', yaxis_label=''*)

The second pipeline step: data analysis.

Parameters

- **model** – the model name
- **inputdir** – the directory containing the data to analyse
- **outputdir** – the output directory containing the results
- **sim_plots_dir** – the directory to save the plots
- **exp_dataset** – the full path of the experimental data set
- **plot_exp_dataset** – True if the experimental data set should also be plotted
- **cluster** – local, lsf for Load Sharing Facility, sge for Sun Grid Engine.
- **xxaxis_label** – the label for the x axis (e.g. Time [min])
- **yaxis_label** – the label for the y axis (e.g. Level [a.u.])

Returns True if the task was completed successfully, False otherwise.

classmethod **generate_data** (*simulator, model, inputdir, outputdir, cluster='local', lo-
cal_cpus=2, runs=1*)

The first pipeline step: data generation.

Parameters

- **simulator** – the name of the simulator (e.g. Copasi)
- **model** – the model to process
- **inputdir** – the directory containing the model
- **outputdir** – the directory containing the output files
- **cluster** – local, lsf for Load Sharing Facility, sge for Sun Grid Engine.
- **local_cpus** – the number of CPUs.
- **runs** – the number of model simulation

Returns True if the task was completed successfully, False otherwise.

classmethod `generate_report` (*model*, *outputdir*, *sim_plots_folder*)

The third pipeline step: report generation.

Parameters

- **model** – the model name
- **outputdir** – the output directory to store the report
- **sim_plots_folder** – the folder containing the plots

Returns True if the task was completed successfully, False otherwise.

parse (*my_dict*)

run (*config_file*)

Module contents

Submodules

sbpipe.pl.pipeline module

class `sbpipe.pl.pipeline.Pipeline` (*models_folder*=*'Models'*, *working_folder*=*'Results'*, *sim_data_folder*=*'sim_data'*, *sim_plots_folder*=*'sim_plots'*)

Bases: `object`

Generic pipeline.

Parameters

- **models_folder** – the folder containing the models
- **working_folder** – the folder to store the results
- **sim_data_folder** – the folder to store the simulation data
- **sim_plots_folder** – the folder to store the graphic results

get_models_folder ()

Return the folder containing the models.

Returns the models folder.

get_sim_data_folder ()

Return the folder containing the in-silico generated data sets.

Returns the folder of the simulated data sets.

get_sim_plots_folder ()

Return the folder containing the in-silico generated plots.

Returns the folder of the simulated plots.

classmethod `get_simul_obj` (*simulator*)

Return the simulator object if this exists. Otherwise throws an exception. The simulator name starts with an upper case letter. Each simulator is in a package within *sbpipe.simulator*.

Parameters **simulator** – the simulator name

Returns the simulator object.

get_working_folder ()

Return the folder containing the results.

Returns the working folder.

classmethod `load` (*config*)

Safely load a YAML configuration file and return its structure as a dictionary object.

Parameters `config` – a YAML configuration file

Returns the dictionary structure of the configuration file

Raise `yaml.YAMLError` if the config cannot be loaded.

parse (*config_dict*)

Read a dictionary structure containing the pipeline configuration. This method is abstract.

Returns a tuple containing the configuration

run (*config_file*)

Run the pipeline.

Parameters `config_file` – a configuration file for this pipeline.

Returns True if the pipeline was executed correctly, False otherwise.

Module contents

sbpipe.report package

Submodules

sbpipe.report.latex_reports module

`sbpipe.report.latex_reports.get_latex_header` (*pdftitle='SBpipe report', title='SBpipe report', abstract='Generic report.'*)

Initialize a Latex header with a title and an abstract.

Parameters

- **pdftitle** – the pdftitle for the LaTeX header
- **title** – the title for the LaTeX header
- **abstract** – the abstract for the LaTeX header

Returns the LaTeX header

`sbpipe.report.latex_reports.latex_report` (*outputdir, sim_plots_folder, model_noext, filename_prefix, caption=False*)

Generate a generic report.

Parameters

- **outputdir** – the output directory
- **sim_plots_folder** – the folder containing the simulated plots
- **model_noext** – the model name
- **filename_prefix** – the prefix for the LaTeX file
- **caption** – True if figure captions (=figure file name) should be added

`sbpipe.report.latex_reports.latex_report_pe` (*outputdir, sim_plots_folder, model_noext, filename_prefix*)

Generate a report for a parameter estimation task.

Parameters

- **outputdir** – the output directory
- **sim_plots_folder** – the folder containing the simulated plots

- **model_noext** – the model name
- **filename_prefix** – the prefix for the LaTeX file

```
sbpipe.report.latex_reports.latex_report_ps1 (outputdir,          sim_plots_folder,
                                              filename_prefix,      model_noext,
                                              scanned_par)
```

Generate a report for a single parameter scan task.

Parameters

- **outputdir** – the output directory
- **sim_plots_folder** – the folder containing the simulated plots
- **filename_prefix** – the prefix for the LaTeX file
- **model_noext** – the model name
- **scanned_par** – the scanned parameter

```
sbpipe.report.latex_reports.latex_report_ps2 (outputdir,          sim_plots_folder,
                                              filename_prefix,      model_noext,
                                              scanned_par1, scanned_par2)
```

Generate a report for a double parameter scan task.

Parameters

- **outputdir** – the output directory
- **sim_plots_folder** – the folder containing the simulated plots
- **filename_prefix** – the prefix for the LaTeX file
- **model_noext** – the model name
- **scanned_par1** – the 1st scanned parameter
- **scanned_par2** – the 2nd scanned parameter

```
sbpipe.report.latex_reports.latex_report_sim (outputdir,          sim_plots_folder,
                                              model_noext, filename_prefix)
```

Generate a report for a time course task.

Parameters

- **outputdir** – the output directory
- **sim_plots_folder** – the folder containing the simulated plots
- **model_noext** – the model name
- **filename_prefix** – the prefix for the LaTeX file

```
sbpipe.report.latex_reports.pdf_report (outputdir, filename)
```

Generate a PDF report from LaTeX report using pdflatex.

Parameters

- **outputdir** – the output directory
- **filename** – the LaTeX file name

Module contents

sbpipe.simul package

Subpackages

sbpipe.simul.copasi package

Submodules

sbpipe.simul.copasi.copasi module

```
class sbpipe.simul.copasi.copasi.Copasi
    Bases: sbpipe.simul.simul.Simul (page 11)
    Copasi simulator.
    pe (model, inputdir, cluster, local_cpus, runs, outputdir, sim_data_dir, output_msg=False)
    ps1 (model, scanned_par, simulate_intervals, single_param_scan_intervals, inputdir, outputdir,
        cluster='local', local_cpus=1, runs=1, output_msg=False)
    ps2 (model, sim_length, inputdir, outputdir, cluster='local', local_cpus=1, runs=1, out-
        put_msg=False)
    replace_str_in_report (report)
    sim (model, inputdir, outputdir, cluster='local', local_cpus=1, runs=1, output_msg=False)
```

Module contents

sbpipe.simul.python package

Submodules

sbpipe.simul.python.python module

```
class sbpipe.simul.python.python.Python
    Bases: sbpipe.simul.pl_simul.PLSimul (page 10)
    Python Simulator.
```

Module contents

Submodules

sbpipe.simul.pl_simul module

```
class sbpipe.simul.pl_simul.PLSimul (lang, lang_err_msg, options)
    Bases: sbpipe.simul.simul.Simul (page 11)
    A generic simulator for models coded in a programming language.
    get_lang ()
        Return the programming language name :return: the name
    get_lang_err_msg ()
        Return the error if the programming language is not found :return: the error message
    get_lang_options ()
        Return the options for the programming language command :return: the options. Return None, if no
        options are used.
    pe (model, inputdir, cluster, local_cpus, runs, outputdir, sim_data_dir, output_msg=False)
    ps1 (model, scanned_par, simulate_intervals, single_param_scan_intervals, inputdir, outputdir,
        cluster='local', local_cpus=1, runs=1, output_msg=False)
```

```

ps2 (model, sim_length, inputdir, outputdir, cluster='local', local_cpus=1, runs=1, output_msg=False)
replace_str_in_report (report)
sim (model, inputdir, outputdir, cluster='local', local_cpus=1, runs=1, output_msg=False)

```

sbpipe.simul.simul module

class `sbpipe.simul.simul.Simul`

Bases: `object`

Generic simulator.

get_all_fits (*path_in='', path_out='', filename_out='all_estimates.csv'*)

Collect all the parameter estimates. Results are stored in `filename_out`.

Parameters

- **path_in** – the path to the input files
- **path_out** – the path to the output files
- **filename_out** – a global file containing all fits from independent parameter estimations.

Returns the number of retrieved files

get_best_fits (*path_in='', path_out='', filename_out='final_estimates.csv'*)

Collect the final parameter estimates. Results are stored in `filename_out`.

Parameters

- **path_in** – the path to the input files
- **path_out** – the path to the output files
- **filename_out** – a global file containing the best fits from independent parameter estimations.

Returns the number of retrieved files

pe (*model, inputdir, cluster, local_cpus, runs, outputdir, sim_data_dir, output_msg=False*)
parameter estimation.

Parameters

- **model** – the model to process
- **inputdir** – the directory containing the model
- **cluster** – local, lsf for load sharing facility, sge for sun grid engine
- **local_cpus** – the number of cpu
- **runs** – the number of fits to perform
- **outputdir** – the directory to store the results
- **sim_data_dir** – the directory containing the simulation data sets
- **output_msg** – print the output messages on screen (available for `cluster='local'` only)

ps1 (*model, scanned_par, simulate_intervals, single_param_scan_intervals, inputdir, outputdir, cluster='local', local_cpus=1, runs=1, output_msg=False*)
Single parameter scan.

Parameters

- **model** – the model to process

- **scanned_par** – the scanned parameter
- **simulate_intervals** – the time step of each simulation
- **single_param_scan_intervals** – the number of scans to perform
- **inputdir** – the directory containing the model
- **outputdir** – the directory to store the results
- **cluster** – local, lsf for Load Sharing Facility, sge for Sun Grid Engine.
- **local_cpus** – the number of CPU used.
- **runs** – the number of model simulation
- **output_msg** – print the output messages on screen (available for cluster='local' only)

ps1_postproc (*model, scanned_par, simulate_intervals, single_param_scan_intervals, outputdir*)

Perform post processing organisation to single parameter scan report files.

Parameters

- **model** – the model to process
- **scanned_par** – the scanned parameter
- **simulate_intervals** – the time step of each simulation
- **single_param_scan_intervals** – the number of scans to perform
- **outputdir** – the directory to store the results

ps2 (*model, sim_length, inputdir, outputdir, cluster='local', local_cpus=1, runs=1, output_msg=False*)

Double paramter scan.

Parameters

- **model** – the model to process
- **sim_length** – the length of the simulation
- **inputdir** – the directory containing the model
- **outputdir** – the directory to store the results
- **cluster** – local, lsf for Load Sharing Facility, sge for Sun Grid Engine.
- **local_cpus** – the number of CPU.
- **runs** – the number of model simulation
- **output_msg** – print the output messages on screen (available for cluster='local' only)

ps2_postproc (*model, sim_length, outputdir*)

Perform post processing organisation to double parameter scan report files.

Parameters

- **model** – the model to process
- **sim_length** – the length of the simulation
- **outputdir** – the directory to store the results

replace_str_in_report (*report*)

Replaces strings in a report file.

Parameters **report** – a report file with its absolute path

sim(*model*, *inputdir*, *outputdir*, *cluster*='local', *local_cpus*=1, *runs*=1, *output_msg*=False)
Time course simulator.

Parameters

- **model** – the model to process
- **inputdir** – the directory containing the model
- **outputdir** – the directory containing the output files
- **cluster** – local, lsf for Load Sharing Facility, sge for Sun Grid Engine.
- **local_cpus** – the number of CPU.
- **runs** – the number of model simulation
- **output_msg** – print the output messages on screen (available for cluster='local' only)

Module contents

sbpipe.utils package

Submodules

sbpipe.utils.io module

sbpipe.utils.io.files_with_pattern_recur(*folder*, *pattern*)
Return all files with a certain pattern in folder+subdirectories

Parameters

- **folder** – the folder to search for
- **pattern** – the string to search for

Returns the files containing the pattern.

sbpipe.utils.io.get_pattern_pos(*pattern*, *filename*)
Return the line number (as string) of the first occurrence of a pattern in filename

Parameters

- **pattern** – the pattern of the string to find
- **filename** – the file name containing the pattern to search

Returns the line number containing the pattern or “-1” if the pattern was not found

sbpipe.utils.io.refresh(*path*, *file_pattern*)
Clean and create the folder if this does not exist.

Parameters

- **path** – the path containing the files to remove
- **file_pattern** – the string pattern of the files to remove

sbpipe.utils.io.remove_file_silently(*filename*)
Remove a filename silently, without reporting warnings or error messages. This is not really needed by Linux, but Windows sometimes fails to remove the file even if this exists.

Parameters **filename** – the file to remove

sbpipe.utils.io.replace_str_in_file(*filename_out*, *old_string*, *new_string*)
Replace a string with another in filename_out

Parameters

- **filename_out** – the output file
- **old_string** – the old string that should be replaced
- **new_string** – the new string replacing old_string

`sbpipe.utils.io.replace_str_in_report (report)`

Replace nasty strings in COPASI report file.

Parameters **report** – the report

`sbpipe.utils.io.write_mat_on_file (path, filename_out, data)`

Write the matrix results stored in data to filename_out

Parameters

- **path** – the path to filename_out
- **filename_out** – the output file
- **data** – the data to store in a file

sbpipe.utils.parcomp module

`sbpipe.utils.parcomp.call_proc (params)`

Run a command using Python subprocess.

Parameters **params** – A tuple containing (the string of the command to run, the command id)

`sbpipe.utils.parcomp.is_output_file_clean (filename, stream_type='standard output')`

Check whether a file contains the string 'error' or 'warning'. If so a message is printed.

Parameters

- **filename** – a file
- **stream_type** – 'stderr' for standard error, 'stdout' for standard output.

Returns True

`sbpipe.utils.parcomp.parcomp (cmd, cmd_iter_substr, output_dir, cluster='local', runs=1, local_cpus=1, output_msg=False)`

Generic function to run a command in parallel

Parameters

- **cmd** – the command string to run in parallel
- **cmd_iter_substr** – the substring of the iteration number. This will be replaced in a number automatically
- **output_dir** – the output directory
- **cluster** – the cluster type among local (Python multiprocessing), sge, or lsf
- **runs** – the number of runs
- **local_cpus** – the number of cpus to use at most
- **output_msg** – print the output messages on screen (available for cluster='local' only)

Returns True if the computation succeeded.

`sbpipe.utils.parcomp.quick_debug (cmd, out_dir, err_dir)`

Look up for *error* and *warning* in the standard output and error files. A simple debugging function checking the generated log files. We don't stop the computation because it happens that these messages are more *warnings* than real errors.

Parameters

- **cmd** – the executed command

- **out_dir** – the directory containing the standard output files
- **err_dir** – the directory containing the standard error files

Returns True

`sbpipe.utils.parcomp.run_cmd(cmd)`

Run a command using Python subprocess.

Parameters **cmd** – The string of the command to run

`sbpipe.utils.parcomp.run_cmd_block(cmd)`

Run a command using Python subprocess. Block the call until the command has finished.

Parameters **cmd** – A tuple containing the string of the command to run

`sbpipe.utils.parcomp.run_jobs_local(cmd, cmd_iter_substr, runs=1, local_cpus=1, output_msg=False)`

Run jobs using python multiprocessing locally.

Parameters

- **cmd** – the full command to run as a job
- **cmd_iter_substr** – the substring in command to be replaced with a number
- **runs** – the number of runs to execute
- **local_cpus** – The number of available cpus. If local_cpus <=0, only one core will be used.
- **output_msg** – print the output messages on screen (available for cluster_type='local' only)

Returns True

`sbpipe.utils.parcomp.run_jobs_lsf(cmd, cmd_iter_substr, out_dir, err_dir, runs=1)`

Run jobs using a Load Sharing Facility (LSF) cluster.

Parameters

- **cmd** – the full command to run as a job
- **cmd_iter_substr** – the substring in command to be replaced with a number
- **out_dir** – the directory containing the standard output from bsub
- **err_dir** – the directory containing the standard error from bsub
- **runs** – the number of runs to execute

Returns True if the computation succeeded.

`sbpipe.utils.parcomp.run_jobs_sge(cmd, cmd_iter_substr, out_dir, err_dir, runs=1)`

Run jobs using a Sun Grid Engine (SGE) cluster.

Parameters

- **cmd** – the full command to run as a job
- **cmd_iter_substr** – the substring in command to be replaced with a number
- **out_dir** – the directory containing the standard output from qsub
- **err_dir** – the directory containing the standard error from qsub
- **runs** – the number of runs to execute

Returns True if the computation succeeded.

sbpipe.utils.rand module

`sbpipe.utils.rand.get_rand_alphanum_str (length)`

Return a random alphanumeric string

Parameters `length` – the length of the string

Returns the generated string

`sbpipe.utils.rand.get_rand_num_str (length)`

Return a random numeric string

Parameters `length` – the length of the string

Returns the generated string

sbpipe.utils.re_utils module

`sbpipe.utils.re_utils.escape_special_chars (text)`

Escape ^,%, .[,.(, {, } from text :param text: the command to escape special characters inside :return: the command with escaped special characters

`sbpipe.utils.re_utils.nat_sort_key (str)`

The key to sort a list of strings alphanumerically (e.g. “file10” is correctly placed after “file2”)

Parameters `str` – the string to sort alphanumerically in a list of strings

Returns the key to sort strings alphanumerically

Module contents

Submodules

sbpipe.__main__ module

sbpipe.main module

`sbpipe.main.main (argv=None)`

SBpipe main function.

Returns 0 if OK, 1 if trouble

`sbpipe.main.read_file_header (filename)`

Read the first line of a file

Parameters `filename` – the file name to read

Returns the first line

`sbpipe.main.sbpipes (create_project='', simulate='', parameter_scan1='', parameter_scan2='',
parameter_estimation='', logo=False, license=False, nocolor=False,
log_level='', quiet=False, verbose=False)`

SBpipe function.

Parameters

- **create_project** – create a project with the name as argument
- **simulate** – model simulation using a configuration file as argument
- **parameter_scan1** – model one parameter scan using a configuration file as argument
- **parameter_scan2** – model two parameters scan using a configuration file as argument

- **parameter_estimation** – model parameter estimation using a configuration file as argument
- **logo** – True to print the logo
- **license** – True to print the license
- **nocolor** – True to print logging messages without colors
- **log_level** – Set the logging level
- **quiet** – True if quiet (WARNING+)
- **verbose** – True if verbose (DEBUG+)

Returns 0 if OK, 1 if trouble (e.g. a pipeline did not execute correctly).

`sbpipe.main.sbpipe_logo()`

Return sbpipe logo.

Returns sbpipe logo

`sbpipe.main.set_basic_logger(level='INFO')`

Set a basic StreamHandler logger. :param level: the level for this console logger

`sbpipe.main.set_color_logger(level='INFO')`

Replace the current logging.StreamHandler with colorlog.StreamHandler. :param level: the level for this console logger

`sbpipe.main.set_console_logger(new_level='NOTSET', current_level='INFO', no-color=False)`

Set the console logger to a new level if this is different from NOTSET

Parameters

- **new_level** – the new level to set for the console logger
- **current_level** – the current level to set for the console logger
- **nocolor** – True if no colors should be used

`sbpipe.main.set_logger(level='NOTSET', nocolor=False)`

Set the logger :param level: the level for the console logger :param nocolor: True if no colors should be used

sbpipe.sbpipe_config module

`sbpipe.sbpipe_config.isPyPackageInstalled(package)`

Utility checking whether a Python package is installed.

Parameters `package` – a Python package name

Returns True if it is installed, false otherwise.

`sbpipe.sbpipe_config.which(cmd_name)`

Utility equivalent to *which* in GNU/Linux OS.

Parameters `cmd_name` – a command name

Returns return the command name with absolute path if this exists, or None

Module contents

INDICES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

S

- `sbpipe`, 17
- `sbpipe.__main__`, 16
- `sbpipe.main`, 16
- `sbpipe.pl`, 8
 - `create`, 1
 - `create.newproj`, 1
 - `pe`, 3
 - `pe.parest`, 2
 - `pipeline`, 7
 - `ps1`, 4
 - `ps1.parscan1`, 3
 - `ps2`, 6
 - `ps2.parscan2`, 5
 - `sim`, 7
 - `sim.sim`, 6
- `sbpipe.report`, 9
 - `report.latex_reports`, 8
- `sbpipe.sbpipe_config`, 17
- `sbpipe.simul`, 13
 - `simul.copasi`, 10
 - `simul.copasi.copasi`, 10
 - `pl_simul`, 10
 - `python`, 10
 - `python.python`, 10
 - `simul`, 11
- `sbpipe.utils`, 16
 - `utils.io`, 13
 - `utils.parcomp`, 14
 - `utils.rand`, 16
 - `utils.re_utils`, 16

A

analyse_data() (sbpipe.pl.pe.parest.ParEst class method), 2
 analyse_data() (sbpipe.pl.ps1.parscan1.ParScan1 class method), 3
 analyse_data() (sbpipe.pl.ps2.parscan2.ParScan2 class method), 5
 analyse_data() (sbpipe.pl.sim.sim.Sim class method), 6

C

call_proc() (in module sbpipe.utils.parcomp), 14
 Copasi (class in sbpipe.simul.copasi.copasi), 10

E

escape_special_chars() (in module sbpipe.utils.re_utils), 16

F

files_with_pattern_recur() (in module sbpipe.utils.io), 13

G

generate_data() (sbpipe.pl.pe.parest.ParEst class method), 2
 generate_data() (sbpipe.pl.ps1.parscan1.ParScan1 class method), 4
 generate_data() (sbpipe.pl.ps2.parscan2.ParScan2 class method), 5
 generate_data() (sbpipe.pl.sim.sim.Sim class method), 6
 generate_report() (sbpipe.pl.pe.parest.ParEst class method), 3
 generate_report() (sbpipe.pl.ps1.parscan1.ParScan1 class method), 4
 generate_report() (sbpipe.pl.ps2.parscan2.ParScan2 class method), 5
 generate_report() (sbpipe.pl.sim.sim.Sim class method), 6
 get_all_fits() (sbpipe.simul.simul.Simul method), 11
 get_best_fits() (sbpipe.simul.simul.Simul method), 11
 get_lang() (sbpipe.simul.pl_simul.PLSimul method), 10
 get_lang_err_msg() (sbpipe.simul.pl_simul.PLSimul method), 10
 get_lang_options() (sbpipe.simul.pl_simul.PLSimul method), 10

get_latex_header() (in module sbpipe.report.latex_reports), 8
 get_models_folder() (sbpipe.pl.pipeline.Pipeline method), 7
 get_pattern_pos() (in module sbpipe.utils.io), 13
 get_rand_alphanum_str() (in module sbpipe.utils.rand), 16
 get_rand_num_str() (in module sbpipe.utils.rand), 16
 get_sim_data_folder() (sbpipe.pl.pipeline.Pipeline method), 7
 get_sim_plots_folder() (sbpipe.pl.pipeline.Pipeline method), 7
 get_simul_obj() (sbpipe.pl.pipeline.Pipeline class method), 7
 get_working_folder() (sbpipe.pl.pipeline.Pipeline method), 7

I

is_output_file_clean() (in module sbpipe.utils.parcomp), 14
 isPyPackageInstalled() (in module sbpipe.sbpipes_config), 17

L

latex_report() (in module sbpipe.report.latex_reports), 8
 latex_report_pe() (in module sbpipe.report.latex_reports), 8
 latex_report_ps1() (in module sbpipe.report.latex_reports), 9
 latex_report_ps2() (in module sbpipe.report.latex_reports), 9
 latex_report_sim() (in module sbpipe.report.latex_reports), 9
 load() (sbpipe.pl.pipeline.Pipeline class method), 7

M

main() (in module sbpipe.main), 16

N

nat_sort_key() (in module sbpipe.utils.re_utils), 16
 NewProj (class in sbpipe.pl.create.newproj), 1

P

parcomp() (in module sbpipe.utils.parcomp), 14
 ParEst (class in sbpipe.pl.pe.parest), 2

ParScan1 (class in sbpipe.pl.ps1.parscan1), 3
 ParScan2 (class in sbpipe.pl.ps2.parscan2), 5
 parse() (sbpipe.pl.pe.parest.ParEst method), 3
 parse() (sbpipe.pl.pipeline.Pipeline method), 8
 parse() (sbpipe.pl.ps1.parscan1.ParScan1 method), 4
 parse() (sbpipe.pl.ps2.parscan2.ParScan2 method), 6
 parse() (sbpipe.pl.sim.sim.Sim method), 7
 pdf_report() (in module sbpipe.report.latex_reports), 9
 pe() (sbpipe.simul.copasi.copasi.Copasi method), 10
 pe() (sbpipe.simul.pl_simul.PLSimul method), 10
 pe() (sbpipe.simul.simul.Simul method), 11
 Pipeline (class in sbpipe.pl.pipeline), 7
 PLSimul (class in sbpipe.simul.pl_simul), 10
 ps1() (sbpipe.simul.copasi.copasi.Copasi method), 10
 ps1() (sbpipe.simul.pl_simul.PLSimul method), 10
 ps1() (sbpipe.simul.simul.Simul method), 11
 ps1_postproc() (sbpipe.simul.simul.Simul method), 12
 ps2() (sbpipe.simul.copasi.copasi.Copasi method), 10
 ps2() (sbpipe.simul.pl_simul.PLSimul method), 11
 ps2() (sbpipe.simul.simul.Simul method), 12
 ps2_postproc() (sbpipe.simul.simul.Simul method), 12
 Python (class in sbpipe.simul.python.python), 10

Q

quick_debug() (in module sbpipe.utils.parcomp), 14

R

read_file_header() (in module sbpipe.main), 16
 refresh() (in module sbpipe.utils.io), 13
 remove_file_silently() (in module sbpipe.utils.io), 13
 replace_str_in_file() (in module sbpipe.utils.io), 13
 replace_str_in_report() (in module sbpipe.utils.io), 14
 replace_str_in_report()
 (sbpipe.simul.copasi.copasi.Copasi method), 10
 replace_str_in_report()
 (sbpipe.simul.pl_simul.PLSimul method), 11
 replace_str_in_report() (sbpipe.simul.simul.Simul method), 12
 run() (sbpipe.pl.create.newproj.NewProj method), 1
 run() (sbpipe.pl.pe.parest.ParEst method), 3
 run() (sbpipe.pl.pipeline.Pipeline method), 8
 run() (sbpipe.pl.ps1.parscan1.ParScan1 method), 4
 run() (sbpipe.pl.ps2.parscan2.ParScan2 method), 6
 run() (sbpipe.pl.sim.sim.Sim method), 7
 run_cmd() (in module sbpipe.utils.parcomp), 15
 run_cmd_block() (in module sbpipe.utils.parcomp), 15
 run_jobs_local() (in module sbpipe.utils.parcomp), 15
 run_jobs_lsf() (in module sbpipe.utils.parcomp), 15
 run_jobs_sge() (in module sbpipe.utils.parcomp), 15

S

sbpipe (module), 17
 sbpipe() (in module sbpipe.main), 16
 sbpipe.__main__ (module), 16
 sbpipe.main (module), 16
 sbpipe.pl (module), 8
 sbpipe.pl.create (module), 1

sbpipe.pl.create.newproj (module), 1
 sbpipe.pl.pe (module), 3
 sbpipe.pl.pe.parest (module), 2
 sbpipe.pl.pipeline (module), 7
 sbpipe.pl.ps1 (module), 4
 sbpipe.pl.ps1.parscan1 (module), 3
 sbpipe.pl.ps2 (module), 6
 sbpipe.pl.ps2.parscan2 (module), 5
 sbpipe.pl.sim (module), 7
 sbpipe.pl.sim.sim (module), 6
 sbpipe.report (module), 9
 sbpipe.report.latex_reports (module), 8
 sbpipe.sbpipe_config (module), 17
 sbpipe.simul (module), 13
 sbpipe.simul.copasi (module), 10
 sbpipe.simul.copasi.copasi (module), 10
 sbpipe.simul.pl_simul (module), 10
 sbpipe.simul.python (module), 10
 sbpipe.simul.python.python (module), 10
 sbpipe.simul.simul (module), 11
 sbpipe.utils (module), 16
 sbpipe.utils.io (module), 13
 sbpipe.utils.parcomp (module), 14
 sbpipe.utils.rand (module), 16
 sbpipe.utils.re_utils (module), 16
 sbpipe_logo() (in module sbpipe.main), 17
 set_basic_logger() (in module sbpipe.main), 17
 set_color_logger() (in module sbpipe.main), 17
 set_console_logger() (in module sbpipe.main), 17
 set_logger() (in module sbpipe.main), 17
 Sim (class in sbpipe.pl.sim.sim), 6
 sim() (sbpipe.simul.copasi.copasi.Copasi method), 10
 sim() (sbpipe.simul.pl_simul.PLSimul method), 11
 sim() (sbpipe.simul.simul.Simul method), 12
 Simul (class in sbpipe.simul.simul), 11

W

which() (in module sbpipe.sbpipe_config), 17
 write_mat_on_file() (in module sbpipe.utils.io), 14