EFMIrs Documentation

Release 2.3.5+2.g8bbfdf4.dirty

Bianca Buchner

Jun 25, 2021

Contents

Contents:

CHAPTER 1

Installation

At the command line:

```
$ pip install efmlrs
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv efmlrs
$ pip install efmlrs
```

CHAPTER 2

Usage

To use EFMlrs in a project:

```
import efmlrs
```

CHAPTER 3

efmlrs

3.1 efmlrs package

3.1.1 Subpackages

efmlrs.postprocessing package

Subpackages

efmlrs.postprocessing.decompressions package

Submodules

efmlrs.postprocessing.decompressions.deadend module

efmlrs.postprocessing.decompressions.deadend.**decompressions**(*compressed*, *mapping*)

Decompresses the part of the current compressed efm that has been compressed during deadend compression by inserting zeros at the index of deadend metabolites.

Parameters

- **compressed** (*list*) – list containing efm for decompression
- **mapping** – list containing decompression information for deadend decompressions

Returns compressed: list containing current efm with decompressed deadend part

efmlrs.postprocessing.decompressions.deadend.**parse_info**(*file*)

Reads decompression information from infofile and builds decompression list for deadend compressions.

Parameters **file** – infofile automatically created during compressions, containing all information for decompressions

Returns cmp_info: list containing decompression information for deadend compressions

efmlrs.postprocessing.decompressions.null_space module

efmlrs.postprocessing.decompressions.one2many module

Module contents

Submodules

efmlrs.postprocessing.decompressing module

efmlrs.postprocessing.decompressing.**build_reverse_mapping**(*infofile*, *counter*)
Reads compression information form infofile and builds a mapping for deompressions which is in reversed order of the previously applied compressions.

Parameters

- **infofile** – file automatically created during compressions, containing all information for decompressions
- **counter** (*int*) – number of compression rounds

Returns mappings - list of different tuple with compression information for each compression step

efmlrs.postprocessing.decompressing.**decompressing**(*compressed_efms*, *outputfile*, *mappings*, *bound_info*)
Iteratively decompresses one efm after another by applying decompressions according to decompression information stored in mappings. If model had additional boundaries: removes lambda vector entry and additional boundary reactions from current efm and normalizes it. Writes final decompressed efm to user specified output file.

Parameters

- **compressed_efms** (*list*) – list of lists containing compressed efms
- **outputfile** – user specified filed to write decompressed files in
- **mappings** – list of different tuples with compression information for each compression step
- **bound_info** (*int*) – number of additional bounds

Returns None

efmlrs.postprocessing.decompressing.**find_counter**(*infofile*)
Parses compression info and stores information on how many compression rounds were done during preprocessing and how many additional bounds have been applied.

Parameters **infofile** – file automatically created during compressions, containing all information for decompressions

Returns

- round_counter - int number of compression rounds
- bounds - int number of bounds

efmlrs.postprocessing.decompressing.**normalize_efms**(*decompressed*, *bound_info*)
Only called if model had additional boundaries. Removes lambda vector entry and additional boundary reactions from current efm and normalizes it.

Parameters

- **decompressed** (*list*) – current efm as list
- **bound_info** (*int*) – number of additional bounds

Returns decompressed: current efm as list

efmlrs.postprocessing.decompressing.**run**(*compressed_efms*, *infofile*, *outputfile*)
Entry point for decompressing. Reads compression information form infofile and builds a mapping for deompressions which is in reversed order of the previously applied compressions. Iteratively decompresses one efm after another by applying decompressions according to decompression information stored in mappings. If model had additional boundaries: removes lambda vector entry and additional boundary reactions from current efm and normalizes it. Writes final decompressed efm to user specified output file.

Parameters

- **infofile** – file automatically created during compressions, containing all information for decompressions
- **compressed_efms** (*list*) – compressed efms
- **compression_infos** – info file created during preprocessing
- **outputfile** – path to output file

Returns None

efmlrs.postprocessing.decompressing.**write_decompressed_efms**(*decompressed*, *outputfile*)
Writes final decompressed efm to user specified output file.

Parameters

- **decompressed** – current efm as list
- **outputfile** – user specified filed to write decompressed files in

Returns None

efmlrs.postprocessing.get_data module

efmlrs.postprocessing.get_data.**get_efmtool_efms**(*inputfile*)
Entry point for get_data for efmtool. Reads efmtool output file and stores compressed efms in a list of lists.

Parameters **inputfile** – efmtool output file

Returns list compressed_efms

efmlrs.postprocessing.get_data.**get_mplrs_efms**(*inputfile*, *infofile*)
Entry poitn for get_data for mplrs. Parses mplrs output file, merges splitted reactions, removes rows containing only zeros and stores compressed efms in a list of lists.

Parameters

- **inputfile** – mplrs output file
- **infofile** – info file created during preprocessing

Returns list compressed_efms

efmlrs.postprocessing.get_data.**get_rev_info**(*infofile*)
Reads info file and stores reaction reversibility information in a list.

Parameters **infofile** – info file created during preprocessing

Returns list rev_info

efmlrs.postprocessing.get_data.**parse_lrs**(*inputfile*, *reversibilities*)
Parses mplrs output file, merges splitted reactions, removes rows containing only zeros and stores compressed efms in a list of lists.

Parameters

- **inputfile** – mplrs output file
- **reversibilities** (*list*) – list of reaction reversibilities

Returns list compressed_efms - list of lists with compressed efms

efmlrs.postprocessing.get_data.**remove_zeros**(*row*)
Checks if extracted efm (row) contains only zeros.

Parameters **row** (*list*) – extracted efm

Returns row or None

Module contents

efmlrs.preprocessing package

Subpackages

efmlrs.preprocessing.compressions package

Submodules

efmlrs.preprocessing.compressions.deadend module

efmlrs.preprocessing.compressions.deadend.**check_reactions**(*smatrix*, *i*, *reversibilities*)
Checks if corresponding reactions to a deadend metabolite are irreversible. Returns list of corresponding irreversible reactions to be removed.

Parameters

- **smatrix** – sympy matrix that contains the stoichiometric matrix
- **i** (*int*) – index of deadend metabolite
- **reversibilities** (*list*) – list of reaction reversibilities

Returns list of reaction names to be removed from stoichiometric matrix

efmlrs.preprocessing.compressions.deadend.**check_row**(*row*)
Checks if rows qualifies as deadend metabolite. If row consists of only zeros and either positive or negative entries, it is a deadend metabolite and functions returns true.

Parameters **row** – row of stoichiometric matrix

Returns bool

efmlrs.preprocessing.compressions.deadend.**find_deadends**(*smatrix*, *reversibilities*)

Checks all rows of stoichiometric matrix if they are deadend metabolites. Checks if corresponding reactions are irreversible.

Parameters

- **smatrix** – sympy matrix that contains the stoichiometric matrix
- **reversibilities** (*list*) – list of reaction reversibilities

Returns

- remove_reactions - list of reaction names that will be removed
- remove_metabolites - list of metabolite names that will be removed

efmlrs.preprocessing.compressions.deadend.**run**(*smatrix*, *reactions*, *reversibilities*, *metabolites*, *core_name*, *outer_counter*)

Entry point for deadend compression. Finds and removes deadend metabolites and corresponding reaction that only contain zeros after a deadend metabolite is removed. This is done iteratively as long as deadend metabolites or corresponding reactions are found in the stoichiometric matrix. Writes information on removed reactions and metabolites to compression log.

Parameters

- **smatrix** – sympy matrix that contains the stoichiometric matrix
- **reactions** (*list*) – list of reaction names
- **reversibilities** (*list*) – list of reaction reversibilities
- **metabolites** (*list*) – list of metabolite names
- **core_name** (*str*) – string that consists of path to and name of the input file excluding file extension
- **outer_counter** (*int*) – int that counts how many iterative steps with all compressions have been performed

Returns

- smatrix - sympy matrix reduced stoichiometric matrix
- reactions - list of reduced reactions names
- reversibilities - list of reduced reaction reversibilities
- metabolites - list of reduced metabolite names

efmlrs.preprocessing.compressions.deadend.**write_deadend_info**(*core_name*, *outer_counter*, *removedReactions*)

Writes compression information to *.info file for decompression during post-processing.

Parameters

- **core_name** (*str*) – string that consists of path to and name of the input file excluding file extension
- **outer_counter** (*int*) – int that counts how many iterative steps with all compressions have been performed
- **removedReactions** (*list*) – list of removed reaction indices

Returns None

efmlrs.preprocessing.compressions.echelon module

efmlrs.preprocessing.compressions.echelon.**find_redundant_metabolites**(*smatrix*, *inner_counter*)

Calculates the reduced row echelon form of the stoichiometric matrix. Finds metabolites that are redundant due to conservation relations.

Parameters

- **smatrix** – sympy matrix that contains the stoichiometric matrix
- **inner_counter** (*int*) – integer that counts iterative steps of nullspace compression

Returns list redundant_metas: list of metabolites to be removed

efmlrs.preprocessing.compressions.echelon.**remove_redundant_metabolites**(*smatrix*, *metabolites*, *redundant_metas*)

Removes redundant metabolites from stoichiometric matrix and from list of metabolite names. Writes compression information to log file. (https://academic.oup.com/bioinformatics/article/24/19/2229/246674)

Parameters

- **smatrix** – sympy matrix that contains the stoichiometric matrix
- **metabolites** (*list*) – list of metabolite names
- **redundant_metas** (*list*) – list of metabolites to be removed

Returns

- smatrix - sympy matrix reduced stoichiometric matrix
- metabolites - list of reduced metabolite names

efmlrs.preprocessing.compressions.echelon.**run**(*smatrix*, *metabolites*)

Entry point for echelon compression. Iteratively finds inconsistencies due to conservation relations of metabolites in the reduced row echelon form of the stoichiometric matrix and removes them.

Parameters

- **smatrix** – sympy matrix that contains the stoichiometric matrix
- **metabolites** (*list*) – list of metabolite names

Returns

- smatrix - sympy matrix reduced stoichiometric matrix
- metabolites - list of reduced metabolite names

efmlrs.preprocessing.compressions.null_space module

efmlrs.preprocessing.compressions.one2many module

Module contents

Submodules

efmlrs.preprocessing.boundaries module

efmlrs.preprocessing.boundaries.**add_boundary_names**(*bounds_info*)
Extends reaction name for reactions with additional bounds by adding "min" or "max" to lower respectively upper bounds.

Parameters **bounds_info** – dictionary containing information on reaction and associated bounds

Returns dictionary containing information on reaction bounds with extended reaction name

efmlrs.preprocessing.boundaries.**build_new_metas**(*bounds*, *meta_ori*)
Builds new list of metabolite names including slack metabolites.

Parameters

- **bounds** – dictionary with bound information
- **meta_ori** – list of metabolite names

Returns new_metas list of metabolite names including added slack metabolites

efmlrs.preprocessing.boundaries.**build_new_reas**(*bounds*, *reas_ori*)
Builds new list of reaction names including slack metabolites.

Parameters

- **bounds** – dictionary with bound information
- **reas_ori** – list of reactions names

Returns new_reas list reaction names including added slack reactions

efmlrs.preprocessing.boundaries.**build_new_reversibilities**(*bounds*, *revs_ori*)
Builds new list of reaction reversibilities. All added reaction are irreversible.

Parameters

- **bounds** – dictionary with bound information
- **revs_ori** – list of reaction reversibilities

Returns new_revs list of reaction reversibilities including added slack reactions

efmlrs.preprocessing.boundaries.**build_new_smatrix**(*smatrix*, *bounds*, *reactions*)
Extends stoichiometric matrix according to additional bounds with slack reactions and slack metabolites. Builds and adds slack reactions and metabolites to stoichiometric matrix. Converts matrix coefficients to fractions.

Parameters

- **smatrix** – stoichiometric matrix
- **bounds** – dictionary with bound information
- **reactions** – list of reaction names

Returns extended stoichiometric matrix

efmlrs.preprocessing.boundaries.**build_slack_metas_01**(*bounds*, *reactions*)
Builds first part of slack metabolites. Length equals length of original stoichiometric matrix. Adds 1 or -1 as coefficient at index of slack metabolite for upper respectively lower bound.

Parameters

- **bounds** – dictionary with bound information

- **reactions** – list of reaction names

Returns list of lists with new build slack reactions

efmlrs.preprocessing.boundaries.**build_slack_metas_02**(*bounds*)
Builds entries for entries for slack reactions length equals number of additional bounds.

Parameters **bounds** – dictionary with bound information

Returns list slack_reas_2 of lists with entries for slack reactions

efmlrs.preprocessing.boundaries.**build_slack_metas_03**(*bounds*)
Builds last part of slack metabolite (= lambda column) with actual reaction bound as coefficient.

Parameters **bounds** – dictionary with bound information

Returns list slack_metas_3 of list with actual reaction bound as entry

efmlrs.preprocessing.boundaries.**building_slack_metabolites**(*bounds*, *reactions*)
Builds slack metabolites. Consists of three parts. First part builds list with length of original matrix and coefficients according to lower or upper bound. Second part build entries for slack reactions length equals number of additional bounds. Third part builds entry for lambda reaction, length always 1, entry is the actual reaction bound.

Parameters

- **bounds** – dictionary with bound information
- **reactions** – list of reaction names

Returns list slack_metas of lists with new build slack metabolites

efmlrs.preprocessing.boundaries.**extend_smatrix_1**(*matrix*, *bounds*)
Extends columns of stoichiometric matrix by number of bounds plus 1 with arrays containing only zeros.

Parameters

- **matrix** – stoichiometric matrix
- **bounds** – dictionary with bound information

Returns extended stoichiometric matrix

efmlrs.preprocessing.boundaries.**extend_smatrix_2**(*slack_metas*, *smatrix_extended*)
Extends stoichiometric matrix by new build slack metabolites.

Parameters

- **slack_metas** (*list*) – list of lists with new build slack metabolites
- **smatrix_extended_1** – stoichiometric matrix that has been extended by columns containing only zero

Returns new build smatrix with slack reactions and slack metabolites

efmlrs.preprocessing.boundaries.**format_bounds**(*bounds_info*)
Changes bound information for additional reaction bounds stored in bounds_info from str (cobrapy format) to int.

Parameters **bounds_info** – dictionary containing information on reaction and associated bounds

Returns dictionary containing information on reaction bounds with extended reaction name and int as bounds

efmlrs.preprocessing.boundaries.**get_bounds_index**(*reas*, *bounds*)
Gets index for addtional bounds according to list of reaction names.

Parameters

- **reas** (*list*) – list of reaction names
- **bounds** – dictionary with bound information

Returns list of indicis of reaction bounds

efmlrs.preprocessing.boundaries.**run**(*model*, *smatrix*, *reactions*, *reversibilities*, *metabolites*)

Entry point for boundaries. Searched for additional reaction bounds. Adds slack reaction and metabolites to the stoichiometric matrix, the lists of reaction names and metabolie names and the list of reaction reversibilities.

Parameters

- **model** – cobrapy model
- **smatrix** – stoichiometric matrix
- **reactions** – list of reaction names
- **reversibilities** – list of reaction reversibilities
- **metabolites** – list of metabolite names

Returns

- smatrix - sympy matrix reduced stoichiometric matrix
- reactions - list of reactions names
- reversibilities - list of reaction reversibilities
- metabolites - list of metabolite names
- bound_counter - integer number of added bounds

efmlrs.preprocessing.boundaries.**search_bounds**(*model*)

Searches cobrapy model for additional reaction bounds.

Parameters **model** – cobrapy model

Returns dictionary containing information on reactions with additional bounds (reaction id, lower and upper bound) or returns None if no additional reaction bounds were found in the model.

efmlrs.preprocessing.get_data module

efmlrs.preprocessing.get_data.**check_bounds**(*model*, *smatrix*, *reactions*, *reversibilities*, *metabolites*)

Calls boundaries script.

Parameters

- **model** – cobrapy model
- **smatrix** – stoichiometric matrix
- **reactions** – list of reaction names
- **reversibilities** – list of reaction reversibilities
- **metabolites** – list of metabolite names

Returns

- smatrix - matrix stoichiometric matrix
- reactions - list of reactions names

- reversibilities - list of reaction reversibilities
- metabolites - list of metabolite names
- bound_counter - integer number of added bounds

efmlrs.preprocessing.get_data.**compartments_2_rm**(*model*, *comp_list: list*)
Checks if compartment were specified to ignore and removes metabolites that belong to specified compartments.

Parameters

- **model** – cobrapy model
- **comp_list** (*list*) – list of compartments that will be removed

Returns model: altered cobrapy model

efmlrs.preprocessing.get_data.**get_smatrix**(*model*)
Gets stoichiometric matrix from cobrapy model using the cobrapy function "cobra.util.array.create_stoichiometric_matrix".

Parameters **model** – cobrapy model

Returns matrix: stoichiometric matrix

efmlrs.preprocessing.get_data.**orphaned_metas_rm**(*model*)
Searches and removes orphaned metabolites from cobrapy model. Orphaned metabolites are not involved in any reaction.

Parameters **model** – cobrapy model

Returns model: altered cobrapy model

efmlrs.preprocessing.get_data.**read_model**(*input_filename*)
Reads metabolic model from sbml file using the 'cobra.io.read_sbml_model' functions. Reads io string during reading model and catches exchange reactions added by cobrapy. These reactions are going to be removed from the model again. :param input_filename: sbml file with cobrapy compatible metabolic model :return:

- model - cobrapy model
- reas_added - list of reactions added by cobrapy

efmlrs.preprocessing.get_data.**rm_empty_reas**(*model*)

efmlrs.preprocessing.get_data.**rm_metas_in_specified_compartment**(*comp*, *model*)
Removes metabolites in specified compartment.

Parameters

- **comp** (*str*) – comparment to ignore
- **model** – cobrapy model

Returns model: altered cobrapy model

efmlrs.preprocessing.get_data.**rm_reactions**(*model*, *rea_list*)
Removes exchange reaction that were added by cobrapy.

Parameters

- **model** – cobrapy model
- **rea_list** (*list*) – list of reaction names that will be removed

Returns

- model - altered cobrapy model

efmlrs.preprocessing.get_data.**run**(*inputfile*, *ignore_compartments*, *boundflag*)
Entry point for get_data. Takes sbml file as input. Removes exchange reactions that were added by cobrapy during reading. Using cobrapy the model name and properties are extracted. Removes orphaned metabolites. As specified by user input ignores compartments a and creates boundary reactions and metabolites. Converts stoichiometric matrix coefficients into fractions for precise arithmetic in later calculations and converts stoichiometric matrix from list of lists to sympy matrix. Creates the following files: sfile (smatrix), mfile (metabolite names), rfile (reaction names), rvfile (reaction reversibilities) and info file.

Parameters

- **inputfile** – sbml file with cobrapy compatible metabolic model
- **ignore_compartments** (*str*) – (optional) user input as string with compartment name that will be ignored
- **boundflag** (*bool*) – (optional) user input as bool flag if boundaries from sbml file will be taken into account

Returns

- smatrix - sympy matrix of stoichiometric matrix
- reactions - list of reactions names
- reversibilities - list of reaction reversibilities
- metabolites - list of metabolite names
- model - cobrapy model
- core_name - path to input file without extensions

efmlrs.preprocessing.get_data.**write_bound_info**(*core_name*, *bound_counter*)
Writes boundary information to info file.

Parameters

- **core_name** – string that consists of path to and name of the input file excluding file extension
- **bound_counter** (*int*) – integer number of added bounds

Returns None

efmlrs.preprocessing.mplrs_output module

efmlrs.preprocessing.mplrs_output.**run_cmp**(*core_name*)
Entry point for mplrs_output. Creates input file from sfile and rvfile suitable for mplrs algorithm.

Parameters **core_name** – path to input file without extensions

Returns None

efmlrs.preprocessing.mplrs_output.**run_uncmp**(*core_name*)
Entry point for mplrs_output. Creates input file from sfile and rvfile suitable for mplrs algorithm.

Parameters **core_name** – path to input file without extensions

Returns None

efmlrs.preprocessing.mplrs_output.**split_reversible_reas**(*smatrix*, *reversibilities*)
Splits all reversible reactions into two single reactions. One for forward and one for backward.

Parameters

- **smatrix** – list of lists of stoichiometric matrix
- **reversibilities** – list of reaction reversibilities

Returns list of lists of reconfigured stoichiometric matrix

efmlrs.preprocessing.mplrs_output.**write_header**(*mplrs_file*, *core_name*)
Writes header for mplrs input file.

Parameters

- **mplrs_file** – path to mplrs input file
- **core_name** – path to input file without extensions

Returns None

efmlrs.preprocessing.mplrs_output.**write_lrs**(*core_name*, *reconf_smatrix*)
Write input file for mplrs algorithm.

Parameters

- **core_name** – path to input file without extensions
- **reconf_smatrix** – list of lists of reconfigured stoichiometric matrix

Returns None

efmlrs.preprocessing.mplrs_output.**write_lrs_cmp**(*core_name*, *reconf_smatrix*)
Write input file for mplrs algorithm.

Parameters

- **core_name** – path to input file without extensions
- **reconf_smatrix** – list of lists of reconfigured stoichiometric matrix

Returns None

efmlrs.preprocessing.mplrs_output.**write_smatrix**(*mplrs_file*, *reconf_smatrix*)
Writes reconfigured stoichiometric matrix, unity matrix and ending to mplrs input file.

Parameters

- **mplrs_file** – path to mplrs input file
- **reconf_smatrix** – list of lists of reconfigured stoichiometric matrix

Returns None

Module contents

efmlrs.util package

Submodules

efmlrs.util.convert_matrix module

efmlrs.util.convert_matrix.**get_denominators**(*smatrix*)
Gets greatest denominator of each row in stoichiometric matrix :param smatrix: list of lists of stoichiometric matrix :return: list of greatest denominators for each row

efmlrs.util.convert_matrix.**get_int_smatrix**(*smatrix*, *greatest_denominators*)
Converts stoichiometric matrix from fractions to integer by multiplying each row with greatest denominator. :param smatrix: list of lists of stoichiometric matrix :param greatest_denominators: list of greatest denominators for each row :return: list of lists of converted stoichiometric matrix

efmlrs.util.convert_matrix.**get_min_smatrix**(*int_smatrix*)
Gets greatest divisor for each row and divides each row through greatest divisor. Converts matrix from list of lists to sympy matrix. :param int_smatrix: list of lists of stoichiometric matrix with integers as coefficients :return: sympy matrix of stoichiometric matrix with smallest coefficients

efmlrs.util.convert_matrix.**run**(*smatrix*)
Converts stoichiometric matrix from fractions to integer by multiplying each row with greatest denominator. For each row in stoichiometric matrix reduces coefficients to smallest number by dividing each row through greatest divisor. Converts matrix from list of lists to sympy matrix. :param smatrix: list of lists of stoichiometric matrix :return: sympy matrix of stoichiometric matrix with smallest coefficients

efmlrs.util.data module

Collection of small helper scripts.

efmlrs.util.data.**bounds_print**(*bounds*)

efmlrs.util.data.**convert_df2matrix**(*smatrix_df*)

efmlrs.util.data.**convert_float2fraction**(*matrix*)

efmlrs.util.data.**convert_fraction2float**(*matrix*)

efmlrs.util.data.**convert_matrix2df**(*smatrix*)

efmlrs.util.data.**efmlrs_finish_compressions**()

efmlrs.util.data.**efmlrs_finish_decompressions**()

efmlrs.util.data.**efmlrs_start_compressions**()

efmlrs.util.data.**efmlrs_start_decompressions**()

efmlrs.util.data.**read_mfile**(*name*)

efmlrs.util.data.**read_rfile**(*name*)

efmlrs.util.data.**read_rvfile**(*name*)

efmlrs.util.data.**read_sfile**(*name*)

efmlrs.util.data.**read_sfile_int**(*name*)

efmlrs.util.data.**reversibilities4printing**(*reversibilities*)

efmlrs.util.data.**write_all**(*smatrix*, *reactions*, *reversibilities*, *metabolites*, *name*)

efmlrs.util.data.**write_cmp_int_matrix**(*core_name*)

efmlrs.util.data.**write_efms**(*name*, *efms*)

efmlrs.util.data.**write_info**(*core_name*, *reversibilities*, *outer_counter*)

efmlrs.util.data.**write_init_mfile**(*name*, *metabolism*)

efmlrs.util.data.**write_init_rfile**(*name*, *reactions*)

efmlrs.util.data.**write_initial_files_no_bounds**(*core_name*, *reactions*, *reversibilities*, *metabolites*)

efmlrs.util.data.**write_initial_files_with_bounds**(*core_name*, *reactions*, *reversibilities*, *metabolites*)

efmlrs.util.data.**write_mfile**(*name*, *metabolism*)

efmlrs.util.data.**write_rfile**(*name*, *reactions*)

efmlrs.util.data.**write_rvfile**(*name*, *reversibles*)

efmlrs.util.data.**write_sfile**(*name*, *smatrix*)

efmlrs.util.data.**write_sfile_float**(*name*, *smatrix*)

efmlrs.util.data.**write_uncmp_int_matrix**(*core_name*)

efmlrs.util.log module

Collection of small scripts for creating log file.

efmlrs.util.log.**log**()

efmlrs.util.log.**log_add**()

efmlrs.util.log.**log_close**()

efmlrs.util.log.**log_del**()

efmlrs.util.log.**log_delete**(*rea1*, *rea2*)

efmlrs.util.log.**log_delete_meta**(*meta*)

efmlrs.util.log.**log_delete_rea**(*rea*)

efmlrs.util.log.**log_init**(*name*)

efmlrs.util.log.**log_keep**()

efmlrs.util.log.**log_merge**(*remove*, *keep*, *keep_factor*)

efmlrs.util.log.**log_merge_many**(*remove*, *keep*, *remove_factor*, *factors*, *force=False*)

efmlrs.util.log.**log_merge_rea**(*rea*)

efmlrs.util.log.**log_module**()

efmlrs.util.log.**log_strip**(*rea*)

efmlrs.util.log.**log_time**()

efmlrs.util.log.**merge_reaction_names**(*R1_keep*, *R2_remove*)

Module contents

3.1.2 Submodules

3.1.3 efmlrs.post module

efmlrs.post.**main**(*inputfile*, *outputfile*, *infofile*, *efmtool*)

Main script of prostprocessing part of EFMlrs. Calls all other scripts. First output files are read and compressed efms are stored in a list. Then the decompressing process is started iteratively in reversed order compared to the formerly applied compressions and each compressed efm is being decompressed one after another. The deompressed efms are written to the user specified output file. :param inputfile: output containing compressed efms either form mplrs or efmtool :param outputfile: file in which uncompressed efms are written to :param

infofile: file automatically created during compressions, containing all information for decompressions :param efmtool: paramter that indicates whether decompressions for mplrs or efmtool is being executed :return: None

efmlrs.post.**start**(*inputfile*, *outputfile*, *infofile*, *efmtool*)
Takes all parameters form commandline, checks if parameters are okay and calls main function if an error occurs an exception is raised and ends the program. :param inputfile: output containing compressed efms either form mplrs or efmtool :param outputfile: file in which uncompressed efms are written to :param infofile: file automatically created during compressions, containing all information for decompressions :param efmtool: paramter that indicates whether decompressions for mplrs or efmtool is being executed :return: None

efmlrs.post.**start_from_command_line**()
Entry point for prostprocessing EFMlrs. Contains all information and arguments for command line call. Calls start function. :return: None

3.1.4 efmlrs.pre module

efmlrs.pre.**main**(*inputsbml*, *ignore_compartments*, *boundflag*)
Main script of preprocessing part of EFMlrs. Calls all other scripts. Compressions are being done iteratively as long as compressions can be found. Loop breaks after one loop with no changes in length of metabolites or reactions. Writes uncompressed and compressed output files including additional versions of the sfile with integers (instead of fractions) to be compatible with efmtool. Writes compressed input file for mplrs algorithm. Writes info file that is needed for decompressions and log file that contains all information on the applied compressions.

Parameters

- **inputsbml** – path to sbml input file that contains the metabolic model
- **ignore_compartments** – list of compartments to ignore
- **boundflag** – bool flag when additional bounds should be taken into account

Returns None

efmlrs.pre.**start**(*inputsbml*, *ignore_compartments*, *bounds*)
Takes all parameters from command line, checks if parameters are okay and if everything is fine initialises log file and calls main function if an error occurs an exception is raised and ends the program. :param inputsbml: path to sbml input file that contains the metabolic model :param ignore_compartments: list of compartments to ignore :param bounds: bool flag when additional bounds should be taken into account :return: None

efmlrs.pre.**start_from_command_line**()
Entry point for preprocessing EFMlrs. Contains all information and arguments for command line call. Calls start function. :return: None

3.1.5 Module contents

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/BeeAnka/EFMlrs/issues.

If you are reporting a bug, please include:

- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

EFMlrs could always use more documentation, whether as part of the official EFMlrs docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/BeeAnka/EFMlrs/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *EFMlrs* for local development.

1. Fork the *EFMlrs* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/EFMlrs.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv EFMlrs
$ cd EFMlrs/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 efmlrs tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.

2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.3, 3.4, 3.5 and for PyPy. Check https://travis-ci.org/BeeAnka/EFMlrs/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 5

Credits

5.1 Maintainer

- Bianca Allegra Buchner <bianca.buchner@boku.ac.at>

5.2 Contributors

None yet. Why not be the first? See: CONTRIBUTING.rst

CHAPTER 6

History

We are currently in a pre-release state. Beta release it planed for begin of april 2020.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

F

G

L

M

N

O

P

R

S

W