influx_si Documentation

Release 6.0

Serguei SOKOL

Jun 08, 2022

CONTENTS

1 Introduction 1

2 Change Log for influx_si 5

3 Installation 21

4 Quick Start 25

5 User's manual 27

6 FTBL format evolution 61

7 Programmer's documentation for influx_si 65

8 How to . . . 79

9 Troubleshooting 81

10 Consulting and more 83

11 License for influx_si software 85

12 Indices and tables 93

Python Module Index 95

Index 97

CHAPTER

ONE

INTRODUCTION

`influx_s` and `influx_i` are programs designed for estimation of flux and chemical specie concentrations based on labeling data using stable isotopes (essentially ^{13}C but combination of multiple isotopes like ^{2}H, ^{13}C, ^{15}N, ... is also possible). `influx_s` works with stationary data while the `influx_i` is able to simulate instationary labeling (hence the `_i` in the name). Both work in metabolically stationary context. The whole project is referred as `influx_si`. Note also that the term `influx_si` is used in contexts where `influx_s` and `influx_i` are interchangeable.

influx_si

Flux and metabolite concentration values are obtained as a result of a fitting between simulated labeling data and the data measured by MS or NMR techniques. In this documentation, the terms *fitting* and *optimization* are used as synonyms.

influx_s

For the theory behind flux calculations in stationary labeling context, see the following papers:

Wiechert, W., Möllney, M., Isermann, N., Wurzel, M., and de Graaf, A. A. (1999). Bidirectional reaction steps in metabolic networks: III. Explicit solution and analysis of isotopomer labeling systems. *Biotechnol Bioeng,* 66(2), 69-85.

Antoniewicz, M. R., Kelleher, J. K., and Stephanopoulos, G. (2007). Elementary metabolite units (EMU): a novel framework for modeling isotopic distributions. *Metab Eng*, 9(1), 68-86.

Sokol, S., Millard, P., and Portais, J-C. (2012). influx_s: increasing numerical stability and precision for metabolic flux analysis in isotope labeling experiment. *Bioinformatics*, 2012, 28, 687-693

The main additional value to flux calculation of `influx_si` compared to other publicly available software (13CFlux, OpenFlux, INCA, ...) is the usage of NLSIC algorithm for fitting purposes. This algorithm provides:

- more reliable convergence which results in better numerical precision, i.e. even started from random initial points, it converges to the same solution if no local minima are present. So the spread of final solutions is close to zero.
- better accuracy, i.e., the found numerical solution lies closer to the theoretical solution than solutions provided by concurrent minimization algorithms. Thus, `influx_s` provides better numerical accuracy.

For more details, see the paper on `influx_s` cited above.

Moreover, `influx_s` provides:

- both cumomer and EMU frameworks for describing label distribution in the metabolites;
- parallel experiment treatment both in stationary and instationary modes;

- estimation of specie concentration in particular in stationary contexts (since v2.0. A methodology behind metabolite concentration evaluation is not yet published at the moment of this writing.);
- a possibility to deal with metabolite pool confusion appearing either in compartmentation or in coelution;
- taking into account non-carbon carrying fluxes like the balances of ADP/ATP, H2O, energy, electrons and so on;
- an optional automatic choice of free fluxes;
- optional equality and inequality constraint on fluxes and metabolite concentrations;
- short time execution and design for many core computers. So it facilitates high throughput flux calculations in parallel way;
- a 'least norm' option that, in presence of structurally non identifiable fluxes, still allows to estimate some of fluxes (those remained identifiable);
- a khi2 statistical test 'goodness of fit'
- an optional automatic elimination of outliers;
- a command line interface letting an easy integration in automatic processing chains as well as many others features and options;
- a possible scripting of post-treatment or graphic generating tasks;
- multi-platform support. It runs everywhere R and Python run, i.e. on Linux, Windows, MacOS and other Unix variants.

`influx_i`

Instationary labeling (hence the final 'i' in the name) is the domain of `influx_i`. The theory of instationary labeling was developed, for example in

Katharina Nöh, Wolfgang Wiechert (2006) Experimental Design Principles for Isotopically Instationary 13C Labeling Experiments *Biotechnology and Bioengineering*, 94(2), 234-251

Sokol S, Portais J-C (2015) Theoretical Basis for Dynamic Label Propagation in Stationary Metabolic Networks under Step and Periodic Inputs. *PLoS ONE* 10(12): e0144652. doi:10.1371/journal.pone.0144652

As `influx_i` capitalizes on `influx_s` development and shares a big part of code, `influx_i` presents the same advantages as listed in the previous section. It uses the same FTBL file format for network and measurements definitions and includes all options available for `influx_s`. Instationary labeling data can be supplied by an additional tab formatted ASCII file making a shift from stationary to instationary calculations as simple as possible. Some of advantages of `influx_i` over the concurrent software coping with instationary labeling data are:

- fast calculations (e.g. on our Intel Xeon 2.50GHz workstation, `e_coli_i` case runs in 17s while the most important part devoted to optimization takes as low as 10s);
- parallel experiment treatment;
- available choice between first and second order time schemes for ODE (ordinary differential equations) resolution;
- unconditional stability during ODE solving.

Documentation organization

Changes brought to every new version and bug fixes are resumed at the beginning of the next chapter *Change Log* which is also distributed as a stand alone PDF file.

The rest of the documentation is organized as follows. *Installation* chapter provides brief instructions for software installation. *Quick start* chapter gives an opportunity to a user to quickly start and evaluate the software and to see if it corresponds to what he is looking for. A more detailed but still short *User's manual* precedes a *Programmer's documentation*. The latter chapter can be safely skipped by a user not interested in developing new features or fixing some problems in `influx_si`. A small collection of *How to...* and *Troubleshooting* notice conclude the documentation.

Licensing

The original version of `influx_si` software was developed in the MetaSys team in the LISBP, Toulouse, FRANCE.

A file `influx_si/R/psoptim_ic.R` is based on the code from CRAN package pso v1.0.3 published in 2012 by Claus Bendtsen (papyrus.bendtsen at gmail.com). The original code is licensed under LGPL-3 terms so our modifications are licensed under the same terms .

If you publish results obtained with `influx_s` you have to cite the original paper in Bioinformatics 2012 (cf. above). A paper describing `influx_i` is yet to publish.

Software and documentation author:

Serguei SOKOL, INRAE, France <sokol [at] insa-toulouse.fr>

CHAPTER

TWO

CHANGE LOG FOR INFLUX_SI

2022-05-16 version 6.0

New features:

- introduced new front-end MTF format (cf. ftbl2mtf, txt2ftbl)
- added plot_ilab.R for plotting instationary MS/NMR measurements and simulations (replaces plot_imass.R)
- ftbl2xgmml: added orange color for dead-end metabolites
- txt2ftbl: added detection of disconnected sub-networks
- ftbl2netan: dead-ends are reported
- ftbl2metxml: added pathway information and compartment treatment

Bug fix:

- fixed MS measurements on long molecules in –emu mode

2021-02-26 version 5.4.0

New features:

- influx_i: user can add custom R expressions for label input varying in time (cf. funlabR)
- influx_i: explicit error message on non valid entries in label kinetic data (requested by Pierre Millard, INRAE, France)
- added optctrl:mumps:icntl_NN option (case reported by Albert Fina Romero, UAB, Spain)
- influx_i: comment string in kinetic data has changed to `//` (`#` will still work with old files)
- added `-i` option to ftbl2netan (for influx_i-like analysis)

Bug fix:

- fixed error raising on redundant equality when `--ffguess` is used (reported by Pierre Millard, INRAE, France)
- fixed fatal error with scale factor infrastructure in some cases when `--ffguess` is used
- `plot_imass.R` is fixed for a case when there is no any MS measurement
- explicit error message on fail while reading label kinetic data
- fixed not working error messages in few utilities

- fixed too early closed kvh file connection
- fixed post-production calculations when optimization in influx_si failed

2020-07-24 version 5.3.0

New features:

- added PSO optimization method (suggested by Pierre Millard, INRAE, Toulouse, France)
- now optimization methods can be chained in the same run, e.g. `--meth pso --meth nlsic`
- the field `optctrl...` is now formatted to allow homonym parameters with different values for different optimization methods
- option `--install_rdep` becomes interactive and reproduces the behavior of R's `install.packages()` to allow installation in a user's R personal library when the default directory is not writable (suggested by Matthieu Guionnet, CNRS, Toulouse, France)

Bug fix:

- `plot_smeas.R` is fixed for cases where NA is present in MS measurements.

2020-05-28 version 5.2.0

New features:

- `ff2ftbl` can accept an input kvh file not ending with `_res.kvh`
- `ftbl2netan` will write a partial output in case of error during FTBL parsing
- added error message if input metabolite appears in measurment FTBL sections
- in kvh module added optional boolean parameter 'strip'

Bug fix:

- fixed txt2ftbl, removed xrange() hidden call
- fixed kvh.dict2kvh() for writing objects derived from dict()
- fixed putting border line starting values inside feasible domain for BFGS method
- fixed endline problem in ftbl2metxml and res2ftbl_meas on Windows (reported by Loic Le Gregam, INSA, Toulouse, France)

2020-04-07 version 5.1.0

New features:

- option –cupn is now considered in absolute sens, i.e. net fluxes are limited from both sides: '-cupn <= netflux <= cupn' (before it was 'netflux <= cupn' only)
- influx_i.py is no more symbolic link neither just a copy of influx_s.py, just a python proxy based on exec().
- e_coli.ftbl file from test collection is configured to create e_coli.pdf with graphics instead of e_coli.RData

Bug fix:

- fixed executable files of additional tools
- fixed plot_smeas.R for plotting simulated but not measured MS data
- fixed plot_smeas.R for plotting repeatedly measured fragments by MS
- fixed non reproducible R code generation (Python must be >= 3.6 for this)
- fixed kvh reading which was impacting several additional tools
- parallel experiment examples are back in the test/prl_exp/ directory

2020-03-05 version 5.0.3

New feature:

- all Python scripts are doubled with executable files without '.py' extension (suggested by Pierre Millard, INRAE, and Matthieu Guionnet, CNRS, Toulouse, France)

Bug fix:

- fixed few typos in documentation

2020-02-26 version 5.0.2

Bug fix:

- fixed parallel execution of multiple FTBL on windows
- fixed influx_si hanging on in-existing directory in FTBL name
- fixed dimnames()[[1]]<- failure when nrow==1 (reported by Younes Dellero, INRAE, Rennes, France)
- fixed –ffguess silently failing when redundant reaction was present (idem)
- fixed use of *influx_s.py –install_rdep* (reported by Loic Le Gregam, INSA, Toulouse, France)
- updated user manual for trouble-shooting and consulting services

2020-01-10 version 5.0.1

Bug fix:

- fixed fatal error on windows platform
- fixed startup message about Rcpp_Rmumps module on some platforms
- fixed dependency on libsbml

2019-10-25 version 5.0

New features:

- converted to Python 3

- packaged with python distutils and Conda package management system. Now installation can be made as simple as `pip install influx_si` or `conda install influx_si`. Internet connection is required for both methods.
- auxiliary C++ routines are placed in multbxxc R packages. So, no more compilation is needed at first execution of `influx_si`.

Bug fix (all bugs in this release are reported by Baudoin Delépine, INSA, Toulouse, France):

- fixed doc about R version (now 3.4.0 or higher) and rmumps (now 5.2.1-3 or higher)
- fixed use of *–sln* option
- added explicit error message if `tmax` is Inf (can happen if time grid could not be read from file indicated in `file_labcin` ftbl field)

2017-07-04 version 4.4.3

New features:

- added 95% quantile in monte-carlo/cost/ci field in _res.kvh file. It makes possible to do a mono-tail chi2 test for goodness of fit
- added possibility for FTBL files to be encoded in UTF16 and UTF32 (based on case reported by Lucille Stuani, INSERM, Toulouse, France)
- added an explicit error message if no label information could be found in parallel experiment FTBL (reported idem)

Bug fix:

- fixed a warning in Monte-Carlo iteration about multiple values in if() close (reported idem)

2017-06-15 version 4.4.2

New features:

- added a new field “constrained net-xch01 fluxes” to the result kvh file
- ff2ftbl.py: instead of only free fluxes, all fluxes are read in kvh file. Thus in a modified FTBL, a partition on free/dependent/constrained fluxes can differ from those used in the kvh file.
- ff2ftbl.py: if kvh and ftbl files have the same prefix, only this prefix can be given as a unique command line argument

Bug fix:

- ff2ftbl.py: fixed “end of line” bug on Windows platform
- plot_smeas.R: fixed metabolite names retrieving in parallel experiments
- plot_smeas.R: fixed disgraceful exit if simulation in influx_s failed

2017-05-24 version 4.4.1

New features:

- in plot_smeas.R and plot_imass.R few cosmetic improvements in plot titles and legends

Bug fix:

- fixed non varying free pools in influx_s
- fixed cases where some cumomer (or EMU) weights can have no cumomers (EMU)
- fixed libs.R by including some files (impacted preamble.R)

2017-05-22 version 4.4

New features:

- added plot_smeas.R file to be included in posttreat_R field in FTBL/OPTIONS. It plots all stationary measurements vs their simulated counterparts in a pdf file.
- added preamble.R, an example of starting session when working with mynetwork.RData issued from save_all.R or save_minenv.R
- R can be again of version 3.3+ (not necessarily 3.4+)
- minor speedup in instationary simulations

Bug fix:

- fixed names in dev vector of pool measurements

2017-04-28 version 4.3

New features:

- speed up of about 30-40% is achieved for instationary simulations with 2nd order time scheme (need upgrade R at least to 3.4.0).
- in plot_imass.R, each measured mass fragment is presented in a separate plot instead of regrouping all fragments for a given metabolite on the same plot.
- in plot_imass.R, non measured metabolites are plotted too now

Bug fix:

- fixed Monte-Carlo iterations with –np=1
- added a mention of python-libsbml in installation procedure

2017-03-30 version 4.2

New features:

- added a script ftbl2metxml.py converting an ftbl to an xml file suitable for visualization on http://metexplore.toulouse.inra.fr . Additionally, it reads flux values from corresponding . . . _res.kvh file (if available) and put them in files . . . _net.txt, . . . _fwd.txt and . . . _rev.txt for later copy/pasting on the MetExplore site (suggested by Tony Palama, INSA, Toulouse, France)
- added comment tags '###' to txt network format (recognized by txt2ftbl.py and respectively '//##' tag in FTBL format recognized by all programs using FTBL format) to mark a new pathway. It allows ftbl2metxml.py to assign reactions to pathways and thus make a network graph more readable.

Bug fix:

- a duly error message is added to signal a network without any label entry in a reduced cumomer network.

2017-03-03 version 4.1

New features:

- added explicit error message when label transitions are missing for any reaction in NETWORK section
- improved speed of labeling simulation in influx_i
- added parameter –tblimit[=0] for trace back limit in errors generated by python (for developers only).

Bug fix:

- fixed error appearing in influx_i during parallel experiments in situation where time intervals are different in different experiments (reported by Maria Fatarova, INSA, Toulouse, France)
- fixed file creation in plot_imass.R (it created pdf in the current directory instead of the working one)

2016-12-20 version 4.0.1

Bug fix:

- file txt2ftbl.py was lacking in the previous version (reported by Tony Palama, INSA, Toulouse, France)

2016-12-19 version 4.0

New features:

- parallel experiments (i.e. same metabolic state but different label entries) can now be processed both in stationary (influx_s) and instationary (influx_i) labeling
- reaction having more than 2 metabolites on ever side can now be entered in FTBL as a series of reactions with the same name
- metabolites with no carbon transitions (like ATP, NADP etc. when they are just co-factors) can now be entered in NETWORK section. They can have stoechiometric coefficients different from 1
- the same metabolite can now appear on both sides of a reaction. It can be helpful for some special carbon shuffles
- reactions without carbon transitions can now be entered in a new FTBL section NOTRACER_NETWORK. It is a good place to enter for example biomass reaction. Stoechiometric coefficients different from one are allowed at this place
- new utility txt2ftbl.py translates an easier readable/writable format for chemical reactions to an FTBL file
- added –addnoise option to facilitate creation of realistic simulated measurements

Bug fixes:

- fixed a bug preventing Monte-Carlo simulations with influx_i (reported by Maria Fatarova, INSA, Toulouse, France)

2016-07-29 version 3.2

New features:

- added controls for coherence of label transitions
- added detection of incoherent fragments in MASS_MEASUREMENTS (e.g. longer one than a whole molecule)
- in LABEL_INPUT section, if incomplete labeled forms don't sum up to 1, and several labeled forms are absent, the lacking label fraction is assigned to the fully unlabeled form
- R package 'snow' is no more needed on windows platform to run Monte-Carlo simulations in parallel mode
- on all platforms, Monte-Carlo simulations are now run on a PSOCK cluster and no more on a FORK cluster (Linux) or SOCK (Windows)
- inequalities involving only constrained fluxes or depending solely on such fluxes are now simply ignored with a warning
- fixed Jacobian calculation when no free flux exists

Bug fixes:

- fixed building a library mult_bxxc.dll on Windows platform (reported by Tony Palama, INSA/MetaToul Toulouse, France)
- fixed building mult_bxxc.so in parallel context (multiple ftbls)
- fixed formulas in equalities and inequalities with flux names having parenthesis, brackets, spaces and alike
- cluster workers are parsimoniously created in case of multiple starting points

2016-06-13 version 3.1

New features:

- added controls and explicit error messages for DEVIATION=0 in FTBL file
- added column "p-value" to residual values in _res.kvh file (may help for outlier choice)
- added check-points for infinite values that can appear in residual and Jacobian

Bug fixes:

- fixed EMU mode in instationary case
- fixed renewal of mult_bxxc.so library in case of source update
- fixed cost value calculation in case of outlier exclusion

2016-04-18 version 3.0.1

Bug fixes:

- fixed including mult_bxxc.so file in zip archive which prevented from proper compiling of this dynamic library
- fixed an absence of C++11 flag on platforms where it is not a standard by default.

2016-04-15 version 3.0

New features:

- influx_i.py is introduced for instationary label modeling
- some critical calculations are written in C++ so some compilation is needed at first execution.
- new optional package limSolve is used and need to be installed (as well as its dependencies) if `--lim` option is used
- more TIMEIT points introduced for finer time control

Bug fixes:

- fixed performance issue in slam package for '-' and '+' operations
- fixed sparse matrix preparation when there is only one non zero entry

2016-02-18 version 2.15

New features:

- calculation speed was improved due to the use of packages slam and rmumps instead of Matrix;
- added "cpu" field when timing is requested

Bug fixes:

- fixed a bug preventing a use of a flux added in EQALITIES in measured fluxes (reported by Edern Cahoreau, INSA, Toulouse, France)
- fixed minor problems in MC iterations (parameter distribution was not significantly affected)

2015-01-19 version 2.14

New features:

- `commandArgs` field in FTBL file can have comments in it and occur more than once somewhere in the `OPTIONS` section
- `--DEBUG` option is removed as obsolete
- R package `bitops` is not required anymore to be installed (valid for R-3.0.0 or higher)

Bug fixes:

- fixed a bug in delivering an error message when `commandArgs` had a comment in it
- fixed the precedence of command line options over `commandArgs` given in FTBL
- fixed a bug in parsing FTBL file having a BOM (invisible utf8 encoding mark) in it (reported by Yanfen Fu, University of Washington, USA)
- fixed representation of growth fluxes by ftbl2xgmml.py utility

2014-09-17 version 2.13

New features:

- `posttreat_R` field can have several file names separated by `'; '`
- added explicit error message if a valid float value is missing for free or constrained flux
- added explicit error message if no dependent flux is included in the balance on any metabolite (suggested by a case submitted by Marc O. Warmoes, Cornell University, USA)
- in the documentation, added a paragraph about consulting offer
- result .kvh file is greatly shortened, keeping only essential information. Custom additional information can be stored in some file via `posttreat_R` option
- now, `influx_s` returns a non zero code to shell if an error occurred during execution;
- added a parameter `monotone` to the control list of NLSIC.

Bug fixes:

- fixed a bug in generating EMU systems (manifested in some special cases)
- fixed an error preventing from producing a message suggesting a new partition among dependent, free and constrained fluxes (reported by Stéphane Mottelet, University of Compiegne, France)
- fixed metabolite pooling weights (manifested in some special cases)
- Windows platform: fixed passing command line options to R code
- Windows platform: precompiled `nnls` R package (version 32 bits) can produce wrong results. Recompile it by hand or use 64 bits version.

2014-07-02 version 2.12

New features:

- parsing badly formatted ftbl files is made more robust

Bug fixes:

- fixed a bug in `--emu` option (was introduced in v.2.11)

2014-06-12 version 2.11.1

Bug fix:

- an option `--noopt` broken in 2.11 is repaired (reported by Pierre Millard, Manchester Institute of Biotechnology, UK)

2014-06-11 version 2.11

New features:

- a joint use of the options `--fseries` and `--irand` gives a possibility to mix fixed and random values in starting points

- post treatment option `posttreat_R` is introduced in FTBL file. A user script written in R can be used to chain flux estimation and customized data treatment, e.g. graph plotting in a pdf file or simply saving of all the environment for later use and exploring in an R interactive session
- added optional `INEQUALITIES/METAB` section in FTBL file. It can be helpful to limit variations of estimated metabolite concentrations (suggested by Marc Carnicer, INSA of Toulouse, France)
- added optional `EQUALITIES/METAB` section in FTBL file. It can be helpful to fix a ratio between varying metabolite concentrations (suggested by Marc Carnicer, idem)
- the default value of `btdesc` parameter in NLSIC algorithm is lowered from 0.75 to 0.1. In some cases, it can accelerate the optimization convergence.

Bug fixes:

- fixed EMU list of participants in measurements
- fixed measurement matrix when only one measurement is available
- fixed a fatal error when no free flux is available but at least one metabolite quantity must be estimated
- fixed a bug in pooled measurements. This bug was harmful only if the metabolite pooling was used in more than one type of measurements, e.g. mass *and* labeling. If only one type of measurements used pooling (e.g. mass), the bug was without effect
- where appropriate, a word "labeling" was replaced by "label" in the field names of the `_res.kvh` file
- fixed superfluous backtracking iterations present for some particular residual functions
- if a flux or a metabolite is present more than once in formulas of (IN)EQUALITIES sections, its coefficients are summed up instead of taking only the last one
- fixed a fatal error in generating inequality matrix for net fluxes

2014-04-08 version 2.10

New features:

- added an option `--tikhreg` which is an alternative for `--ln` option. In case of rank deficient Jacobian, it calculates an increment step of the smallest norm in *approximative* way. It is done by Tikhonov regularization
- added an option `--ffguess` which makes to ignore the partition between free and dependent fluxes defined in FTBL file(s) and automatically guess a new free/dependent flux partition (suggested by Roland Nilsson, Karolinska Institutet, Sweden)
- added utility `ftbl2kvh.py` which is useful for debugging purposes only
- utilities `ftbl2xgmml.py`, `ftbl2cumoAb.py`, `ftbl2netan.py` and `ftbl2kvh.py` are rewritten in such a way that if no output redirection (with operands '>' or '|') occurs on the command line, the name of the output file is automatically derived from the input one. The suffix `.ftbl` is simply replaced with `.xgmml`, `.sys`, `.netan` or `.kvh` respectively. Thus a plain drag-and-drop can work with these utilities
- option `--TIMEIT` reports times with subsecond precision. The actual precision depends on the platform but typically a 0.01 s precision should be available. On Windows, the precision is usually 1/60 of a second

Bug fixes:

- fixed `include_growth_flux` option for `ftbl2cumoAb.py` utility (reported by Marc Carnicer, INSA of Toulouse, France)

- fixed a bug preventing from checking for a linear dependence between rows of stoechiometric matrix if no constrained net flux is defined in the FTBL file (reported by Roland Nilsson, idem)

2014-02-05 version 2.9

New features:

- utility `ftbl2xgmml.py` replaces `ftbl2rsif.py`. Now, a standalone XGMML file describes both a network and its graphical properties instead of a collection of files where this information was spread. New graphical conventions are now used.
- an obsolete utility `ftbl2cytoscape.bat` is removed form the distribution.
- added utility `res2ftbl_meas.py` generating measurement section from a result file _res.kvh
- added utility `expa2ftbl.R` transforming stoechiometric information in EXPA format (http://gcrg.ucsd.edu/Downloads/ExtremePathwayAnalysis) to various sections of FTBL file, namely to `EQUALITY` section where non carbon carrying fluxes can appear
- files generated by `influx_s` and collecting values for graphical representation (like `edge.netflux.mynetwork` and others) are renamed by adding a suffix `.attrs` to make them compatible with Cytoscape v3.0
- utilities `ffres2ftbl.sh` and `ff2ftbl.py` distributed for a long time ago, are now mentioned in the documentation

Bug fixes:

- fixed `--fullsys` option broken in the previous release.

2014-01-27 version 2.8

New features:

- `EQUALITY` section in FTBL file may include fluxes absent in `NETWORK` section, e.g. fluxes involved in non carbon carrying reactions (suggested by Roland Nilsson, Karolinska Institutet, Sweden)
- when a meaningful partition between free and dependent fluxes cannot be made, a proposition is made as to stoechiometric equations to be eliminated by hand (suggested by Roland Nilsson, idem)
- when `--clownr` option is used, reduced size of cumomer system is more efficient than without this option (replace a fix in 2.6 version)

Bug fixes:

- fixed useless memory consumption during ftbl parsing when `--emu` option is used and very long molecules (say >20 carbons) are present (reported by Roland Nilsson, idem)
- some error messages are made more explicit during FTBL parsing
- fixed Jacobian calculation for condensing input reaction
- fixed matrix constructions when no free flux is defined
- fixed b term for full cumomer system
- fixed inequality enforcement when adaptive backtracking is used in NLSIC
- fixed inequality precedence, now specific inequalities from FTBL file prevail on `--cupn=CUPN` option

2013-10-22 version 2.7

New features:

- Monte-Carlo simulations are done in parallel on Windows platform too (needs R package `snow`)
- if the option `--seed=SEED` is used, Monte-Carlo simulations are now reproducible even if run in parallel on multiple cores
- for rank deficient Jacobian, the inequalities are now better enforced
- starting value for `maxstep` parameter is set to $10||p||$ instead of $||p||$ where p is a vector of starting values for free parameters to fit.

Bug fixes:

- fixed a bug preventing to report partial Monte-Carlo results if some simulations failed and some not
- fixed a bug making to use all available cores instead of only one when `NP` was set to 1
- fixed a fatal error when inequality enforcement fails
- error and log messages during zero cross passes are made more explicit
- fixed sending some error messages on standard output instead of `.err` file
- when cumomer matrix is singular, fixed an error message about zero fluxes

2013-10-02 version 2.6

New features:

- added option `--sln` (solution least norm) which applies 'least norm' to the whole solution vector of free parameters, not just to the increment vector (like `--ln` does)
- a parallel calculation of multiple FTBLs is moved from python to R code. In such a way, some economies of repeated R starting up and library loading are made
- when zero crossing is used (`--zc=ZC`) a third pass is added without any zc constraint.
- added an option `maxstep` to control list of nlsic(). In some situations, it can make the convergence more stable at early iterations.

Bug fixes:

- fixed a fatal error preventing from using BFGS optimization method
- fixed an error in calculating reduced size of cumomer or EMU systems. It did not impact the results (at least for well defined network) but made calculations a little bit longer (reported by Stephane Mottelet, University of Compiegne, France).
- a more explicit error message is generated when a given choice of free fluxes leads to a square but singular flux (stoechiometric) matrix.
- some error messages were printed on standard output instead of `.err` file.

2013-06-28 version 2.5

New features:

- an argument of the option `--np=NP` (number of processes) can be fractional, between 0 and 1 in which case the number of requested cores is calculated as NP*number_of_available_cores
- in documentation, added a section describing some problematic cases and measures which could be undertaken to solve or to work around them. Few more field names in the output file are described (based on discussions with Yanfen Fu, University of Washington, USA)
- missing values in measurements (NA as Non Available) are allowed in FTBL files.

Bug fixes:

- fixed a fatal error if the rights of generated R file cannot be changed
- fixed a bug for `--ln` (least norm) option when without inequalities, increments were not of least norm (reported by Stephane Massou, INSA of Toulouse, France)
- fixed an algorithm used in `--ln`. Now for all inequality systems, both least residual norm and least solution norm are achieved (before, for some systems it was not the case). **Due to this fix, we highly recommend to update to this version if you use `--ln` option**
- fixed a bug in "zero crossing" inequalities. Now, inequalities involving only constrained fluxes are canceled.

2013-04-11 version 2.4

New features:

- number of parallel processes (in case of multiple FTBL files) is limited to a number of cores or to an argument of the `--np` option
- some consistency controls were added on flux names in various FTBL sections.

Bug fixes:

- fixed a bug in formatting some error messages during FTBL parsing;
- fixed an accidental removing of kvh.py file from the previous release;
- fixed non signaling to check `.err` file while some parsing errors did produce;

2013-03-28 version 2.3

New features:

- external `multicore` R package is replaced by native `parallel` package;
- convergence information of Monte-Carlo simulations is reported in the result file;
- relative SD (rsd) in Monte-Carlo statistics is calculated as SD/abs(mean) and no more as SD/abs(estimated parameter);
- if the number of really calculated samples in Monte-Carlo is less than 2, statistics are not calculated;
- R code is self sufficient to be executed via source() function, even in parallel way;
- with a new option `--nocalc`, R code is generated but not executed.

Bug fixes:

- fixed concurrent access to a global variable in Monte-Carlo parallel execution;
- fixed scope issue in Monte-Carlo simulations preventing from update of the current solution;

- fixed some redundant warning messages;
- fixed placement of .err and .log files if FTBL(s) is (are) given with subdirectories in their names.

2013-03-15 version 2.2.1

Bug fixes:

- fixed a fatal error in Jacobian matrix construction when no measured fluxes are provided in FTBL file (reported by Yanfen Fu, University of Washington, USA);
- in the User's manual, added a naming convention for variable growth fluxes.

2013-03-13 version 2.2

New features:

- if more than one FTBL file is given in argument to `influx_s`, all files are proceeded simultaneously in independent processes;
- outliers in measurements can be automatically detected and excluded from parameter fitting.

Bug fixes:

- fixed an error preventing Monte-Carlo results to be written if `multicore` package is not installed;
- fixed a documentation error about $\ln(M)$ in `mynetwork.pres.csv` file;
- fixed warning resuming if there are many of them;
- fixed some error message generation on FTBL parsing.

2013-02-15 version 2.1

New features:

- in nlsic() a new field 'retres' is added to the list of returned values. It transfers "as is" the list returned by a last call to residual calculation function;
- added a writing of generalized inverse of Jacobian to the result file;

Bug fix:

- fixed a typo preventing Monte-Carlo statistics on forward-reverse fluxes to be written in the result file.

2013-02-05 version 2.0

New features:

- metabolite pooling is modeled. Such pooling can appear due to compartmentation phenomenon or due to isomer coelution in chromatography. Starting from this version, metabolite concentrations can be part of fitted parameters;
- adaptive backtracking algorithm is introduced to NLSIC algorithm;
- history of convergence during minimization can be retrieved;

- symbolic equations for dependent fluxes expressed as functions of free and constrained fluxes are generated by ftbl2cumoAb.py script;
- METAB_MEASUREMENTS section is added to FTBL format;
- added χ^2 test for evaluating the goodness of fit;
- removed `metab_scale` field from OPTIONS section in FTBL format;
- "dead end" internal metabolites are allowed in a network without being an output metabolite. As consequence, input-output fluxes must be explicitly declared as non reversible in the FTBL;
- added optional EMU framework (`--emu`);
- added optional series of starting points, fixed or randomly generated (`--fseries`, `--iseries`);
- matrix construction is reworked and fortran code is removed. Now, no more `Rtool` installation is required for running `influx_s`;
- some error messages are made more explicit and more precise;
- outdated R package `fUtilities` is no more required;

Bug fixes:

- fixed stoechiometric matrix construction when for a given metabolite; all fluxes are free or constrained;
- fixed candidate propositions for free fluxes;
- fixed standard deviation value for a `DD/T` field in `PEAK_MEASUREMENTS` section.

2011-10-11 version 1.0

Initial release. Main features:

- NLSIC algorithm;
- FTBL input format from 13CFlux project;
- reduced cumomer set for cumomer balance equations;
- sparse matrices;
- usage of `multicore` R package for Monte-Carlo simulations on Unix platform;
- usable on platforms having Python+numpy and R+some modules;
- command line interface;
- brief user's and programmer's documentation;
- OpenSource (ECL) license.

CHAPTER

THREE

INSTALLATION

The software was developed on Linux but can be used both on Linux (or other UNIX, MacOS included) and Windows platforms.

Note: The code examples here after are given for Unix shell environment. On Windows, in PowerShell or DOS environment the syntax is often similar and in cygwin or Ubuntu environment (Unix tools on Windows) the syntax is identical to the Unix's one.

Note: In command examples to run, we use script names with extension *.py*. However, starting from version 5.0.3, this extension can be omitted as all Python scripts are doubled with executable files without '.py'. For example, commands:

```
$ influx_s.py --prefix e_coli
```

and

```
$ influx_s --prefix e_coli
```

are now equivalent. Even if it works on all platforms, it can be particularly useful for Windows, where supplementary effort can be required to associate .py file with Python interpreter. Using executable programs (i.e. without .py extension) makes this extra configuration step no more mandatory.

Installation with `conda`

If you have Anaconda or Miniconda installed on your system, installation of `influx_si` resumes to:

```
$ conda install influx_si -c conda-forge -c bioconda
```

It installs `influx_si` itself as well as all needed dependencies both in Python and in R.

Installation with `pip`

If you don't have any version of `conda` (neither `miniconda` nor `Anaconda`) but do have a Python and R installed on your system, you can install `influx_si` with the following procedure.

You need a python tool called `pip` which manages pure python packages. If it is not present on your system, you'll have to install it first to continue with this method. If you have multiple Python versions installed on your system (e.g. Python2 and Python3) you'll have to use `pip3` to install the software in the Python3 universe.

The first step will install only Python part of `influx_si`:

```
$ pip3 install influx_si
```

or

```
$ pip3 install --user influx_si
```

if you wish to install `influx_si` not system-wide but only in your own userspace.

To use the software `influx_si`, you'll need some R dependencies listed below. You can try to install them by:

```
$ influx_s.py --install_rdep
```

If this procedure fails, you'll have to solve the underlying problem identified from its error messages and rerun the command again.

R dependencies

As of influx_si version 5.0, user has not to install R dependencies manually from an R session. So they are listed here just for information.

- R-3.4.0 or higher (cf http://www.r-project.org/ or your system packaging solution) + the following packages.
  - nnls
  - rmumps (5.2.1-6 or higher)
  - arrApply
  - slam
  - limSolve (optional, needed only for `--lim` option)
  - multbxxc

Warning: As of this writing (September 17, 2014), an R package `nnls` distributed in precompiled form on Windows platform, can produce wrong results if a 32 bits version is used on Windows 64 bits. To avoid this, use 64 bit version of R on Windows 64 bits or recompile it by hand. To be sure to use 64 bits version of R, check that the `Path` system variable has the R path ending by `\bin\x64` and not just by `\bin`.

Python dependencies

As of influx_si version 5.0, user has not to install Python dependencies manually. So they are listed here just for information.

- python 3.6 (or higher) and modules
  - scipy
  - libsbml (optional, needed for ftbl2metxml.py)

Test of installation

Open a shell window and get to your working directory. Copy the distributed test directory to the current directory by running

```
$ influx_s.py --copy_test
```

then you can get in the newly created directory `test` and run some tests:

```
$ cd test/mtf
$ influx_s.py --prefix e_coli
```

If everything was correctly installed, you should see in your shell window an output looking like:

```
"/home-local/sokol/.local/bin/influx_s" "--prefix" "e_coli"
code gen: 2022-05-25 12:10:53
calcul  : 2022-05-25 12:10:53
end     : 2022-05-25 12:10:55
```

The meaning of this output is quit simple. First, an R code is generated from input MTF files (cf. *MTF format* for more details) then it is executed till it ends. Time moments at which these three events occur are reported.

The calculation result will be written in `e_coli_res.kvh`. It should be almost identical to the same file in `ok/mtf` subdirectory. On Unix you can do

```
$ diff e_coli_res.kvh ../ok/mtf/e_coli_res.kvh
```

to see if there is any difference. Some small differences in numerical values can be ok. They might come from variations in versions of R and underlying numerical libraries (BLAS, LAPACK and so on).

If something went wrong, check the error messages in `e_coli.err`, interpret them, try to figure out why the errors occurred and correct them.

In high throughput context, you can find it useful to run `influx_si` in parallel on many independent MTF sets. It can be done by providing more than one `--prefix` options. For example, with two of cases provided with the package you can run:

```
$ influx_s.py --prefix e_coli --prefix e_coli_growth
```

In this case, the output looks sightly different than in one by one run:

```
"/home-local/sokol/.local/bin/influx_s" "--pref" "e_coli" "--pref" "e_coli_growth"
e_coli_growth: code gen: 2022-05-25 14:44:56
e_coli: code gen: 2022-05-25 14:44:56
//calcul: 2022-05-25 14:44:57
//end   : 2022-05-25 14:44:58
```

The time moments for code generation is preceded by a short version of file names. The symbol `//` means parallel proceeding. Parallel calculations are launched after all files are proceeded for the code generation.

It is the operating system that dispatches and equilibrates the charge among available CPUs and cores, not `influx_si` who simply launches these processes.

One of the main interest of MTF format is an ability to multiplex constant and variable parts of information describing a set of experiments. In this case, many calculations can run in parallel on inter-dependent input files, cf. `.vmtf` description in *MTF format*.

For a quick test of `influx_i`, you can run in the same directory:

```
$ influx_i.py --prefix e_coli_i
```

Normal output looks like

```
"/home-local/sokol/.local/bin/influx_i.py" "--pref" "e_coli_i"
code gen: 2022-05-25 14:50:51
calcul  : 2022-05-25 14:50:52
end     : 2022-05-25 14:51:02
```

Calculation results are written in `e_coli_i_res.kvh` and they can be compared with the same file in the `ok/mtf` sub-directory. You can also visually check a generated graphic file `e_coli_i.pdf` to see if all simulated label kinetics based on estimated fluxes and metabolite concentrations are close to experimental data.

Installation of documentation

`influx_si` is distributed with its documentation. To get it locally accessible from your personal disk space, you can run:

```
$ influx_s.py --copy_doc
```

It will create a subdirectory `doc` in the current directory. This subdirectory contains `influx_si.pdf`, all-in-one documentation file but also an `html` subdirectory with the documentation browsable in your prefered navigator.

The both documentation versions are also available on-line: pdf and html.

For a quick reminder of available options, launch

```
$ influx_s.py --help
```

or

```
$ influx_i.py --help
```

depending on what context you want to treat: stationary or instationary labeling.

For more detailed documentation, read *User's manual*.

CHAPTER

FOUR

QUICK START

A basic work-flow with `influx_si` is composed of the following steps:

1. Create a MTF file set (Multiple TSV Files) describing your metabolic reactions and carbon transitions (.netw), experimental data (.miso) label input (.linp) and some non mandatory measurements options (.mflux, .mmet, .tvar, .cntsr, .opt). Let an example MTF set has a prefix `mynetwork`. The syntax rules for reactions will be more or less obvious to someone working on metabolism biochemistry. So, to go quickly, you can inspire from example files `test/mtf/e_coli.netw` and co. distributed with the `influx_si` software (run `influx_s --copy_test` to bring them to your current directory). You can also consult the help message from `txt2ftbl -h` for `--mtf` option.

   Note: `NA` values (as "Non Available") are admitted as measurements values where appropriate. The difference with the situation where measurements are simply omitted is that NA measurements are simulated and are present in the vectors `simulated unscaled labeling measurements` and `simulated scaled labeling measurements` in the result kvh file.

   Note: In case of `influx_i`, label kinetics can be provided in .miso file using non-empty `Time` column. Empty cells in `Value` column are equivalent to `NA`.

2. Set your current directory to the directory of `mynetwork.*` and run

   ```
   $ influx_s.py --prefix mynetwork
   ```

   or

   ```
   $ influx_i.py --prefix mynetwork
   ```

   depending on stationary or instationary labeling context. We suppose here that directory of `influx_si` binaries is in the PATH variable.

   An `influx_si` run will produce the following files in the same directory that `mynetwok.*`:

   `mynetwork.ftbl` FTBL is a previously used format as a front-end format. It is still in use but behind the scenes. This file can be necessary as entry for `ftbl2*` utilities.

   `mynetwork.log` contains the run-time output from various scripts, in particular, it contains a report on convergence history during the fitting process. It can be helpful for identifying potential problems, but if everything is going well, the user does not have to examine the content of this file;

   `mynetwork.err` contains the warning and error messages. Normally, this file should be empty (0 byte size);

mynetwork_res.kvh contains all the results. KVH format is a lightweight plain text format for hierarchically structured data. It can be seen in a text editor or in a spreadsheet software as its fields are tab separated. It can also be processed by user's custom software for post-processing, graphics output and alike. If `influx_si` is run on a series of starting points, there will be generated a common result file `mynetwork_res.kvh` which contains common information to all starting points but also a series of kvh files, one by starting point, e.g. `mynetwork_res.V1.kvh`, `mynetwork_res.V2.kvh` and so on;

mynetwork.pres.csv contains a matrix of fitted parameters and final cost values. Each column corresponds to a particular starting point if run with `--fseries` and /or `--iseries` options. If `influx_si` was run without these options, the file will contain only one column corresponding to the starting point defined in the `mynetwork.tvar` file or to the random starting point.

edge.netflux.mynetwok, edge.xchflux.mynetwok, node.log2pool.mynetwork as the middle name of these files suggest, they can be used to map the corresponding values on the network graph in the cytoscape software.

Note: All these files are silently overwritten if already exist. So take care to copy your results elsewhere if you want to protect them from overwriting.

---

custom files (e.g. mynetwork.pdf) These files can be produced by user supplied scripts that are executed at the end of `influx_si` simulations. For example, we provide a script `plot_ilab.R` which can be used to plot label kinetics simulated by `influx_i`. One or many of such custom scripts can be given in .opt file, field `posttreat_R` (cf. e_coli_i.opt for example)

Note: It can be helpful to do some "dry runs" by executing

```
$ influx_s.py --noopt --pref mynetwork
```

before collecting actual measurement data to see if intended measurements will be sufficient to well define all fluxes, or at least the fluxes of interest. It is possible to do so because the measurement values in the .miso file have no impact on flux SD calculation when `--noopt` option is used. So it can be used any values, even NA at this moment. On the contrary, `SD` values set in .miso file, must be realistic. It is generally not a problem as they express measurements errors and are more or less known for a given measurement method.

It is worthwhile to stress that a "dry run" is done for some presumed free flux values. If they reveal to be very different from actual flux values, it can happen that a network considered as well defined at moment of "dry run" turned into a badly defined network with actual measurement data and corresponding estimated fluxes. So it is important to do his best to guess the most realistic free fluxes for "dry runs" and log their values in .tvar file.

---

3. See warning and error messages in `mynetwork.err` if any. Correct what has to be corrected and retry p. 2
4. Extract and use the numerical results from the `mynetwork_res.kvh` file.
5. Optionally, visualize net fluxes (or exchange fluxes or logarithm of metabolite concentrations $\log_2(M)$) in cytoscape using `ftbl2xgmml` to produce a .xgmml file and then mapping `edge.netflux.mynetwok.attrs`, `edge.xchflux.mynetwok.attrs` or `node.log2pool.mynetwork.attrs` on graphical attributes like edge width, color etc. in cytoscape.

CHAPTER

FIVE

USER'S MANUAL

MTF format

Let start by describing file formats used by `influx_si`. MTF (Multiple TSV files) format was introduced in v6.0 of `influx_si`. Its introduction came in replacement to FTBL format and served several purposes:

- to simplify network and other information formatting;
- to allow multiplexing of constant and variable information for large experience series. E.g., a same network can be tested under several biological conditions or vice versa a given data set can be confronted to different network models to see which one can better fit it;
- to facilitate automatic assembling of network/data/options coming from tiers workflows such as MS or NMR data treatment.

Even if FTBL format was replaced as front-end format in `influx_si`, it is still a valid format for calculation and continues to be used behind the scene on in accompanying utilities.

MTF format is composed of a series of plain text files, each having a particular role and distinguished by their suffixes:

.netw describes biochemical reactions and label transitions.

.linp describes label inputs forms and fractions

.miso describes stationary and instationary label measurements, their type, value, and standard deviation

.mflux describes stationary net flux measurements (e.g., substrate consumption or product secretion)

.mmet describes stationary concentration measurements

.cnstr describes constraints, equality or inequality type, on fluxes and concentrations

.tvar describes variable type such as Free, Constrained or Dependent with possible starting values for Free variables

.opt describes options such as optimization parameters, post-treatment R scripts to execute and others

These suffixes will be used throughout this manual, as they facilitate `influx_si` usage. But they are not mandatory. For example, a classical .tsv or .txt suffixes can be used instead. In this case, a file meaning can be indicated in `--mtf` option of `influx_si` of `txt2ftbl`, e.g. `--mtf netw=ecoli.txt,miso=gcms.tsv,linp=glucoseU.8.tsv`

Only the first 3 are mandatory, the rest of MTF set is optional.

Now, we pass in more detailed review each of the file content.

.netw

This is a central file to the rest of the work – network file .netw It represents a list of biochemical reactions with label transitions. Here is an example of such reaction:

```
edd: Gnt6P (ABCDEF) -> Pyr (ABC) + GA3P (DEF)
```

Let detail its elements (all reaction elements are case-sensitive):

edd the unique reaction name

: name-reaction separator

Gnt6P, Pyr, GA3P reactant names. They must not contain +, (,). Other common separators such as **:**, **;** and special characters such as {, |, ... are to avoid. However, other than Latin alphabet letters are welcome in UTF-8 encoding, e.g. α, β etc.

(ABCDEF), (ABC), (DEF) are labels of reactants. Here, their meaning is that first 3 carbons of Gnt6P goes to Pyr in the same order and the rest goes to GA3P also in the same order. Label atom numbering is left free for user's choice but once chosen, it must remain consistent between reactions. It is advised to follow some common conventions. E.g., the most oxidized carbon atom (if it is a carbon which is used for labeling) should have number 1 (here letter A in Gnt6P and Pyr, and letter D in GA3P). Its neighbor has number 2 etc. influx_si does not impose to use ^{13}C as labeling atom. User is free to use any other atom or even a combination of them, e.g., ^{13}C and ^{15}N. A reactant can be without label atoms. In this case, it will participate in mass balance but not in label balance. Such situation can be useful e.g., for co-factors, e.g., ATP/ADP, NADP/NADPH etc. if they are not synthesized in the modeled network. The left opening parenthesis of the label pattern must be separated by a space from a precedent reactant name. The label identifiers can be composed of Latin or foreign letters as well as digits, each symbol representing one label atom. If a label symbol is present on one reaction side, it must be present as well on the other side. Each symbol must be present only once, except for so called "scrambling" label molecules. For example, Formate is a symmetric molecule from carbon point of view. So, its labeling can be given as (ABCD+DCBA). Thus, for example, a letter A appearing on the other side of reaction can come from both ends of Formate.

-> reaction left and right side separator. Here, the reaction is indicated as non-reversible, i.e., exchange flux is imposed to be 0. However, the net flux can be either positive or negative. For a positive net flux, the reactants on the left are consumed and those on the right are produced. If user wishes to impose the sens of reaction, a sign ->> can be used. In this case, the net flux is imposed to be non-negative (i.e. ≥ 0). For reversible reactions, a sign <-> can be used. A sign <->> is meaningful. It indicates that a reaction is reversible, but the net flux must be non-negative. A non-zero exchange flux for a positive net flux can be responsible for a backward label propagation.

Input/output reactions must have exchange flux set to 0. So they can be edited either with -> or ->>. This is necessary to distinguish them from reactions having so-called "dead-end" reactants. They occur only on one side of reaction(s) (either left or right, like input/output reactants do) but have an exchange flux different from 0. The dead-ends are rarely desirable and most often result from network topology errors or simply leaving <-> where -> is meant. However, there are some special situations when they can be used on purpose in influx_i, for example, for modeling label dilution from stock species.

+ reactant separator. The surrounding spaces are mandatory.

In the above reaction example, all stoichiometric coefficients are 1. If it is not the case in some reaction, they can be given as plain number (integer or with decimal point) preceding a reactant name and separated by an optional * sign (it can be replaced by a space), e.g.:

```
v47:  Ser (abc) + AcCoA (de) + 3.0*ATP () + 4.0*NADPH () + SO4 () -> Cys (abc) +␣
↪Ac (de)
```

Note that coefficients different from 1 can only be used with non labeled reactants.

A comment can be introduced by `#`. The line content starting from this character to the end of the line is simply ignored with one exception: a triple hash sign `###` at the line beginning is used to introduce a pathway name. Pathway name can be useful for `ftbl2metxml.py` script which prepare xml and txt files for visualization on a partner site MetExplore.

.linp

Label input can be indicated in .linp file. Starting from this file type, the rest of the files are in TSV (tab separated values) format. I.e. they are plain text files where data are organized in tables, one table row per file line and where columns are separated with the tabulation character. The first non commented row contains column names. The comments start with # sign, they are simply ignored till the end of the row where they occur. The left- and right-trailing white spaces are stripped.

The .linp file can contain the following column names:

`Id` Not used by `influx_si` but can be useful for user's information tracking system.

`Comment` Not used by `influx_si` but left for user's convenience.

`Specie` Chemical specie name such as used in reactions in .netw file, e.g. `Glucose`

`Isotopomer` Particular isotopomer form used as label entry and composed only of "0"s and "1"s, e.g. `111111` for uniformly labeled Glucose, `100000` for Glucose labeled in first carbon.

`Value` For step-wise labeling experiments, a number between 0 and 1 indicating a fraction of the given labeled form in the mix. For experiments with labeling varying in time, a R expression describing time dependent function for the given label form.

Normally, all labeling form for a given specie must sum up to 1. If they don't, the following conventions apply:

- *"the rest is unlabeled"*: if many labeling forms are lacking in the file (including fully unlabeled specie) then the fully unlabeled form (e.g. `000000` for Glucose) is considered as completing the set to 1;
- *"guess the lacking one"*: if only one form is lacking in the file (no matter which one), then its fractions is considered as completing the present set to 1.

Here is a complete example:

```
Id      Comment Specie  Isotopomer      Value
                Gluc_U  111111          1
                Gluc_U  000000          0.
                Gluc_1  100000          1.
                Gluc_1  000000          0.
```

which can be shortened, due to above conventions, to:

```
Id      Comment Specie  Isotopomer      Value
                Gluc_U  111111          1
                Gluc_1  100000          1.
```

.miso

The .miso contains and describes labeled measurements. This file contains the following column names:

`Id` Not used by `influx_si` but can be useful for user's information tracking system.

Comment Not used by `influx_si` but left for user's convenience.

Specie Specie names such as used in reactions in .netw file, e.g. `Glucose`

Fragment Integer sequences or intervals describing label fragment used in a given measurement, e.g. `1-3` or `1,2,3` or `2-5,7,9-11`. Empty field means that the entire molecule is measured.

Dataset Any character sequence identifying measurement method. It can be useful for distinguishing measurements on the same combination specie/fragment. Examples: `MS-1`, `HSQC`. A given dataset can have its own scaling factor if they are in use.

Isospecies For MS measurements, a `M0`, `M1` etc. isotopologue identification. For NMR label measurements, a combination of binary cumomers involved in measurements. They are separated by "+" sign. Each binary cumomer can be composed of "0", "1" and "x" symbols, e.g. `01x+10x`. This notation is universal enough to describe any NMR (or even MS) method. However, for methods focused on a particular specie fragment, it can be more practical to use notation "label transferring" like `2->`, `2->1`, `2->3` and `2->1,3`. Here, the second atom is labeled and in a given measurement method it interacts:

2-> with no other atom (given a singlet peak);

2->1 with labeled atom 1 (giving a peak doublet 1);

2->3 then labeled atom 3 (giving peak doublet 2);

2->1,3 and finally with both labeled atoms 1 and 3 (doublet of doublets).

Value Measured value in floating point notation. Can be empty or NA, meaning "non-available". There is a difference between a measurement absent in file and a measurement with NA value. The former is simply ignored, while the latter is simulated and reported is simulated measurements.

SD Standard deviation value in floating point notation. Cannot be empty, neither NA. We recall that SD is characterizing a given measurement technique, not its particular realization. So it is perfectly possible to have only one measurements if SD of the given technique was already estimated from previous experiments. If the `Value` contains an average of n measurements, than a standard SD should be reduced by a factor $\sqrt{n}$

Time For instationary labeling, the time point to which a given measurement corresponds. For stationary labeling, must be empty.

A multi-line example is the following:

```
Id      Comment Specie  Fragment        Dataset Isospecies      Value           SD      ␣
↪  Time
                GA3P                    LAB-10  1xx             0.03304418      0.002
                GA3P                    LAB-11  x1x             0.01260362      0.002
                GA3P                    LAB-12  xx1             0.1207158       0.002
                PEP     1,2             PEAK-1  1->             0.66            0.005
                PEP     1,2             PEAK-1  1->2            0.01            0.005
                PEP     1,2,3           PEAK-2  2->             1.26            0.005
                PEP     1,2,3           PEAK-2  2->1            0.03960991      0.005
                PEP     1,2,3           PEAK-2  2->3            0.01183004      0.005
                PEP     1,2,3           PEAK-2  2->1,3          0.0007686513    0.005
                PEP     1,2,3           MS-1    M0              12601           68.005
                PEP     1,2,3           MS-1    M1              2301            16.505
                PEP     1,2,3           MS-1    M2              96              5.48
                PEP     1,2,3           MS-1    M3              1               5.005
```

Here, column `Time` is left empty intentionally, thus signaling a stationary labeling.

.mflux

Starting from this file format, we consider that column names are either similar to already described or are self-explanatory, and we will just give multi-line examples with few possible comments. So, for net flux stationary measurements, we could have:

```
Id      Comment Flux    Value   SD
                upt     1.02    0.05
```

.mmet

For stationary specie concentration, we could have:

```
Id   Comment Specie  Value                   SD
             Fru6P   0.4263681348074568      0.01
             GA3P    0.469998855791378       0.01
```

.cnstr

Constraints on fluxes and specie concentrations can look like:

```
Id      Comment Kind    Formula                 Operator        Value
                NET     Glucupt_1+Glucupt_U     ==              1
                NET     edd                     >=              0.0001
```

Column `Kind` indicates if a constraint is on net fluxes: `NET`; on exchange fluxes: `XCH` or on specie concentrations: `MET`. The `Formula` content must be a linear function of involved entities. Column `Value` can have either a float number or a simple Python arithmetic expression which evaluates to a float number.

.tvar

Types of variables can resemble to

```
Id      Comment Name    Kind    Type    Value
                upt     NET     F       1.02369
                emp1    NET     F       0.511098
                emp2    NET     D
                ppp2    XCH     F       0.778786
                ppp3    XCH     C       0.83932
```

The `Type` can be either `F` (for "free", requires a float number in `Value`), `D` (for "dependent") or `C` (for "constrained", requires a float number in `Value`)

.opt

Options passed to `influx_si` can be similar to:

```
Id      Comment Name                    Value
                dt                      1
                nsubdiv_dt              4
                file_labcin             e_coli_msen.txt
```

(continues on next page)

(continued from previous page)

```
                commandArgs             --noscale --TIMEIT --time_order=2 --zc=0 --
↪clowp 1.e-9
                optctrl:nlsic:errx      1.e-3
                optctrl:nlsic:maxit     50
                optctrl:nlsic:btmaxit   16
                optctrl:nlsic:btstart   1
                optctrl:nlsic:btfrac    0.5
                optctrl:nlsic:adaptbt   1
                optctrl:nlsic:monotone  1
                posttreat_R             plot_ilab.R
```

The meaning of each possible option is described in different sections of this manual.

.vmtf

Variable part of MTF approach can be used to combine constant and variable parts of experiments to launch a calculation of flux maps in a batch. E.g. in a set of experiments on the same organism in different biological conditions, `.miso`, `.mflux` can vary from one experiment to another while `.netw` and other files can remain the same in the whole experiment set. In this case, files containing variable sections (here `.miso` and `.mflux`) can be given in a special file having an extension '.vmtf' while constant parts will be given in `--mtf` or `--prefix` options.

`.vmtf` file is a TSV file with columns using the same name as extensions described above: `netw`, `linp`, etc. Each row contains file names that will be used to produce a particular FTBL file used in calculation. Thus, each row must have `ftbl` column with unique and non empty name. If a file type is present both in column names of 'vmtf' and in `--mtf`/`--prefix` option then the content of 'vmtf' file will take precedence. Empty values in `vmtf` file are ignored. All file paths in `vmtf` file are considered relative to the location of `vmtf` file itself. If in `.vmtf`, a file name is given without extension, it is deduced from column name. Example of `.vmtf file`:

```
Id      Comment miso            mflux           tvar            ftbl
                model_WT_BW_1   model_WT_BW_1   model_WT_BW_1   vmtf_WT_BW_1
                model_WT_BW_2   model_WT_BW_2   model_WT_BW_2   vmtf_WT_BW_2
                model_zwf_1     model_zwf_1     model_zwf_1     vmtf_zwf_1
```

Example of command line using `.vmtf`:

```
--prefix ecoli --mtf variable.vmtf
```

Basic influx_si usage

`influx_si` can be run as

```
$ influx_s.py --prefix mynetwork
```

or

```
$ influx_i.py --prefix mynetwork
```

Letters `s` and `i` stand for "stationary" and "instationary". We suppose here that a valid MTF file set was created. Moreover, we supposed `influx_s.py` and `influx_i.py` are in the PATH variable.

In the rest of this manual, we'll use just `influx_s.py` as an example if the example is valid for both stationary and instationary contexts. If some usage is valid exclusively for `influx_i.py`, it will be duly signaled.

In a high throughput context, it can be useful to proceed many MTF set files in parallel. This can be done by giving all variable parts of experiment set in a `.vmtf` file, e.g.

```
$ influx_s.py --prefix mynetwork --mtf variable.vmtf
```

All files are then proceeded in separate independent processes launched almost simultaneously by a bunch of size equal to the number of available or requested cores (if an option `--np=NP` is used). It is an operating system who is in charge to make a distribution of all these processes among all available CPUs and cores.

Sometimes, particular cases need usage of special options of `influx_si`. The list of available options can be seen by running:

```
$ influx_s.py --help
```

If used with options, `influx_si` can be run like

```
$ influx_s.py [options] --prefix mynetwork
```

where `[options]` is an option list separated by a white character.

Note: Here and throughout this manual, content placed in brackets `[...]` is meant to be an optional part of the command. If the user does wish to type an optional part of a command, the brackets themselves must be omitted.

Each option starts with a double dash `--` and can be followed by its argument if applicable. For example, to use BFGS optimization method instead of the default NLSIC algorithm, a user can run:

```
$ influx_s.py --meth BFGS --prefix mynetwork
```

or

```
$ influx_s.py --meth=BFGS --prefix mynetwork
```

The option names can be shortened till a non-ambiguous interpretation is possible, e.g., in the previous example, the option could be shortened as `--me BFGS` or `--me=BFGS` because there is no other option name starting by `me`. But an option `--no` could not be distinguished between `--noopt` and `--noscale`. So at least `--nos` (for `--noscale`) or `--noo` (for `--noopt`) should be provided. There is only one option that does not admit a usage of an equal sign to provide an argument, it is `--excl_outliers`. Use only a space character to provide an argument to this option when required.

Here after, the available options with their full names are enumerated and detailed.

`influx_si` command line options

--version show program's version number and exit

-h, --help show the help message and exit

--noopt no optimization, just use free fluxes as is (after a projection on feasibility domain), to calculate dependent fluxes, cumomers, stats and so on

--noscale no scaling factors to optimize => all scaling factors are assumed to be 1

This option can be useful if your measurements are already scaled to sum up to 1 which is often the case of MS data. Then, user saves

some free parameters corresponding to scaling factors. This option can become mandatory if user wants to prevent scaling factors to be adjusted by optimization process.

--meth=METH method for optimization, one of `nlsic|BFGS|Nelder-Mead|pso`. Default: `nlsic`. Multiple occurrences of this option can appear on command line. In this case, specified minimization methods are applied successively, e.g. `--meth pso --meth nlsic` means that `pso` will be used first, then `nlsic` will take over from the point where `pso` ends. In case of multiple methods, it is recommended starting with non-gradient methods like `pso` or `Nelder-Mead` and make them follow by gradient based methods like `nlsic` or `BFGS`. If `pso` or `Nelder-Mead` are indeed used as the first method, it is not recommended combining them with `--zc` option.

--fullsys calculate all cumomer set (not just the reduced one necessary to simulate measurements)

This option influences only post-optimization treatment. The fitting itself is still done with the reduced cumomer set or EMU variables if requested so. See the original paper on `influx_s` for more information on the reduced cumomer set.

--emu simulate labeling in EMU approach

This option should not produce a different result in parameter fitting. It is implemented and provided in a hope that on some network the results can be obtained in a shorter time

--irand ignore initial approximation for free parameters (free fluxes and specie concentrations) from the FTBL file or from a dedicated file (cf –fseries and –iseries option) and use random values drawn uniformly from [0,1]

It is recommended to use this option in conjunction with "–zc 0" option.

--sens=SENS sensitivity method: SENS can be 'mc[=N]', mc stands for Monte-Carlo. N is the number of Monte-Carlo simulations. Default for N: 10

The sensitivity information (i.e., the influence of the noise in the data on the estimated parameter variation) based on linearized statistics is always provided. So the user has to use this option only if he wants to compare this linearized information to the Monte-Carlo simulations. Note that the default value 10 for the number of simulations is far from to be sufficient to get reliable statistical estimations. This default option allows only to quickly check that this option is working as expected.

--cupx=CUPX upper limit for reverse fluxes. Must be in interval [0, 1]. Default: 0.999

--cupn=CUPN upper limit for net fluxes. Default: 1.e3

--cupp=CUPP upper limit for specie pool. Default: 1.e5

--clownr=CLOWNR lower limit for not reversible free and dependent fluxes. Zero value (default) means no lower limit

A byproduct of this option is that it can drastically reduce cumomer system sizes. As it ensures that non-reversible fluxes cannot change the sign, revers fluxes can be eliminated from pathways leading to observable cumomers.

--cinout=CINOUT lower limit for input/output free and dependent fluxes. Must be non-negative. Default: 0

--clowp=CLOWP lower limit for free specie pools. Must be positive. Default 1.e-8

--np=NP When integer >= 1, it is a number of parallel threads (on Unix) or subprocesses (on Windows) used in Monte-Carlo (M-C) simulations or for multiple FTBL inputs. When NP is a float number between 0 and 1, it gives a fraction of available cores (rounded to closest integer) to be used. Without this option or for NP=0, all available cores in a given node are used for M-C simulations.

--ln Least norm solution is used for increments during the non-linear iterations when Jacobian is rank deficient

Jacobian can become rank deficient if provided data are not sufficient to resolve all free fluxes. It can be useful to determine fluxes that can still be resolved by the available measurements. If the Jacobian does not become rank deficient, this option has no influence on the found solution, neither on the optimization process. But if the Jacobian does become rank deficient, a warning message is printed in the error file even if the optimization process could go to the end.

Note: Use this option with caution, in particular, when used in conjunction with Monte-Carlo simulations. As undetermined fluxes will be given some particular value, this value can be more or less stable from one Monte-Carlo simulation to another. This can create an illusion that a flux is well determined. See the linearized statistics in the result file to decide which fluxes are badly resolved.

A correct way to deal with badly defined metabolic network is to provide additional data that can help to resolve all the fluxes and/or to optimize input label, not just put `--ln` option and cross the fingers.

> **Warning:** In this option, the notion of "least norm" is applied to *increments* during the optimization, not to the final solution. So undetermined fluxes could vary from one run to another if the optimization process is started from different points, while well determined fluxes should keep stable values.

--sln Least norm of the solution of linearized problem (and not just of increments) is used when Jacobian is rank deficient

--tikhreg Approximate least norm solution is used for increments during the non-linear iterations when Jacobian is rank deficient

To obtain an approximate solution, a Tikhonov regularization is used when solving an LSI problem. Only one of the options `--ln` and `--tikhreg` can be activated in a given run.

--lim The same as –ln but with a function limSolve::lsei()

--zc=ZC Apply zero crossing strategy with non-negative threshold for net fluxes

This option can accelerate convergence in situations when a net flux has to change its sign during the optimization iterations. Once such flux is identified, it is better to write the corresponding reaction in opposite sens in the FTBL file or to give a starting value with a correct sign to avoid such zero crossing situation.

--ffguess Don't use free/dependent flux definitions from FTBL file(s). Make an automatic guess.

The fact that free fluxes are chosen automatically does not allow specifying a starting point for optimization iterations so a random starting point is used (drawn uniformly in [0; 1] interval). An option `--seed` can be useful to make the results reproducible.

--fseries=FSERIES File name with free parameter values for multiple starting points. Default: '' (empty, i.e. only one starting point from the FTBL file is used)

The file must be formatted as plain text file with tab separator. There must be as many columns as starting points and at least as many rows as free parameters assigned in this file. A subset of free parameters can be used in this file. In this case, the rest of parameters take their unique starting values from the FTBL file. The first column must contain the names of free parameters used in this file. If there are extra rows whose names are not in the set of free parameter names, they are simply ignored. The first row must contain the names of starting points. These names can be just numbers from 1 to the number of starting points.

--iseries=ISERIES Indexes of starting points to use. Format: '1:10' – use only first ten starting points; '1,3' – use the first and third starting points; '1:10,15,91:100' – a mix of both formats is allowed. Default '' (empty, i.e. all provided starting points are used)

When used with conjunction with `--fseries`, this option indicates the starting points to use from FSERIES file. But this option can also be used in conjunction with `--irand` to generate a required number of random starting points, e.g., `influx_s.py --irand --iseries 1:10 mynetwork` will generate and use 10 random starting points.

For both `--fseries` and `--iseries`, one result file is generated per starting point, e.g., `mynetwork_res.V1.kvh`, `mynetwork_res.V2.kvh` and so on. If starting points comes from a `--fseries` then the suffixes `V1`, `V2`, ... are replaced by the column names from this file. In addition, a file `mynetwork.pres.csv` resuming all estimated parameters and final cost values is written.

--seed=SEED Integer (preferably a prime integer) used for reproducible random number generating. It makes reproducible random starting points (–irand) but also Monte-Carlo simulations for sensitivity analysis. Default: none, i.e., current system value is used, so random drawing will be varying at each run.

--excl_outliers This option takes an optional argument, a p-value between 0 and 1 which is used to filter out measurement outliers. The filtering is based on Z statistics calculated on reduced residual distribution. Default: 0.01.

Excluded outliers (if any) and their residual values are reported in the `mytework.log` file. Non-available (`NA`) measurements are considered as outliers for any p-value. An optional p-value used here does not give a proportion of residuals that will be excluded from the optimization process, but rather a degree of being a valuable measurement. So, closer to zero is the p-value, the fewer data are filtered out. If in contrary, you want to filter out more outliers than with the default p-value, use a value grater than the default value of 0.01, e.g.:

```
influx_s.py --excl_outliers 0.02 mynetwork.ftbl
```

Note: Don't use an equal sign "=" to give a p-value to this option. Here, only a white space can be used as a separator (as in the example above).

--nocalc generate an R code but not execute it.

This option can be useful for parallel execution of the generated R files via `source()` function in cluster environment

--addnoise Add centered gaussian noise to simulated measurements written to _res.kvh file. SD of this noise is taken from FTBL file

This option can be helpful for generating synthetic FTBL files with realistic simulated measurements (cf. *How to make FTBL file with synthetic data?*).

--mtf MTF option passed to txt2ftbl. See help there.

--prefix PREFIX option passed to txt2ftbl. See help there.

--eprl EPRL option passed to txt2ftbl. See help there.

--force FORCE option passed to txt2ftbl. See help there.

--copy_doc copy documentation directory in the current directory and exit. If ./doc exists, its content is silently overwritten.

--copy_test copy test directory in the current directory and exit. If ./test exists, its content is silently owerriten.

--install_rdep install R dependencies and exit.

starting from v5.3, this installation is made in interactive mode. I.e. if the default installation directory (the first one from a list returned by R's `.libPaths()`) is not writable by the user then `influx_si` will try to install the needed packages in the directory defined in R session variable `R_LIBS_USER`. If this last does not exist, the user is asked for a permission to create it. This behavior is the default one of R's `install.packages()` which is used here.

--TIMEIT developer option

Some portions of code are timed, and the results is printed in the log-file. A curious user can use this option without any harm.

--prof developer option

This option provides much more detailed profiling of the execution than `--TIMEIT` option. Only developers can be interested in using such information.

All command line options can also be provided in a .opt file. A user can put them in the field `commandArgs`, e.g.

```
Name            Value
commandArgs     --meth BFGS --sens mc=100 --np 1
```

If an option is provided both on the command line and in the .opt file, it is the command line that has the priority. In such a way, a user is given an opportunity to overwrite any option at the run time. Nevertheless, there is no way to cancel a flag option (an option without argument) on a command line if it is already set in the .opt file. For example, if `--fullsys` flag is set in the .opt file, the full system information will be produced whatever command line options are.

Parallel experiments

Staring from v4.0, `influx_si` offers possibility to treat labeling data from parallel experiments. Parallel experiments for stationary labeling were described in the literature (e.g. cf. "Parallel labeling experiments and metabolic flux analysis: Past, present and future methodologies.", Crown SB, Antoniewicz MR., *Metab Eng.* 2013 Mar;16:21-32. doi: 10.1016/j.ymben.2012.11.010). But for instationary labeling, at the best of our knowledge, `influx_si` is the first software offering parallel experiments treatment.

The main interest of parallel experiments is increased precision of flux estimations. This comes at a price of additional work for experiments and data gathering, but the result is often worth the effort. As usual, before doing a real "wet" experiment, it can be useful to run a few "dry" simulations to see if planned experiments will deliver desired precision.

To deal with parallel experiments, a user have to prepare a series of additional .miso/.linp couples, one per additional experiment. While the "main" .miso/.linp couple can be given in `--prefix` or `--mtf` options.

Each couple provides input labeling and measured labeling data corresponding to an experiment. This file architecture ensures that a network topology, flux, and specie values are common to all experiments, while entry label and measurements on labeled species are proper to each experiment.

When files are ready, you can run `influx_si` on them, e.g. in `test/prl_exp/mtf` directory run:

```
$ influx_s.py --pref e_coli_glc1-6n --eprl e_coli_glc2n,e_coli_glc3n,e_coli_glc4n,e_
↪coli_glc5n,e_coli_glc6n
```

In this example, 6 parallel experiments were used, the "main" being described in files ecoli_glc1-6n and 5 additional ones in files going from `e_coli_glc2n` to `e_coli_glc6n`. Note that we used a compact form of `--eprl` options as all .miso/.linp couples used canonical suffixes. In that way, giving only a prefix like `e_coli_glc2n` was sufficient to find the both corresponding files.

The command can be shortened even more if `prl_exp` option is used in `.opt` file. For example if we write in the main set `e_coli_glc1-6n.opt`:

```
Name    Value
prl_exp e_coli_glc2n,e_coli_glc3n,e_coli_glc4n,e_coli_glc5n,e_coli_glc6n
```

then the command to run parallel experiments becomes simply:

```
$ influx_s.py --pref e_coli_glc1-6n
```

This example set of files correspond to stationary labeling experiments described in "Complete-MFA: Complementary parallel labeling experiments technique for metabolic flux analysis", Robert W. Leighty, Maciek R. Antoniewicz, *Metabolic Engineering* 20 (2013) 49–55 (with only difference that we use simulated and noised data instead of measured ones).

We also provide an example of simulated instationary parallel experiments in the files `e_coli_GX_prl` (main files) and `e_coli_GX_X` (secondary files) corresponding to simultaneous consumption of glucose and xylose. The network for these simulations was borrowed from "13C metabolic flux analysis of microbial and mammalian systems is enhanced with GC-MS measurements of glycogen and RNA labeling", Christopher P. Long, Jennifer Au, Jacqueline E. Gonzalez, Maciek R. Antoniewicz, Metabolic Engineering 38 (2016) 65–72. The experiment consisted in dynamic labeling by uniformly labeled glucose (main experiment) and by uniformly labeled xylose (secondary one). Labeling kinetics MS data are given in `e_coli_GX_MS.miso` and `e_coli_GX_X_MS.miso` files respectively. To play with this example (still in the same directory), you can run:

```
$ influx_i.py e_coli_GX_prl
```

Note that set of measured specie fragments as well as sampling time points for instationary labeling are not necessary the same for instationary experiments. They do can differ. That's why a `.opt` can be necessary to add to `.miso/.linp` couple to form a complete parallel experiment.

It should be made a clear distinction between parallel experiments described in this section and independent experiments calculated in parallel. The main difference is that parallel experiments of this section share all the same flux map (only labeling pattern and data can differ) while independent experiments can have each its own flux map, be they calculated in parallel or sequentially.

Options in .opt file

In this section, we describe different options that can appear in .opt file

Optimization options

These options can help to tune the convergence process of the NLSIC (or any other chosen algorithm). These options are prefixed with `optctrl` which is followed by a particular optimization method name and ended by an option name. For example, `optctrl:nlsic:errx` corresponds to the stopping criterion. A corresponding `.opt` portion could look like

```
Name                    Value
optctrl:nlsic:errx      1.e-3
```

NLSIC parameters

All possible options and their default values for NLSIC algorithm follow:

errx=1.e-5 stopping criterion. When the L2 norm of the increment vector of free parameters is below this value, the iterations are stopped.

maxit=50 maximal number for non-linear iterations.

btstart=1. backtracking starting coefficient

btfrac=0.25 backtracking fraction parameter. It corresponds to the alpha parameter in the paper on `influx_s`

btdesc=0.1 backtracking descending parameter. It corresponds to the beta parameter in the paper on `influx_s`

btmaxit=15 maximal number of backtracking iterations

trace=1 report (=1) or not (=0) minimal convergence information

rcond=1.e10 condition number over which a matrix is considered as rank deficient

ci=list(p=0.95, report=F) confidence interval reporting. This option is own to `nlsic()` function. It has no impact on the reporting of linear stats information in the result kvh file after the post-optimization treatment. This latter is always done.

history=FALSE return or not (default) the matrices with optimization steps and residual vectors during optimization. These matrices can then be found as part of `optimization process information/history` field in `mynetwork_res.kvh` file. Use it with caution, big size matrices can be generated requiring much of memory and disk space.

adaptbt=TRUE use (default) or not an adaptive backtracking algorithm.

monotone=FALSE should or not the cost decrease be monotone. If TRUE, then at first non decrease of the cost, the iterations are stopped with a warning message.

PSO parameters

Particle Swarm Optimization (PSO) is a stochastic optimization method. It can help to avoid local minimums but its convergence is very slow. That's why its usage can be particularly useful if combined with a deterministic algorithm like NLSIC. We have implemented PSO method based on the code from CRAN package pso v1.0.3 published in 2012 by Claus Bendtsen (papyrus.bendtsen at gmail.com). The original algorithm was written for box constrained problems. While `influx_si` requires a usage of general linear constraints. So we modified the algorithms accordingly. Its parameters with their default values used in `influx_si` are following:

trace=0 an integer controlling the trace printing. A zero value means no printing

fnscale=1 scale factor for minimized function. It is useless in `influx_si` context.

maxit=100 maximal iteration number to not overcome

maxf=Inf maximal number of a cost function evaluation

abstol=-Inf stopping criterion by absolute tolerance during approximating the searched minimum. This parameter can only be useful if the searched minimal value is known in advance. It is not the case of `influx_si`

reltol=0 stopping criterion by relative change in the found minimal value

REPORT = 10 if tracing is enabled, this parameters gives the number of iterations passed between two successive reports

s=NA, swarm size. If NA, it is automatically determined.

k=3, p=NA, w=1/(2*log(2)), c.p=.5+log(2), c.g=.5+log(2) are parameters governing PSO minimization paths. For their significance see the original pso documentation

d=NA domain diameter

v.max=NA maximum allowed velocity

rand.order=TRUE proceed swarm particles in random order or not

max.restart=Inf maximal allowed restarts

maxit.stagnate=Inf maximal successive iterations allowed without a detected decrease in optimization function.

trace.stats=FALSE return or not detailed statistics about the convergence process (not used in `influx_si`)

type="SPSO2011", which PSO strategy to use. Available options are "SPSO2011" and "SPSO2007". More about this in the original documentation.

tolineq=1.e-10 tolerance for violating of linear constraints that can happen mainly due to rounding errors.

Other optimization methods

Names and default values for BFGS and Nelder-Mead algorithms can be found in the R help on `optim()` function.

Growth flux option

If present, this option makes `influx_si` take into account growth fluxes $-\mu M$ in the flux balance, where μ is a growth rate and M is a concentration of an internal specie M by a unit of biomass. Only species for which this concentration is provided in a .tvar file, contribute to flux balance with a flux $-\mu M$. This flux can be varying or constant during optimization process depending on whether the specie M is part of free parameters to fit or not. Usually, taking into account of this kind of flux does not influence very much on the estimated flux values. So, this option is provided to allow a user to be sure that it is true in his own case.

The option is activated by a field `include_growth_flux`:

```
Name                    Value
include_growth_flux     1
```

Value 0 cancels the contribution of the growth fluxes to the general flux balance.

Another necessary option is `mu` giving the value of μ:

```
Name    Value
mu      0.12
```

Please note that the specie concentrations by a unit of biomass are reported in a file .tvar as:

```
Name    Kind    Type    Value
Fum     METAB   C       2.47158569399681
Suc     METAB   F       15.8893144279264
Mal     METAB   F       6.47828321758155
...     ...
```

Specie names used in this section must be identical to those used in the .netw file and others. "F" is used as an indicator of a varying specie pool. Such varying species are part of fitted parameters. Column "Value" is used as starting value in the optimization process.

One of valuable originality of `influx_s`, it is a possibility to couple fluxomics and metabolomics in stationary experiments. It can be done because specie pools can influence labeling in two ways:

- through specie pooling (due to compartmentalization and/or co-elution during chromatography)
- through growth fluxes.

This last influence is often of low intensity compared to specie transformation fluxes. In literature, it is typically neglected.

Another possibility that was added `influx_si` is to provide measured specie concentrations in `.mmet` file:

```
Specie                  Value                              SD
Suc                     15.8893144279264*1.e-3/10.7        1.e-2
Mal                     6.47828321758155*1.e-3/10.7        1.e-2
Rub5P+Rib5P+Xul5P       1.66034545348219*1.e-3/10.7        1.e-2
```

Like for other measurements, the user has to provide a name, a value, and a standard deviation for each entry. Species listed in this section must be defined in the .netw file and must have type "F" in the `.tvar`. Numerical values can be simple arithmetic expressions (as in the example above) which are evaluated during file parsing.

When a specie name is given as a sum of species (e.g. `Rub5P+Rib5P+Xul5P`) it is interpreted as a list of species to be pooled. It is done proportionally to their concentrations. No numerical factor can appear in this sum. At least one of the species from the list must be free (i.e. to have "F" type in the `.tvar` file). Otherwise, all species from the list would be considered as having a fixed concentration and providing a measurement for such species would be meaningless.

Note: Species having "F" (as "Free") in column "Type" in a .tvar file are treated as fittable parameters. We recall that species in this file are identified as having "METAB" in column "Kind".

An example of an MTF files having specie sections and involving growth fluxes can be found in `test/mtf/e_coli_growth.*`.

Post treatment option

User can specify a name of one or several R scripts that will be automatically executed after non aborted `influx_si` run. This option can be useful, for example, for plain saving of calculation environment in a file for later exploring in an interactive R session or for plotting results in a pdf file and so on. A very basic example of such a script is provided in the file `R/save_all.R` and its use can be found in the options of `test/e_coli.opt` file.

To activate this option, the script names must be provided in the `.opt` file, in the field `posttreat_R` and separated by `'; '`, e.g.:

```
Name            Value
posttreat_R     save_all.R; plot_smeas.R
```

The script name is interpreted as a relative path to the directory where the original MTF files are located. If the file is not found there, it is searched for in `influx_si/R`. After execution of `save_all.R`, a file `e_coli.RData` is created. This particular example can be used to restore a calculation R environment by launching R and executing:

```
> load("e_coli.RData")
```

After that, all variables defined in influx_si at the end of the calculations will be available in the current interactive session. To be able to launch custom calculations on these variables, the user has to do some preliminary actions. An example of such actions can be found in a file `preamble.R` which can be adapted for user's case.

To write his own scripts for post treatments or explore the calculated values in an interactive session, a user have to know some basics about existent variables where all the calculation results and auxiliary information are stored. Here are few of them:

dirw is a working directory (where the original FTBL file is)

dirx is an executable directory (where influx_s.py is)

baseshort is a short name of the input FTBL file (without the suffix `.ftbl` neither the directory part of the path)

param is the vector of the estimated parameters composed of free fluxes, scaling parameters (if any) and specie concentrations (if any)

labargs is an environment with all necessary data for label simulation (e.g. `v=lab_sim(param, cjac=FALSE, labargs)`) or residual calculation (e.g. `rres=lab_resid(param, cjac=TRUE, labargs)`)

jx_f is a environment regrouping calculated quantities. Here are some of its fields:

`lf` a list with different fluxes:

`fallnx` a vector of all net and exchange fluxes (here, exchange fluxes are mapped on [0; 1[interval)

`fwrv` a vector of forward and reverse fluxes (reverse fluxes are "as is", i.e. not mapped)

`xsim` is an internal state label vector

`simlab`, `simfmn` and `simpool` are vectors of simulated measurements for label, net flux and specie pools respectively (fitting at the best of influx_s' capacity the provided measurements)

`res` is the reduced residual vector, i.e. (simulated-measured)/SD

`ures` is the unreduced residual vector, i.e. (simulated-measured)

`jacobian` as its names indicates, is the Jacobian matrix (d res/d param)

`udr_dp` is the Jacobian matrix for the unreduced residual vector (d ures/d param)

measurements is a list regrouping various measurements and their SD

nb_f is a list of various counts, like number of fluxes, parameters to fit, system sizes and so on

nm_list is a list of names for various vectors like fluxes, species, label vectors, measurements, inequalities and so on

ui, ci are inequality matrix and right-hand side respectively

A full list of all available variable and functions can be obtained in an R session by executing:

```
> ls()
```

This list of more than 400 items is too long to be fully described here. We hope that the few items succinctly described in this section will be sufficient for basic custom treatments.

An inspiration for your own custom treatments and/or plotting can be found in files `plot_ilab.R` and `plot_smeas.R` that plot instationary and stationary data respectively in pdf files.

Exclusive `influx_i` options

There is only one exclusive option that can be given on a command line:

--time_order=TIME_ORDER Time order for ODE solving (1 (default), 2 or 1,2). Order 2 is more precise but more time-consuming. The value '1,2' makes to start solving the ODE with the first order scheme then continues with the order 2.

The scheme order can be important for the precision of flux and concentration estimations. The impact is not direct, but can be very significant. Please note that it can happen that order 1 fits the data with lower cost value function, but it does not mean that the fluxes/concentrations are better estimated.

Other options can occur as fields in a `.opt` file.

`nsubdiv_dt` integer number of sub-intervals by which every time interval is divided to increase the precision of time resolution.

It can happen that the value 1 (default) is sufficient for a satisfactory flux/concentration estimation. User can gradually increase this value (2, 3, . . .) in successive `influx_i` runs to be sure that better

time resolution does not impact parameter estimation. This property is called *grid convergence*. A grid convergence is necessary to overcome the result dependency on the choice of a numerical discretization scheme. A grid convergence can be considered as achieved when changes in estimated parameters provoked by a grid refinement are significantly lower than estimated confidence intervals for these parameters.

dt a real positive number, defines a time step in a regular grid in absence of values in "Time" column in `.miso` file. If a "Time" values are well present for label kinetics, then this parameter has no effect.

A regular time grid for label simulations can be useful on preliminary stage when user only elaborates MTF files and wants to see if label simulation are plausible. It can also help to produce simulated measurements (which can be extracted from the `_res.kvh` file) for further numerical experiments like studying convergence speed, parameter identifiability, noise impact and so on.

tmax a real positive number, defines the end of a regular time grid if "Time" is empty or absent in `.miso`. Parameters `dt` and `tmax` must be defined in such a way that there will be at least 2 time points greater than 0 in the time grid.

If "Time" values are well present in `.miso` then this parameter can be used to limit the time grid on which the simulations are done. If the value in `tmax` is greater than the maximal time value defined in `.miso` file then this parameter has no effect.

Note: It is very important that the values for time, flux, and specie concentrations be expressed in concordant units. It would be meaningless to give time in minutes, fluxes in mM/h/g and concentrations in mM. This will lead to wrong results.

For example, if the time is expressed in seconds and concentrations in mM/g then fluxes must be expressed in mM/s/g.

---

funlabR since v5.4, `influx_i` is able to simulate label propagation in a metabolically stationary network from a label input varying in time. User can supply R expressions which will calculate fractions of different input label components as functions of time `t`. Those expressions can be provided in the `Value` column of `.linp` file but they can need some helper functions. Few of them are defined in a file `funlab.R` included in `influx_si` but user can need more of them. Thanks to this field, he can define them in a custom R file, who's name can be provided here. There can be given only one file. However, if user-defined functions are spread over several files, they can be included via `source()` function called from this one. For this purpose, a predefined variable `dirw` pointing to the current working directory can be useful. It is worth mentioning that the file defined in this field will be executed in a particular environment so that variables created during its execution won't affect `influx_si`'s ones. The path of the file provided in this field is relative to the .netw's one. Example:

```
Name            Value
funlabR         e_coli_iv_funlab.R  # the file 'e_coli_iv_funlab.R' is␣
↪in the same directory that 'e_coli_iv.*' MTF set
```

Functions that are available in `funlab.R` are following:

ppulses(tp, Tint, Hint=rep_len(c(1., 0.), length(Tint)))
computes a signal in the form of periodic rectangular pulses in time points `tp`. Each period is composed of one or several intervals whose length in time is given in numeric vector `Tint` and heights of signals are given in optional numeric vector `Hint`. By default, `Hint` is a sequence of 1's and 0's. The very first period starts at t=0. Returns a numeric vector of the same length as `tp`.

linterp(tp, knots, v) computes a signal in the form of continuous linear piecewise functions in time points `tp`. The limits of linear intervals are defined in numeric

vector `knots` and values at limits must be given in numeric vector `v`. All `tp` values must lie between `min(knots)` and `max(knots)`. Returns a numeric vector of the same length as `tp`.

`steplinpath(tp, nu)` computes labeling in a linear pathway of non-reversible reactions under step labeling with fully unlabeled initial state, i.e. starting from 0. The signal is calculated in time points `tp`. The pathway is defined by numeric vector `nu` which represents turn-over rates (i.e. a ratio Flux/Specie_Concentration). All values in `nu` must be pairwise different. Returns a numeric matrix of size m x n, where number of rows m=length(nu) and number of columns n=length(tp). So, for example, if a signal only of the third specie is required, the function can be called as `steplinpath(tp, nu)[3,]`

`steplinpath2(tp, nu, init=double(length(nu)), height=1.)` The same as `steplinpath()` above but initial state can be different from 0. It can be defined in numeric vector `init` of the same length as `nu`. The label amplitude can be given in a scalar `height`.

`ppulseslinpath(tp, nu, Tint, Hint=rep_len(c(1., 0.), length(Tint)), init=double` computes labeling of linear non-reversible pathway under input composed of periodic rectangular pulses. Labeling for all species is calculated in time points `tp`. Each period is composed of one or several intervals whose length in time is given in numeric vector `Tint` and heights of signals are given in optional numeric vector `Hint`. By default, `Hint` is a sequence of 1's and 0's. Initial label levels can be defined in numeric vector `init`. Returns a numeric matrix of size m x n, where number of rows m=length(nu) and number of columns n=length(tp).

`Value` column in `.linp` in this field, user can provide R expression calculating fractions of input label as function of time. While for `influx_s`, it can only be a constant or python expression evaluating to a scalar at compilation time. Such R expressions can refer to a scalar variable `t` representing the current time point and should return a scalar value representing a label level between 0 and 1. The sum of all label levels relative to a given specie must be 1 if a full set of isotopomer is provided by user. If one or many isotopomers are lacking, usual conventions apply for completion to 1 (cf. *MTF format*).

```
# .linp example
Specie  Isotopomer      Value
Gluc_U  111111          ppulses(t, c(T1,T2))  # periodic pulses composed␣
↪of intervals of length T1 ("labeled") and T2 ("unlabeled")
Gluc_1  100000          ppulses(t, c(T1,T2))

# .opt example
Name    Value
funlabR e_coli_iv_funlab.R  // in this R file variables T1 and T2 are␣
↪defined
```

Input isotopomers that are absent in such `funlab` fields are supposed to be 0 all the time (except for above-mentioned conventions).

If a label level cannot be calculated in one arithmetic operation, several R statements can be placed between curled braces `{}` separated by semicolon `;`. The last operation must be the searched result. In the example above, we could exclude usage of helper file `e_coli_iv_funlab.R` by defining `T1` and `T2` directly in the expressions:

```
# .linp example
Specie  Isotopomer      Value
Gluc_U  111111          {T1=2; T2=2; ppulses(t, c(T1,T2))}
Gluc_1  100000          {T1=2; T2=2; ppulses(t, c(T1,T2))}
```

Result file fields

Generally speaking, the names of the fields in the result KVH file are chosen to be self-explanatory. So there is no so much to say about them. Here, we provide only some key fields and name conventions used in the result file.

At the beginning of the `mynetwork_res.kvh` file, some system information is provided. Here, "system" should be taken in two sens: informatics and biological. The information is reported in the fields `influx` and `system sizes`. These fields are followed by `starting point` information regrouping `starting free parameters`, `starting cost value`, `flux system (Afl)` and `flux system (bfl)`. Name conventions used in these and other fields are the following:

net and exchange fluxes are prefixed by `n.` or `x.` respectively

free, dependent, constrained and variable growth fluxes are prefixed by `f.`, `d.`, `c.` and `g.` respectively. So, a complete flux name could look like `f.n.zwf` which means *free net ZWF flux*. Growth fluxes which depend on constant specie concentrations can be found in constrained fluxes. Constant or variable growth fluxes are postfixed with `_gr` (as *growth*) string. For example, a flux `g.n.Cit_gr` corresponds to a net growth flux of Citrate specie. The growth fluxes are all set as non-reversible, so all exchange fluxes like `g.x.M_gr` or `c.x.M_gr` are set to 0.

scaling factors names are formed according to a pattern similar to `label;Ala;1` which corresponds to the first group of measurements on Alanine molecule in labeling experiments. Other possible types of experiments are `peak` and `mass`.

MID vector names are looking like `METAB+N` where `METAB` is specie name and `N` goes from 0 to the number of carbon atoms in the considered molecule.

cumomer names follow classical convention `METAB#pattern_of_x_and_1`, e.g. `Ala#x1x`

forward and reverse fluxes are prefixed by `fwd.` and `rev.` respectively, e.g. `fwd.zwf` or `rev.zwf`

measurement names have several fields separated by a colon `:`. For example, `l:Asp:#xx1x:694` deciphers like:

- `l` stands for *labeling* experiment (others possibilities are `p` for *peak*, `m` for *mass* and `pm` for *specie pool*)
- `Asp` is a specie name
- `#xx1x` is a measurement identification
- `694` is a line number in the FTBL file corresponding to this measurement.

The field `optimization process information` is the key field presenting the results of an optimization process. The fitted parameters are in the subfield `par`. Other subfields provide some additional information.

The final cost value is in the field `final cost`.

The values of vectors derived from free fluxes like dependent fluxes, cumomers, MID and so on are in the corresponding fields whose names can be easily recognized.

Linear stats and Monte-Carlo statistics are presented in their respective fields. The latter field is present only if explicitly requested by user with `--sens mc=MC` option. In this kvh section, a term `rsd` means "relative standard deviation" (in literature, it is often encountered a synonym CV as Coefficient of Variation). It is calculated as SD/Mean and if expressed in percentage then the formula becomes 100%*SD/Mean.

The field `jacobian dr_dp (without 1/sd_exp)` report a Jacobian matrix which is defined as a matrix of partial derivatives $\partial r/\partial p$ where r is residual vector (Simulated–Measured) and p is a free parameter vector including free fluxes, scaling factors (if any) and free specie pools (if any). Note that in this definition, the residual vector is not yet scaled by standard deviation of measurements. Sometimes, Jacobian is called *sensitivity matrix*, in which case a special care should be brought to the sens of derivation. Often, by sensitivity matrix, we intend a matrix expressing how estimated fluxes are sensitive to variations in the measurement data. Such definition corresponds to generalized

inverse of Jacobian and it is reported in the field `generalized inverse of jacobian dr_dp (without 1/sd_exp)`

Network values for Cytoscape

Several network values formatted for cytoscape are written by `influx_si` to their respective files. It can facilitate their visualizing and presentation in graphical mode. All these values can be mapped on various graphical attributes like edge width, node size or color scale of them. All these files are written at the end of calculations, so if an error has interrupted this process, no such file will be produced. Take care to don't use an outdated copy of these files.

A file named `edge.netflux.mynetwork.attrs` can help to map net flux values on edges of a studied network. A file `edge.xchflux.mynetwork.attrs` do the same with exchange fluxes. And finally, `node.log2pool.mynetwork.attrs` provides logarithm (base 2) of pool concentrations. They can be mapped on some graphical attribute of network nodes.

See *Additional tools* section, *ftbl2xgmml: cytoscape view* paragraph to know how to produce files importable in Cytoscape from a given FTBL file. User's manual of Cytoscape has necessary information about using visual mapper for teaching how some values like net flux values can be mapped on graphical elements like edge width and so on.

Warning and error messages

The warning and error messages are logged in the `.err` suffixed file. For example, after running:

```
$ influx_s --prefix mynetwok
```

the warnings and errors will be written in the `mynetwork.err` file. This kind of messages are important for user not only to be aware that during calculations something went wrong but also to understand what exactly went wrong and to have an insight on how to fix it.

Problems can appear in all stages of a software run:

- parsing MTF/FTBL files
- R code writing
- R code execution
  - vector-matrix initialization
  - optimization
  - post-optimization treatment

Most of the error messages are automatically generated by underlying languages Python and R. These messages can appear somewhat cryptic for a user unfamiliar with these languages. But the most critical error messages are edited to be as explicit as possible. For example, a message telling that free fluxes are badly chosen could look like:

```
Error : Flux matrix is not square or singular: (56eq x 57unk)
You have to change your choice of free fluxes in the 'mynetwork.ftbl' file.
Candidate(s) for free flux(es):
d.n.Xylupt_U
```

a message about badly structurally defined network could be similar to

```
Error : Provided measurements (isotopomers and fluxes) are not
        sufficient to resolve all free fluxes.
Unsolvable fluxes may be:
        f.x.tk2, f.n.Xylupt_1, f.x.maldh, f.x.pfk, f.x.ta, f.x.tk1
Jacobian dr_dff is dumped in dbg_dr_dff_singular.txt
```

a message about singular cumomer balance matrix could resemble to

```
lab_sim: Cumomer matrix is singular. Try '--clownr N' or/and '--zc N' options with␣
↪small N, say 1.e-3 or constrain some of the fluxes listed below to be non zero Zero␣
↪rows in cumomer matrix A at weight 1:
cit_c:16
ac_c:2
...
Zero fluxes are:
fwd.ACITL
...
```

Note: In this error message, we report cumomers whose balance gave a zero row in the cumomer matrix (here `cit_c:<N>` cumomers, where <N> is an integer, its binary mask indicates the "1"s in the cumomer definition) as well as a list of fluxes having 0 value. This information could help a user to get insight about a flux whose zero value led to a singular matrix. A workaround for such situation could be setting in the `.cnstr` file an inequality constraining a faulty flux to keep a small non zero value. A more radical workaround could be restricting some flux classes (input-output fluxes with the option `--cinout=CINOUT` or even all non-reversible ones with the option `--clownr=CLOWNR`) to stay out of 0, e.g.:

```
$ influx_s.py --clownr 0.0001 --prefix mynetwork
```

Adding such inequalities does not guaranty that cumomer matrix will become invertible, but often it does help. It's up to the user to check that an addition of such inequalities does not contradict biological sens of his network.

---

a message about badly statistically defined network could appear like

```
Inverse of covariance matrix is numerically singular.
Statistically undefined parameter(s) seems to be:
f.x.pyk
For more complete list, see sd columns in '/linear stats'
in the result file.
```

and so on.

A user should examine carefully any warning/error message and start to fix the problems by the first one in the list (if there are many) and not by the easiest or the most obvious to resolve. After fixing the first problem, rerun `influx_si` to see if other problems are still here. Sometimes, an issue can induce several others. So, fixing the first issue could eliminate some others. Repeat this process, till all the troubles are eliminated.

Problematic cases

Obviously, everyone would like be able just run a flux estimation software and simply get results, but unfortunately it does not work in this way every time. In this section, we review some problematic cases which can be encountered in practice.

Structurally non-identifiable fluxes

It can happen that collected data are not sufficient to resolve some fluxes in your network. Due to the non-linear nature of the issue, this situation can appear for some set of free flux values and disappear for others, or be persistent for any free flux values. An error is reported to signal such situation, e.g.

```
lsi: Rank deficient matrix in least squares
1 unsolvable variable(s):
f.n.PPDK        7
```

and execution is stopped.

Various options are then available for a user facing such situation.

1. Collect more data to resolve lacking fluxes. As a rule of thumb, data must be collected on species which are the nodes of convergence of badly defined fluxes or on species situated downhill of convergence point and preserving labeling pattern. The nature of collected data can also be important. Examples can be constructed where mass data are not sufficient to determine a flux but RMN data can do the job.

   Before using real data collection, you can make a "dry run" with `--noopt` option and with fictitious or even NA values for intended to collect Isospecie in the .miso file. Thus, we can see if, with these new data, the network becomes well resolved. How? If the error message disappear and SD values in the section `linear stats` are not very high then chances are that additionally collected data can help to resolve the fluxes.

2. Optimize input label. It can happen that you do collect data on a specie situated in convergence point for undefined fluxes, but incoming fluxes are bringing the same labeling pattern which prevents flux(es) to be resolved. May be changing substrate label can help in this situation. For label optimization you can use a software called IsoDesign, distributed under OpenSource licence and available here http:://metatoul.insa-toulouse.fr/metasys/software/isodes/ (may be you have received `influx_si` as part of IsoDesign package, in which case you have it already).

   Naturally, this label optimization should be done before doing actual experiments. See IsoDesing tutorial for more details on how to prepare and make such optimization.

   If you don't want or don't have a possibility to use a software for label optimization or you think to have an insight on what should be changed in substrate labeling to better define the fluxes, you can still make a try with `influx_s.py --noopt --prefix mynetwork --mtf new_label.linp` to see if a new labeling will do the job (here `new_label.linp` is an example name for a `.linp` file set that you will prepare with new entries. It is important that `--mtf new_label.linp` comes after `--prefix mynetwork` to take precedence over the old one `mynetwork.linp`)

3. Use `--ln` option. It won't make your fluxes well-defined, it will just continue calculation trying to resolve what can be solved and assigning some particular values (issued from so-called *least norm* solution for rank deficient matrices) to undefined fluxes. You will still have a warning similar to:

   ```
   lsi_ln: Rank deficient matrix in least squares
   1 free variable(s):
   f.n.PPDK        7
   Least L2-norm solution is provided.
   ```

   informing you that some flux(es) in the network is(are) still undefined. This option can be helpful if undefined fluxes are without particular interest for the biological question in hand and their actual values can be safely ignored.

4. You can give an arbitrary fixed value to an undefined flux by declaring it as constrained in the `.tvar` file (letter `C` in the column `Type` followed by some value in `Value` column).

Badly defined fluxes

Also known as *statistically undefined fluxes*, these fluxes have big or even huge SD values. The difference between these fluxes and structurally undefined fluxes is that the badly defined fluxes can become well defined if the noise is reduced or hypothetically eliminated. While the latter will still be undetermined even in the absence of the noise. Despite this difference, all options presented in the previous section are applicable here (all but `--ln` which would be without effect here).

An additional measure can be taken which consist in experimental noise reduction. Generally, it can be done by using better protocols, better instruments or simply by increasing the measurement repetition number.

Once again, a use of `--noopt` with new hoped SD values in the `.miso` file can help to see if these new measurements with better noise characteristics will resolve or not the problem.

Slow convergence

Slow optimization convergence can manifest by following warnings:

```
nlsic: Maximal non linear iteration number is achieved
```

or/and

```
nlsic: Maximal backtrack iteration number is achieved
```

Theoretically, user can increase the limit for those two numbers (`optctrl:nlsic:maxit` and `optctrl:nlsic:btmaxit` respectively in the `.opt` file) but generally it is not a good idea. It can help only in very specific situations that we cannot analyze here, as we estimate them low probable. In all cases, a slow convergence is due to high non-linearity of the solved problem. What can vary from one situation to another, it is the nature of this non-linearity. Depending on this nature, several steps can be undertaken to accelerate optimization:

1. If a non-linearity causing the slow convergence is due to the use of function absolute value $|x|$ in the calculation of forward and revers fluxes from net and exchange fluxes, then an option `--zc=ZC` (zero crossing) can be very efficient. This non-linearity can become harmful when during optimization a net flux has to change its sign, in other words, it has to cross zero.

   This option splits the convergence process in two parts. First, a minimum is searched for fluxes under additional constraints to keep the same sign during this step. Second, for fluxes that reached zero after the first step, a sign change is imposed, and a second optimization is made with these new constraints. If `--zc` option is used with an argument 0 (`--zc=0` or `--zc 0`), it can happen that fluxes reaching zero produce a singular (non invertible) cumomer balance matrix. In this case, an execution is aborted with an error starting like

   ```
   Cumomer matrix is singular. Try '--clownr N' or/and '--zc N' options␣
   ↪with small N, say 1.e-3 or constrain some of the fluxes listed␣
   ↪below to be non zero
   ...
   ```

   To avoid such situation, an argument to `--zc` must be a small positive number, say `--zc 0.001`. In this case, positive net fluxes are kept over 0.001 and negative fluxes are kept under -0.001 value. In this manner, an exact zero is avoided.

   Another way to avoid problem induced by using module function $|x|$ is to add inequality(-ies) imposing sens of reaction in `.cnstr` file e.g.

   ```
   Id      Comment Kind    Formula Operator        Value
                   NET     mae     >=              0
   ```

Naturally, in this example, you have to be sure that the reaction catalyzed by malic enzyme (here `mae`) must go in the sens written in your `.netw` file.

You can find potential candidates to impose sens of reaction by examining the flux values in `mynetwork_res.kvh` after a slow convergence and looking fluxes whose sign (positive or negative) looks suspicious to you. In our practice, we could observe a dramatic increase in convergence speed and stability just after imposing sens of reaction to a "key" reaction. Obviously, such constraint must be in accordance with biological sens of a studied network and its biological condition.

2. A high non-linearity can appear for some particular set of fluxes, especially when they take extreme values. E.g., when exchange fluxes are close to 1 or net fluxes take very high values of order 10^2 or even 10^3 (supposing that the main entry flux is normalized to 1). In such a case, user can low this limits (options `--cupx=CUPX` and `--cupn=CUPN` respectively) or try to exclude outliers (`--excl_outliers P-VALUE`) as outliers can attract the solution in weird zone of fluxes. In this latter case, the first convergence will continue to be slow and will generate corresponding warnings but the second one (after a possible automatic elimination of outliers) can converge much faster.

Convergence aborted

This situation is signaled by an error message:

```
nlsic: LSI returned not descending direction
```

This problem can occur for badly defined network, which are very sensitive to truncation errors. The effect of such errors can become comparable to the effect of the increment step during optimization. It means that we cannot decrease the norm of residual vector under the values resulting from rounding errors. If it happens for relatively small increments, then the results of convergence are still exploitable. If not, there is no so many actions that user could undertake except to make his system better defined as described in previous sections.

Note: By default, we use a very small value for increment norm as stopping criterion (10^{-5}). It can be considered as very drastic criterion and can be relaxed to 10^{-3} or 10^{-2} depending on required precision for a problem in hand (to do that, use an option `optctrl:nlsic:errx` in the `.opt` file).

Additional tools

Tools described in this section are not strictly necessary for running `influx_si` and calculating the fluxes. But in some cases, they can facilitate the task of tracking and solving potential problems in FTBL preparation and usage.

Most of the utilities produce an output written on standard output or in a file whose name is derived from the input file name. This latter situation is signaled with a phrase "The output redirection is optional" and in the usage examples the output redirection is taken in square brackets `[> output.txt]` which obviously should be omitted if an actual redirection is required. Such behavior is particularly useful for drag-and-drop usage.

ftbl2mtf: conversion of FTBL to MTF format

For old `influx_si` users having their projects in FTBL format, this utility can be an invaluable helper for making the transition to the new MTF format. Here is the help message, which can be seen with `ftbl2mtf -h`

```
usage: ftbl2mtf [-h] [-i] [-f] [-o OUT] ftbl
```

(continues on next page)

(continued from previous page)

```
Parse ftbl file from first parameter or from stdin (if input file is '-')
and write a series of mtf (multiple TSV files).
The file stem ('network' in 'network.ftbl') is used as file name basis
for produced files, e.g. 'network.miso'. Parameter --out can be used to␣
↪change it.
If out path includes non existing directories, they are automatically␣
↪created.
Caution! If an existing output file starts with a comment
"# Created by 'ftbl2mft ..."
or is empty, it is silently overwritten.
Otherwise, the writing is aborted with a warning. Other files may continue␣
↪to be created.
To force the overwriting, use '--force'.

Output files will have following extensions/meanings:

 .netw: stoichiometric equations and label transitions in the biochemical␣
↪network;
 .linp: label input;
 .miso: isotopic measurements (MS, label, peak);
 .mflux: flux measurements;
 .mmet: biochemical specie concentration measurements;
 .tvar: flux/specie types partition (free, dependent, constrained) and␣
↪starting values;
 .cnstr: constraints (equalities, inequalities for both fluxes and␣
↪concentrations);
 .opt: options.

Copyright 2022 INRAE, INSA, CNRS
Author: Serguei Sokol (sokol [at] insa-toulouse [dot] fr)

positional arguments:
  ftbl                input file to be converted to MTF

optional arguments:
  -h, --help          show this help message and exit
  -i, --inst          activate instationary mode
  -f, --force         force overwriting of result files
  -o OUT, --out OUT   path prefix for result files
```

txt2ftbl: conversion of MTF format to FTBL format

This tool is implicitly used by `influx_si` to convert MTF to FTBL format. Users desiring to play with format conversion or to produce FTBL file to be used with *Additional tools* can use it explicitly. Here is its help message:

```
usage: txt2ftbl [-h] [--mtf MTF] [--prefix PREFIX] [--eprl EPRL] [--inst] [--
↪force] [netw]

transform a series of TXT and TSV files into FTBL file.

Copyright 2021, INRAE, INSA, CNRS
Author: Serguei Sokol (sokol at insa-toulouse dot fr)
License: Gnu Public License (GPL) v2 http://www.gnu.org/licenses/gpl.html

positional arguments:
```

(continues on next page)

(continued from previous page)

```
  netw
                    If 'netw' file is not given in any option (neither --mtf␣
↪nor --prefix), it
                    can be given as the only argument NETW, e.g.
                      txt2ftbl ecoli.txt
                    or
                      txt2ftbl --mtf ms_nmr_data.miso,glucose.linp ecoli.txt
                    If 'netw' file name is given both in any option and as an␣
↪argument, it
                    is the argument value that will take precedence.

optional arguments:
  -h, --help        show this help message and exit
  --mtf MTF         MTF is a coma separated list of files with following␣
↪extensions/meanings:
                     netw: a text file with stoichiometric reactions and␣
↪label transitions (one per line)
                        Comments starts with '#' but those starting with '###
↪' introduce
                        pathways which are numbered as well as reactions in␣
↪them. Reaction
                        name can precede the reaction itself and is separated␣
↪by ":" If no
                        explicit name is given, reactions in FTBL file will␣
↪be named
                        according a pattern 'rX.Y' where X is pathway number␣
↪and Y is
                        reaction number in the pathway. But it is highly␣
↪recommended to
                        give explicit names to reactions.
                        Symbols "+", "(", ")" and ":" are not allowed in␣
↪metabolite neither reaction names
                        Example of reaction and label transition:
                           edd: Gnt6P (ABCDEF) -> Pyr (ABC) + GA3P (DEF)
                        Non reversible reactions are signaled with '->' (as␣
↪in the example above).
                        A sign '<->' can be used for reversible reactions.
                        If 'netw' name is equal to '-', then its content is␣
↪read from standard input, e.g.
                          '--mtf netw=-'
                     linp: label inputs (starting from this extensions, TSV␣
↪files are assumed)
                     miso: isotopic measurements (NMR (label, peak) and MS)
                     mflux: flux measurements
                     mmet: metabolite concentration measurements
                     tvar: type of variables (NET or XCH , free or dependent,␣
↪starting values, ...)
                     cnstr: equality and inequality constraints on fluxes and␣
↪concentrations
                     opt: options
                     ftbl: name of output FTBL file. If not given, it will be␣
↪equal to 'netw'
                       stem with '.ftbl' extension. If it is equal to '-',␣
↪then the result
                       will be written to standard output, e.g.
                         'txt2ftbl --mtf ftbl=-,ecoli.netw'
                       Intermediate directories in ftbl path are silently␣
↪created if non existent.
```

(continues on next page)

(continued from previous page)

```
                    vmtf: variable part of mtf approach.
                      If a series of FTBL files has to be generated␣
↪partially with
                      information common to all files (constant part) and␣
↪partially with
                      sections proper to each FTBL (variable part) then␣
↪files containing
                      variable sections (e.g. 'miso') can be given in a␣
↪special file
                      having an extension (or prefix, cf. hereafter) 'vmtf'.␣
↪In such a
                      way, each FTBL file will be produced from combination␣
↪of MTF files
                      given directly in this option (constant part) and␣
↪files given on a
                      corresponding row of 'vmtf' file. vmtf file is a TSV␣
↪file with
                      columns using the same names: 'netw', 'linp', etc.␣
↪Each row contains
                      file names that will be used to produce an FTBL file.␣
↪Thus each row
                      must have 'ftbl' column with unique and non empty name.
↪ When 'vmtf'
                      is used, 'ftbl' cannot be present on the command line.␣
↪If a file
                      type is present both in column names of 'vmtf' and in
↪'--mtf' option
                      then the content of 'vmtf' file will take precedence.␣
↪Empty values
                      in 'vmtf' file are ignored. All file paths in 'vmtf'␣
↪file are
                      considered relative to the location of 'vmtf' file␣
↪itself.
                    Only first 3 files are necessary to obtain a workable␣
↪FTBL file, others
                    are optional.
                    Example: 'txt2ftbl --mtf ecoli.netw,glu08C1_02U.linp,
↪cond1.miso,cond1.mflux'
                    NB: no space is allowed around comas. If a file path has␣
↪a spaces in
                    its name, it must be enclosed into quotes or double␣
↪quotes.
                    If an entry file cannot be renamed to have some of these␣
↪extensions, then
                    they can be used as prefixes followed by a '=' sign, e.g.
                      'txt2ftbl --mtf netw=ecoli.txt,linp=glu08C1_02U.tsv,
↪cond1.miso,cond1.mflux'
                    As you can see from this example, both naming schemes can␣
↪be mixed.
                    If for some reason, the same type of file is indicated␣
↪several times
                    (no matter with extension or prefix), the last occurrence␣
↪supersedes
                    all precedent ones.
  --prefix PREFIX   If all input files have the same name pattern and are␣
↪different only
                    in extensions then the pattern can be given as PREFIX, e.
↪g.
```

(continues on next page)

(continued from previous page)

```
                    '--prefix somedir/ecoli'
                  Then in 'somedir', we suppose to have 'ecoli.netw',
↪'ecoli.linp' and
                  other input files having names starting with 'ecoli' and␣
↪ending with
                  corresponding extensions.
                  NB. If some file is given in more than one option: '--
↪prefix' and/or
                  '--mtf' then the last occurrence overrides precedent ones.
  --eprl EPRL     Parallel experiments can be given with this option. It␣
↪must
                  introduce a couple of linp/miso files and optional␣
↪auxiliary ftbl name. These files
                  correspond to a given parallel experiment. This option␣
↪can be repeated as
                  many times as there are additional parallel experiments,␣
↪e.g.
                    'txt2ftbl --mtf ec.netw,glc6.linp,glc6.miso --eprl glc1.
↪linp,glc1.miso --eprl glc4.linp,glc4.miso'
                  This command will produce a main FTBL file 'ec.ftbl'␣
↪including all necessary
                  sections (NETWORK, etc.) but also two auxiliary FTBL␣
↪files: 'glc1.ftbl' and
                  'glc4.ftbl' having only label input/measurement sections.␣
↪They will correspond
                  to 2 additional parallel experiments. If ftbl file is not␣
↪given in --eprl
                  option, the name of miso file will be used for it. If␣
↪intermediate
                  directories in ftbl path are non existent they will be␣
↪silently created.
                  Auxiliary ftbl names will be put in 'OPTIONS/prl_exp'␣
↪field on the main ftbl file.
                  These names will be written there in a form relative to␣
↪the main ftbl.
                  To shorten the writings, it is possible to indicate only␣
↪one of two .miso/.linp files.
                  The other one will be guessed if it has canonical␣
↪extension. If extension is omitted then .miso and .linp files are searched␣
↪with these extensions. In this case, several parallel experiments can be␣
↪given with one --eprl option. So that above example can be shorten to:
                    'txt2ftbl --mtf ec.netw,glc6.linp,glc6.miso --eprl glc1,
↪glc4'
  --inst          Prepare FTBL for instationary case. File 'netw' is␣
↪supposed to have
                  column 'Time' non empty. Isotopic kinetic data will be␣
↪written to a TSV
                  file with 'ikin' extension. Its name will be the same as␣
↪in FTBL file,
                  and FTBL field 'OPTIONS/file_labcin' will contain 'ikin'␣
↪file name.
  --force         Overwrite an existent result file not produced by this␣
↪script.
                  NB. If a result file exists and is actually produced by␣
↪this script,
                  then it is silently overwritten even without this option.␣
↪The script
```

(continues on next page)

(continued from previous page)

```
                    detects if it was the creator of a file by searching for␣
↪a string "//
                    Created by 'txt2ftbl" at the first line of the file. By␣
↪removing or
                    editing this comment, user can protect a file from a␣
↪silent
                    overwriting.
```

ftbl2xgmml: cytoscape view

Once a valid FTBL file is generated, a user can visualize a graph representing his metabolic network in Cytoscape program. To produce necessary graph files, user can run:

```
$ ftbl2xgmml.py mynetwork[.ftbl] [> mynetwotk.xgmml]
```

or drag and drop `mynetwork.ftbl` icon on `ftbl2xgmml.py` icon.

The output redirection is optional.

This will produce a file in the XGMML format `mynetwork.xgmml` in the directory of `mynetwork.ftbl`:

Once a generated file `mynetwork.ftbl` is imported in cytoscape, a user can use one of automatic cytoscape layouts or edit node's disposition in the graph by hand. For those who use CySBML plugin, a saving of a particular layout in a file can be practical for later applying it to a new network.

Graphical conventions used in the generated XGMML are the following:

- specie are presented as rounded square nodes;
- simple (one to one) reaction are represented by simple edges;
- condensing and/or splitting reactions are represented by edges converging and/or diverging from an additional almost invisible node having a label with the reaction name;
- all nodes and edges have tool tips, i.e., when a pointer is put over, their name (specie or reaction) appears in a tiny pop-up window;
- non-reversible reactions are represented by a single solid line, have an arrow on the target end (i.e., produced specie) and nothing on the source end (i.e., consumed specie);
- reversible reactions are represented by a double parallel line and have a solid circle on the source end;
- color code for arrows:
  - green for free net flux;
  - blue for dependent net flux;
  - black for constrained net flux;
- color code for solid circles:
  - green for free exchange flux;
  - blue for dependent exchange flux;
  - black for constrained exchange flux.

ftbl2netan: FTBL parsing

To see how an FTBL file is parsed and what the parsing module "understands" in the network, a following command can be run:

```
$ ftbl2netan.py mynetwork[.ftbl] [> mynetwork.netan]
```

The output redirection is optional.

A user can examine `mynetwork.netan` in a plain text editor (not like Word) or in spreadsheet software. It has an hierarchical structure, the fields are separated by tabulations and the field values are Python objects converted to strings.

ftbl2cumoAb: human-readable equations

Sometimes, it can be helpful to examine visually the equations used by `influx_si`. These equations can be produced in human-readable form by running:

```
$ ftbl2cumoAb.py -r mynetwork[.ftbl] [> mynetwork.sys]
```

or:

```
$ ftbl2cumoAb.py --emu mynetwork[.ftbl] [> mynetwork.sys]
```

The output redirection is optional.

The result file `mynetwork.sys` will contain systems of stoichiometric and cumomer balance equations as well as a symbolic inversion of stoichiometric matrix. I.e., dependent fluxes are represented as a linear combination of free and constrained fluxes and an optional constant value. In the examples above, the option `-r` stands for "reduced cumomer set" and `--emu` stands for "generate EMU framework equations". In this latter case, only isotopologues of mass+0 in each EMU are reported in `mynetwork.sys` file. For other mass weights, the equations does not change and the right-hand side term could get longer for condensation reactions but involves the same EMUs as in mass+0 weight.

If a full cumomer set has to be examined, just omit all options. Keep in mind that on real-world networks this can produce more than a thousand equations by cumomer weight, which could hardly be qualified as *human*-readable form. So use it with caution.

For the sake of brevity, cumomer names are encoded in decimal integer form. For example, a cumomer `Metab#xx1x` will be referred as `Metab:2` because a binary number `0010` corresponds to a decimal number `2`. The binary mask `0010` is obtained from the cumomer mask `xx1x` by a plain replacement of every `x` by `0`.

For a given cumomer weight, the equations are sorted alphabetically.

expa2ftbl: non-carbon carrying fluxes

Deprecated since v6.0. Such kind of fluxes can be directly incorporated in .netw file.

Some reactions of carbon metabolism require cofactor usage like ATP/ADP and some others. A mass balance on cofactors can produce additional useful constraints on the stoichiometric system. Since the version 2.8, such mass balance equation on non carbon carrying species can be put in `EQUATION` section of FTBL file. A utility `expa2ftbl.R` can be helpful for this purpose if a user has already a full set of reactions in expa format. To extract additional equation from an expa file, `expa2ftbl.R` can be used as:

```
$ R --vanilla --slave --args file.expa < expa2ftbl.R > file.ftbl_eq
```

Then an information for the generated `file.ftbl_eq` has to be manually copy/pasted to a corresponding FTBL file.

Note that `expa2ftbl.R` uses a Unix command `grep` and another utility described here above `ftbl2netan.py`.

res2ftbl_meas: simulated data

During preparation of a study, one of the questions that biologist can ask is "Will the intended collected data be sufficient for flux resolution in a given network?" Some clue can be obtained by making "dry runs" of `influx_si` with `--noopt` (i.e. no optimization) option. User can prepare an FTBL file with a given network and supposed data to be collected. At first, the measurement values can be replaced by NAs while the SD values for measurements must be given in realistic manner. After running:

```
$ influx_s.py --noopt mynetwork
```

a utility `res2ftbl_meas.py` can be practical for preparing FTBL files with obtained simulated measurements:

```
$ res2ftbl_meas.py res2ftbl_meas.py mynetwork_res[.kvh] > mynetwork.ftbl_meas
```

(here `.kvh` suffix is optional). The information from the generated file `mynetwork.ftbl_meas` has to be manually copy/pasted into the corresponding FTBL file. Getting an ftbl file with real values instead of NAs in measurement sections gives an opportunity to explore optimization behavior near a simulated point like convergence speed and/or convergence stability to cite few of them.

ffres2ftbl: import free fluxes

This utility imports free flux values and specie concentrations (if any) from a result file _res.kvh and inject them into an FTBL file. Usage:

```
$ ffres2ftbl.sh mynetwork_res.kvh [base.ftbl] > new.ftbl
```

If an optional argument `base.ftbl` is omitted, then the free flux values are injected into an FTBL file corresponding to the _res.kvh file (here `mynetwork.ftbl`). This script can be used on a Unix (e.g., Linux, MacOS) or on a cygwin (Unix tools on Windows) platform. It makes use of another utility written in python `ff2ftbl.py`

ftbl2kvh: check ftbl parsing

This utility simply parses a ftbl file and write what was "understood" in a kvh file. No network analysis occurs here unlike in `ftbl2netan` utility. Usage:

```
$ ftbl2kvh.py mynetwork[.ftbl] [> mynetwork.kvh]
```

The output redirection is optional.

ftbl2metxml: prepare MetExplore visualization

Convert an FTBL file to an xml file suitable for visualization on MetExplore site. If a result kvh file `mynetwork_res.kvh` is present, it will be parsed to extract flux values corresponding to the last `influx_si` run and put them in `mynetwok_net.txt`, `mynetwork_fwd.txt` and `mynetwork_rev.txt`. As their names indicate, they will contain net, forward and revers flux values respectively.

IsoDesign: optimizing input label

One of the means to increase a flux resolution can be an optimization of input label composition. A utility `IsoDesing` solving this problem was developed by Pierre Millard. It is not part of `influx_si` distribution and can be downloaded at http://metatoul.insa-toulouse.fr/metasys/software/isodes/. In a nutshell, it works by scanning all possible input label compositions with a defined step, running `influx_si` on each of them. Then, it collects the SD information on all fluxes for all label compositions and finally selects an input label composition optimal in some sens (according to a criterion chosen by a user).

CHAPTER

SIX

FTBL FORMAT EVOLUTION

Even if FTBL is no more a front-end format for `influx_si` software (it was replaced by MTF to this end), we consider useful to keep track of its evolution during `influx_si` development.

Introduction

FTBL format was conceived by authors of `13CFlux` software in late 1990's (cf. https://www.13cflux.net/). At the beginning of 2000's, `13CFlux` became well spread in scientific community working on metabolism and isotope labeling. When we published the first version of `influx_s` in 2011, we adopted FTBL format to avoid cumbersome rewriting of networks and data already in use by the community. Second version of 13CFlux, published in 2012, abandoned FTBL format which was replaced by FluxML (XML) and was accompanied by a tool for automatic conversion of FTBL to FluxML.

On our side, we decided to continue to use FTBL by extending and evolving some of its features till its replacement by MTF (starting from v6.0). These extensions and evolution are presented hereafter for keeping tracks only. This chapter is not necessary for reading to successfully use `influx_si` software. Version number in titles indicates when described feature was first introduced to `influx_si`.

`METABOLITE_POOLS` and `METAB_MEASUREMENTS` (v2.0)

Sections `METABOLITE_POOLS` and `METAB_MEASUREMENTS` concerning metabolite pools were added. These sections can be useful for stationary labeling when growth fluxes are modeled with μM terms (cf. *Growth flux option*) or when some metabolites are confounded in measurements due to cell compartmentation of co-elution during HPLC step or whatever reason. These sections become mandatory for `influx_i` usage for instationary labeling as not only fluxes but also metabolite concentrations impact label propagation dynamics.

`METABOLITE_POOLS` is structured in two columns named `META_NAME` and `META_SIZE` and as ussual for FTBL indented and separated by tabulations, e.g.

```
METABOLITE_POOLS
        META_NAME       META_SIZE
        AKG     -0.5
        ...
```

Note: The value `-0.5` is not aligned with its column name `META_SIZE` because by default, tab characters are expanded to 8 spaces. As `META_NAME` occupies 9 spaces, `META_SIZE` is just shifted to the next tab position. User has to use only one tab character to separate columns even if they don't look aligned on his screen.

For `influx_i`, every internal metabolite (i.e. metabolites present in `NETWORK` section and not being input or output metabolites) and participating in carbon exchange must be referenced in this section. The value given in the

column META_SIZE is a metabolite concentration. The unit used for these values must be in accordance with the units used for fluxes. For example, if metabolite concentrations are measured in mM/g then fluxes are supposed to be measured in mM/(g*[time_unit]). If the value is positive then corresponding metabolite is considered as having constant concentration which does not vary during fitting iterations. If the value is negative, then this metabolite concentration will be part of fitted variables and its absolute value is used as a starting value for these iterations. A final fitted value will be expressed as a positive number.

For influx_s, this section is optional and only few (not all) internal metabolites can be present in this section.

METAB_MEASUREMENTS section regroups measurements of internal metabolite concentrations. Input and output metabolites may have concentrations varying during an experiment as they are consumed or produced. So they cannot appear in this section. METAB_MEASUREMENTS section has 3 columns: META_NAME, VALUE and DEVIATION, e.g.

```
METAB_MEASUREMENTS
        META_NAME       VALUE   DEVIATION
        Fru6P   0.43    0.01
        ...
```

Column names are self explanatory.

In case of confounded measurements, confounded metabolites can be given as a sum, e.g.

```
METAB_MEASUREMENTS
        META_NAME       VALUE   DEVIATION
        R5P_c+R5P_m     0.32    0.01
        ...
```

In this case, the value 0.32 will be fitted by a sum of simulated metabolite concentrations.

Long reactions (v4.0)

Initially, FTBL admitted no more than 2 metabolites on each side of reactions put in NETWORK section. We had to overcome this limit to facilitate FTBL creation for studies including reactions much longer than that. Now, chemical reaction having more than two metabolites on any side can be split in several sub-reactions, each of which has no more then 2 metabolites on every side. It is important that all sub-reactions be put together one after another and that they have the same name. Based on this name, influx_si will assemble all parts in one reaction. E.g. a reaction named Val_syn

```
Val_syn: Pyr (abc) + Pyr (def) + Glu (ghijk) + NADPH -> Val (abcef) + CO2 (d) + AKG␣
↪(ghijk)
```

can be translated into FTBL format as

```
NETWORK
       FLUX_NAME        EDUCT_1 EDUCT_2 PRODUCT_1       PRODUCT_2
       Val_syn Pyr      Pyr     Val     CO2
               #abc     #def    #abcef  #d
       Val_syn Glu      NADPH   AKG
               #ghijk   #       #ghijk
```

If some reactions have the same name but not placed sequentially one after another, it will be signaled as an error.

Cofactors (v4.0)

Here, we call cofactors metabolites that does not participate in carbon transfer from one or several molecules to another. The main interest of entering cofactors in carbon transferring reactions is additional balance equations that we can put in stoechiometric system. Thus the number of free fluxes is diminished and fluxes are constrained to more realistic values, not violating cofactor balances.

To indicate that a metabolite is a cofactor, user can simply put an empty carbon string in the corresponding carbon transferring line. For example, a reaction

```
v8: PEP (abc) -> Pyr (abc) + ATP
```

can be translated into FTBL as

```
NETWORK
       FLUX_NAME       EDUCT_1 EDUCT_2 PRODUCT_1       PRODUCT_2
       v8      PEP             Pyr     ATP
               #abc            #abc    #
```

Note an empty carbon string `#` at the place corresponding to `ATP`. An important difference between cofactors and other metabolites that the former are allowed to have stoechiometric coefficients different from 1. These coefficients must be separated from cofactors by `*` sign, e.g. a reaction

```
v41: Asp (abcd) + 2 ATP + NH3 -> Asn (abcd)
```

can be translated into FTBL as

```
NETWORK
       FLUX_NAME       EDUCT_1 EDUCT_2 PRODUCT_1       PRODUCT_2
       v41     Asp     2*ATP   Asn
               #abcd   #       #abcd
       v41     NH3
               #
```

Note the presence of `2*ATP` term.

Same metabolite on both sides of reaction (v4.0)

In some particular cases, it can be necessary to have a same metabolite on both sides of reaction. Let us illustrate this situation with the following example:

```
v71: CO2.unlabeled (a) + CO2 (b) -> CO2 (a) + CO2.out (b)
```

Metabolite CO2 is present on both sides of reaction but its carbon atom is not the same. This is the main reason for introducing this feature, to allow tracer rearrangement. In FTBL, it gives

```
NETWORK
       FLUX_NAME       EDUCT_1 EDUCT_2 PRODUCT_1       PRODUCT_2
       v71     CO2.unlabeled   CO2     CO2     CO2.out
               #a      #b      #a      #b
```

Section NOTRACER_NETWORK (v4.0)

In addition to reactions with carbon rearrangements, it can be useful to add reactions with no carbon transfer. The most known reaction of such type is biomass composition but it can there be others, e.g. involving exclusively cofactors:

```
v61: NADH + 0.5 O2 -> 2 ATP
```

This optional section is structured in 2 columns: FLUX_NAME and EQUATION:

```
NOTRACER_NETWORK
       FLUX_NAME        EQUATION
       v61      NADH+0.5*O2 = 2*ATP
```

You can see that the reaction is written in a manner very different form NETWORK section. Its sides are separated by = sign, metabolites are separated by + and they can have stoechiometric coefficients separated by * symbol. It is not visible in this example, but there can be as many metabolites as desired on each side of reaction. The limit "no more than 2 metabolites by side" proper to NETWORK section does not apply here.

Sub-sections EQUALITY/METAB and INEQUALITY/METAB (v2.11)

In the same manner as for fluxes, user can have to constrain variable metabolite concentrations. Constraints can be by equalities and inequalities. These subsections are organized in the same way as for fluxes. In EQUALITY/METAB there are 2 columns VALUE and FORMULA while in INEQUALITY/METAB there are 3 of them: VALUE, COMP and FORMULA. For example,

```
EQUALITIES
       METAB
                VALUE   FORMULA
                0       R5P - 1.5*X5P
                ...
INEQUALITIES
       METAB
                VALUE   COMP    FORMULA
                0.001   <=      PEP
                10      >=      PEP
                ...
```

NA in measurements (v2.5)

Missing values marked as NA are admitted in measurement sections, in columns designated to values. In contrast, they are not admitted in columns designated to standard deviations. The main difference between a measurement just omitted and those marked as NA is that the latter will be simulated and reported in corresponding simulation sections of the result file. This feature can be useful for preliminary simulations when there is no yet data available but user want to know e.g. if fluxes of interest will be well determined or not based on a supposed set of measurements. In this case, all presumed data can be set to NA (but not their SD).

Optimization control parameters (v5.3)

Optimization method(s) can be tuned by control parameters that can be put in OPTIONS section. The format of those fields has changed. Before, the field names were looking like optctrl_maxit i.e. a prefix optctrl_ followed by a parameter name, here maxit. Starting from v5.3, they look like optctrl:nlsic:maxit i.e. a prefix optctrl followed by a method name (here nlsic) and ended by parameter name, like maxit, all 3 separated by colon :. This new format allows tuning parameters for multiple optimization methods simultaneously. It became necessary, as starting from v5.3, several optimization methods can be used successively in one influx_si run. More about parameters can be found in the section *Optimization options*.

CHAPTER

SEVEN

PROGRAMMER'S DOCUMENTATION FOR INFLUX_SI

In this chapter, Application Programming Interface (API) docs are collected. It can be helpful for programmers desiring to extend some features of `influx_si` or to fix some bugs. This chapter can be safely skipped by users aiming at simple usage of `influx_si` for biological research.

C13_ftbl

- Parse .ftbl
- Analyse ftbl

Restrictions:

- metabolite name cannot have

  ":" it's a separator in measure id

  "+" in measurements it can be metab1+metab2+...

C13_ftbl.**aglom**(*na*, *ta*, *loop*)
: new matrix A (na), transpose A (ta) are used to aglomerate neigbour mutually influencing nodes in a supernode. Aglomerated noeds are put in the loop dictionnary. Return False if no nodes were aglomerated.

C13_ftbl.**aglom_loop1**(*A*)
: Agglomerate nodes of A if they mutually influence each other i.e. they are in a loop of length 1. Return a new dictionary of influence where entries are those of A aglomerated and glued "by" tab symbol

C13_ftbl.**allprods**(*srcs*, *prods*, *isos*, *metab*, *isostr*)
: Return a set of tuples (cmetab, cisostr, vmetab, visostr) where cmetab and cisostr describe a contex metabolite which combined with metab+isostr produced vmetab+visostr. if metab is alone on its reaction part cmetab and cisostr are set to an empty string (""). The set covers all combination of metab+isostr and its co-substrates which produce isotopes having at least one labeled carbon from metab+isostr. Co-substrate isotops are in a dictionary isos[cmetab]=list(cisotopes).

C13_ftbl.**bcumo_decomp**(*bcumo*)
: bcumo is a string of the form #[01x]+. It has to be decomposed in the linear combination of cumomers #[1x]+. The coefficients of this linear combination are 1 or -1. So it can be represented as sum(cumos_positive)-sum(cumos_negative). The result of this function is a dictionary {"+": list of icumos, "-": list of icumos}. icumo is an integer whose binary form indicates 1's positions in a cumomer.

C13_ftbl.**conv_mid**(*x*, *y*) → z
: convolute two mid vectors (numpy arrays) and return the result as numpy array.

C13_ftbl.**cumo_infl**(*netan*, *cumo)->list(tuple(in_cumo*, *fl*, *imetab*, *iin_metab)*)
: return the list of tuples (in_cumo, fl, imetab, iin_metab): input cumomer, flux (fwd.fl or rev.fl), index of metab

and index of in_metab generating cumo. cumo is in format "metab:icumo". Condenstation reaction will give the same flux and icumo but various iin_metab. Convergent point will give multiple fluxes.

C13_ftbl.**cumo_iw**(*w*, *nlen*)
iterator for a given cumomer weight w in the carbon length nlen

C13_ftbl.**cumo_path**(*starts*, *A*, *visited={}*)
Enumerate cumomers along reaction pathways. Algo: start from an input, follow chemical pathways till no more neighbours or till only visited metabolite rest in network. Return a list of cumomer pathways. Each pathways is an ordered list.

C13_ftbl.**dom_cmp**(*A*, *i*, *j*)
Compares influances of i-th and j-th lements of A. Returns 0 if i and j are mutually influenced, 1 if i in A[j] (i influences j) , -1 otherwise

C13_ftbl.**enum_path**(*starts*, *netw*, *outs*, *visited={}*)
Enumerate metabilites along to reaction pathways. Algo: start from an input, follow chemical pathways till an output or already visited metabolite. Returns a list of metabolite pathways. Each pathways is an ordered list.

C13_ftbl.**formula2dict**(*f*, *pterm=re.compile('([+-])')*, *pflux=re.compile('(?P<coef>\\d+\\.?\\d*|^)?\\s**?\\s*(?P<var>[a-zA-Z_[\\](][\\w\\.-\\[\\]]*)\\W*')*)
parse a linear combination sum([+|-][a_i][*]f_i) where a_i is a positive number and f_i is a string starting by non-digit and not white character (# is allowed). Output is a dict f_i:[+-]a_i

C13_ftbl.**frag_prod**(*metab*, *frag*, *s*, *cmetab*, *cfrag*, *cs*, *prods*)
Get fragments from labeled substrates

C13_ftbl.**ftbl_netan**(*ftbl*, *netan*, *emu_framework=False*, *fullsys=False*, *case_i=False*)
analyse ftbl dictionary to find

- network inputs (input)
- network outputs (output)
- substrates (subs)
- products (prods)
- metabolites (metabs)
- reactions (reacs)
- not reversible reactions (subset of reacs) (notrev) all above items are in named sets
- stocheometric matrix (sto_r_m)
- stocheometric matrix (sto_m_r)
- fwd-rev flux matrix (flux_m_r)
- cumomer balances (cumo_m_r_m)
- carbon length (Clen)
- reaction formula (formula)
- metabolite network (metab_netw)
- carbon transitions (carbotrans)
- free fluxes (flux_free)
- constrained fluxes (flux_constr)
- measured fluxes (flux_measured)
- variable growth fluxes (flux_vgrowth)

- input isotopomers (iso_input)
- input isotopomers functions (funlab for case_i=True)
- input cumomers (cumo_input)
- input reduced cumomers (rcumo_input)
- flux inequalities (flux_ineqal)
- flux equalities (flux_eqal)
- label measurements, H1 (label_meas)
- peak measurements, C13 (peak_meas)
- mass measurements (mass_meas)
- cumomer ordered lists (vcumo)
- unknown fluxes ordered lists (vflux)
- linear problem on fluxes (Afl, bfl)
- free fluxes ordered lists (vflux_free)
- fw-rv fluxes ordered lists (vflux_fwrv)
- row names ordered lists for Afl (vrowAfl)
- in-out fluxes (flux_in, flux_out)
- measured concentrations (metab_measured)

C13_ftbl.**ftbl_parse**(*f*) → dict
read and parse .ftbl file. The only input parameter f is a stream pointer with read permission or a file name. This function parses the input and returns a dictionnary with items corresponding to sections in .ftbl. One section is added. "TRANS" correponds to carbon transitions.

C13_ftbl.**infl**(*metab*, *netan)->oset(fluxes*)
List incoming fluxes for this metabolite (fwd.reac, rev.reac, . . .)

C13_ftbl.**iso2cumo**(*netan*, *strin*, *in_cumo*, *icumo*, *in_metab*)
calculate cumomer fraction from isotopomer ones

C13_ftbl.**iso2emu**(*netan*, *inmetab*, *mask*, *mpi*, *e*)
calculate emu fraction from isotopomer dict iso_input. The fraction corresponds to a fragment defined by a mask and the mass component mpi. Return a real number in [0; 1] interval.

C13_ftbl.**label_meas2matrix_vec_dev**(*netan*)
use netan["label_meas"] list to construct a corresponding list of measure matrix matx_lab such that scale_diag*metab_pool_diag*matx_lab*(cumos_vector,1) corresponds to label_measurements_vector. matx_lab is defined as list of dict{"scale":scale_name, "coefs":dict{icumo:coef}, "metab": metabolite, "poolid": metabolite pool id if pooled} where coef is a contribution of cumo in linear combination for given measure. scale_name is of the form "metabs;group". Group number is to group measurements of the same measurement set. poolid is the index of pool list in pooled where each list regroups 0-based indexes rows in returned matrix for what has to be pooled together. vec is a list of measurements (values in .ftbl) dev is a list of deviations. Elements in matx_lab, vec and dev are ordered in the same way. The returned result is a dict (mat,vec,dev)

C13_ftbl.**labprods**(*prods*, *metab*, *isostr*, *strs*)
Return a set of tuples (vmetab,visostr) which receive at least one labeled carbon from (metab, isostr)

C13_ftbl.**lowtri**(*A*)
Try low triangular ordering of matrix A entries

C13_ftbl.**mass_meas2matrix_vec_dev**(*netan*)
use netan["mass_meas"] list to construct a corresponding list of measure matrix matx_mass such that scale_diag*matx_mass*cumos_vector corresponds to mass_measures_vector. matx_mass is defined as matx_lab in label_meas2matrix_vec_dev() Elements in matx_mass, vec and dev are ordered in the same way. scale name is defined as "metab;fragment_mask" The returned result is a dict (mat,vec,dev)

C13_ftbl.**mat2graph**(*A*, *fp*)
write digraph file on file pointer fp representing links in matrix A given as bi-level dictionnary. A key of first level (row index) is influenced by keys of second level (column indicies).

C13_ftbl.**mat2pbm**(*A*, *v*, *fp*)
Write an image map of non-zero entries of matrix A to file pointer fp. Matrix A is a dictionnary, v is a list ordering keys of A.

C13_ftbl.**mecoparse**(*terms*, *pmeco=re.compile('\\s*((?P<coef>\\d+\\.?\\d*|^)\\s**\\s*)?(?P<metab>[^*+]*)\\s*$')*)
Parse a string term from a list (or a sing string) of chemical equation entries. The general form of each term is 'coef*metab'. coef (if present) must be separated from metab by '*' and be convertible to float. metab can start with a number (e.g. '6PG') so the presence of '*' is mandatory to separate coef from metab.If coef is absent, it is considered to be 1. Return a list of (or a single for str) tuples (metab (str), coef (real)).

C13_ftbl.**mkfunlabli**(*d*)
transform 'd' dict to a string representing a body of an R function calculating labeling dependent on time 't'

C13_ftbl.**ms_frag_gath**(*netan*)
gather metabolite fragments necessary to obtain a given set of data observed in MS measurements. The fragment mask is encoded in the same way as cumomers, Met:7 <=> Met#(0)111

C13_ftbl.**ntimes**(*n*)
Return charcater string 'once' for n=1, 'twice' for n=2 and 'n times' for other n

class C13_ftbl.**oset**(***kwds*)

copy() → a shallow copy of D

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

C13_ftbl.**peak_meas2matrix_vec_dev**(*netan*, *dmask={'D+': 3*, *'D-': 6*, *'DD': 7*, *'S': 2*, *'T': 7}*)
use netan["peak_meas"] list to construct a corresponding list of measure matrix matx_peak such that scale_diag*matx_peak*cumos_vector corresponds to peak_measures_vector. dmask is a dictionary with 3 carbon labeling pattern mask for various peak types. The middle bit corresponds to the targeted carbon, lower bit corresponds to the next neighbour (D+) and higher bit corresponds to previous carbon (D-). matx_peak is defined as matx_lab in label_meas2matrix_vec_dev() Elements in matx_peak, vec and dev are ordered in the same way. scale name is defined as "metab;c_no;irow" The returned result is a dict (mat,vec,dev)

C13_ftbl.**proc_kinopt**(*ftbl*, *netan*)
Proceed label kinetics options from OPTIONS section: file_labcin, dt, tmax, nsubdiv_dt, funlab

C13_ftbl.**proc_label_input**(*ftbl*, *netan*, *case_i=False*)
Proceed LABEL_INPUT section in ftbl and add result to the list netan["iso_input"] and netan["funlab"] (case_i) List item is a dict {}metab;{isotop_int_index:fraction}}

C13_ftbl.**proc_label_meas**(*ftbl*, *netan*)
Proceed LABEL_MEASUREMENT section of ftbl file, add the result to a list of dicts

C13_ftbl.**proc_mass_meas**(*ftbl*, *netan*)
Proceed PEAK_MEASUREMENT section of ftbl file, add the result to a list of dicts

C13_ftbl.**proc_peak_meas**(*ftbl*, *netan*)
Proceed PEAK_MEASUREMENT section of ftbl file, add the result to a list of dicts

C13_ftbl.**prod**(*metab*, *iso*, *s*, *cmetab*, *ciso*, *cs*, *prods)->oset(*)
get isotops from labeled substrates

C13_ftbl.**rcumo_sys**(*netan*, *emu=False*)
Calculate reduced cumomers or EMU systems A*x=b we start with observed cumomers (emus) of max weight and we include only needed involved cumomers (emus) A list of cumomer (emu) lists (by weight) is stored in netan["vrcumo"] (netan["vemu"])

C13_ftbl.**src_ind**(*substrate*, *product*, *iprod*)
For a given substrate and product carbon strings (e.g. "abc", "ab") calculate substrate index corresponding to product index. Return None if no source found. Return 0 if iprod==0 and intersection of product and substrate strings is not empty

C13_ftbl.**t_iso2cumo**(*n*)
t_iso2cumo(n) return transition matrix from isotopomers fractions to cumomer vector n - carbon number return numpy array of size (2**n,2**n)

C13_ftbl.**t_iso2m**(*n*)
t_iso2m(n) return transition matrix from isotopomers fractions to MID vector n - carbon number return numpy array of size (n+1,2**n)

C13_ftbl.**t_iso2pos**(*n*)
t_iso2pos(n) return transition matrix from isotopomers fractions to positional labelling vector (cumomers of weight 1) n - carbon number return numpy array of size (n,2**n)

C13_ftbl.**topo_order**(*A*, *tA*)
Try to sort keys of A in topological order. tA is just a transpose of A

C13_ftbl.**transpose**(*A*)
Transpose a matrix defined as a dict.

C13_ftbl.**werr**()
Write string to stream. Returns the number of characters written (which is always equal to the length of the string).

C13_ftbl.**wout**()
Write string to stream. Returns the number of characters written (which is always equal to the length of the string).

txt2ftbl

transform a series of TXT and TSV files into FTBL file.

Copyright 2021, INRAE, INSA, CNRS Author: Serguei Sokol (sokol at insa-toulouse dot fr) License: Gnu Public License (GPL) v2 http://www.gnu.org/licenses/gpl.html

txt2ftbl.**compile**(*mtf*, *cmd*, *case_i=False*, *clen=None*)
Compile FTBL content from mtf names: netw, miso etc. Return a dict of ftbl lines

txt2ftbl.**dsec2out**(*dsec*, *fout*)
write lines from dsec to fout

txt2ftbl.**dtstamp**()
formatted date-time stamp

txt2ftbl.**itvl2li**(*v*)
convert interval like '2-5' to ['2', '3', '4', '5']

txt2ftbl.**parse_cnstr**(*f*)
Parse constraint TSV file. Return a tuple of 2 dicts (eq, ineq) with lines to add to ftbl

txt2ftbl.**parse_linp**(*f*, *clen={}*)
Parse label input TSV file. Return a list of lines to add to ftbl

txt2ftbl.**parse_mflux**(*f*, *dfl={}*)
Parse flux measurements TSV file. Return a list of lines to add to ftbl

txt2ftbl.**parse_miso**(*fmiso*, *clen*, *case_i=False*)
Parse isotopic measurements TSV file. Return dict with keys: ms, lab, peak

txt2ftbl.**parse_mmet**(*f*, *smet={}*)
Parse metabolite concentration measurements TSV file. Return a list of lines to add to ftbl

txt2ftbl.**parse_opt**(*f*)
Parse options TSV file. Return a list of lines to add to ftbl

txt2ftbl.**parse_tvar**(*f*, *dfl={}*, *itnl_met={}*)
Parse variable type TSV file. Return a tuple of a dict and a list with lines to add to ftbl

txt2ftbl.**try_ext**(*f*, *li*)
See if file 'f' exists, if not try with extensions from 'li'. The first found is returned as Path() otherwise an exception is raised.

txt2ftbl.**tsv2df**(*f*, *sep='\t'*, *comment='#'*, *skip_blank_lines=True*, *append_iline='iline'*)
Read file 'f' as TSV and return a DataFrame. Separator is 'sep', comment char is 'comment', blank lines are skept, header is in the first row, file line numbers are stored in a column 'line_nb' if there is no a column with this name

txt2ftbl.**txt_parse**(*ftxt*, *re_metab=re.compile('(?:(?P<coef>[\\d\\.]*)\\s+)?(?:(?P<metab>[^()\\t\\r]+)\\s*)(?:\\(\\s*(?P<carb>[^()]*)\\s*\\))?\\s*')*, *re_labpat=re.compile('^[./*\\d\\s]*(?P<labpat>[a-zA-Z]*)\\s*$')*)
Parse txt file from fname which is in format: ### Glycolysis and OPP pathway GLYC (abcdef) -> G6P (abcdef) G6P (abcdef) <-> F6P (abcdef) i.e.

comment: # blabla non reversible reaction: [reac:][N1] metab1 [(carb1)] [... + [N_i] metab_i [(carb_i)]] -> ... where

reac is an optional reaction name; N_i is optional stoechiometric coefficient metab_i is i-th metabolite name (carb_i) is optional 1-letter carbon names for carbon transition mapping

reversible reaction is represented by "<->" sign non reversible reaction is represented by "->" sign. reaction with imposed sens of reaction (from left to right) is representd with double ">>", i.e. "->>" for non reversible reaction or "<->>" for reversible reaction. This reaction with imposed sens will have an inequality "reac_name >= 0" in FTBL/INEQUAITIES/NET section.

Retrun a list with following items: - a list of carbon exchange reactions - a list of non carbon echanging reactions - a list of equalities net and xch - a list of flux tuples (reac, rev, imposed_sens) - a list of two lists: left and right metabolites [(metab, clen),] - a carbon length dictionary {met: N}

the first item is a list of: plain string == just a comments list == reaction items: input, output: lists of tuples (metab, carb, coeff)

ftbl2mtf

Parse ftbl file from first parameter or from stdin (if input file is '-') and write a series of mtf (multiple TSV files). The file stem ('network' in 'network.ftbl') is used as file name basis for produced files, e.g. 'network.miso'. Parameter –out can be used to change it. If out path includes non existing directories, they are automatically created. Caution!

If an existing output file starts with a comment "# Created by 'ftbl2mft ..." or is empty, it is silently overwritten. Otherwise, the writing is aborted with a warning. Other files may continue to be created. To force the overwriting, use '–force'.

Output files will have following extensions/meanings: .netw: stoichiometric equations and label transitions in the biochemical network; .linp: label input; .miso: isotopic measurements (MS, label, peak); .mflux: flux measurements; .mmet: biochemical specie concentration measurements; .tvar: flux/specie types partition (free, dependent, constrained) and starting values; .cnstr: constraints (equalities, inequalities for both fluxes and concentrations); .opt: options.

ftbl2mtf.**dtstamp**()
formatted date-time stamp

ftbl2code

Module for translation of .ftbl file to R code

ftbl2code.**netan2Abcumo_spr**(*varname*, *Al*, *bl*, *vcumol*, *minput*, *f*, *fwrv2i*, *incu2i_b1*)
Transform cumomer linear sytems collection (from ftbl file) to a R code calculating sparse matrix A and vector b in A*x+b=0 for a given weight of fragment iw (index in resulting list) Flux vector fl of all fwd. and rev. fluxes are known at R runtime.

Resulting code is a list sprAb indexed by cumomer weight (cf. generated R comments for details on sprAb) cumomer vector incu=c(1, xi, xl), xi - input cumomers, xl - lighter cumomers.

incu2i_b1 gives i in incu from cumomer name. i=1 corresponds to the constant 1.

ftbl2code.**netan2R_cumo**(*netan*, *org*, *f*) → dict
generate data structures for full cumomer matrices

ftbl2code.**netan2R_fl**(*netan*, *org*, *f*)
generate R code for flux and pool part for more details cf. netan2Rinit()

ftbl2code.**netan2R_ineq**(*netan*, *org*, *f*)
generate inequality code

ftbl2code.**netan2R_meas**(*netan*, *org*, *f*)
generate code for measure treatment

ftbl2code.**netan2Rinit**(*netan*, *org*, *f*, *fullsys*, *emu=False*, *ropts=[]*)
Write R code for initialization of all variables before cumomer system resolution by chi2 minimization. :param netan: a collection of parsed ftbl information :param f: R code output pointer :param fullsys (logical): write a code for the full or only reduced cumomer system :param emu (logical): write equations in EMU framework or cumomer (default) :param ropts: list of items "param=value" to be written as is in R file.

Returns a dictionnary with some python variables: * "measures": measures, * "o_mcumos": o_mcumos, * "cumo2i": cumo2i, * ...

ftbl2netan

Parse ftbl file from stdin or from first parameter and write netan in kvh format on stdout usage: ftbl2netan.py network[.ftbl] [-h] [-i] [–emu] [–clownr] [–fullsys] [> network.netan]

ftbl2optR

Transform an ftbl to R code which will solve an optimization of flux analysis problem $\arg\min_\Theta S$, where $S = ||\text{Predicted} - \text{Observed}||^2_\Sigma$ and Θ is a vector of parameters to fit: free fluxes (net+xch), scaling parameters and metabolite concentrations pools. Two variants of R code can be generated: "s" and "i" for stationary and isotopically nonstationary labeling. Predicted vector is obtained from cumomer or emu vector x (calculated from free fluxes and divided in chunks according to the cumo weight) by multiplying it by the measurement matrices, weighted by metabolite pools (in case of pooling) and scale factor (for stationary case only), boths coming from ftbl file. Observed values vector xo is extracted from ftbl file for "s" case and from special text file for "i" case. It is composed of flux, label measurements and metabolite pools. Σ^2, covariance diagonal matrices sigma[flux|mass|label|peak|metab.pool] is orginated from the ftbl file.

usage: ./ftbl2optR.py [opts] organism where organism is the ftbl informative part of file name (before .ftbl), e.g. organism.ftbl after execution a file organism.R will be created. If it already exists, it will be silently overwritten. The system Afl*flnx=bfl is created from the ftbl file.

Important python variables:

- case_i - if True, the case is "i" otherwise it is the "s" case

Collections:

- netan - (dict) ftbl structured content
- tfallnx - (3-tuple[reac,["d"|"f"|"c"], ["net"|"xch"]] list)- total flux collection
- measures - (dict) exp data
- rAb - (list) reduced linear systems A*x_cumo=b (a system by weight)
- scale - unique scale names
- nrow - counts scale names
- o_sc - ordered scale names
- o_meas - ordered measurement types

File names (str):

- n_ftbl (descriptor f_ftbl)
- n_R (R code) (f)
- n_fort (fortran code) (ff)

Counts:

- nb_fln, nb_flx, nb_fl (dependent fluxes: net, xch, total), nb_ffn, nb_ffx (free fluxes)

Index translators:

- fwrv2i - flux names to index in R:fwrv
- cumo2i - cumomer names to index in R:x
- ir2isc - mapping measurement rows indexes on scale index isc[meas]=ir2isc[meas][ir]

Vector names:

- cumos (list) - names of R:x
- o_mcumos - cumomers involved in measurements

Important R variables:

Scalars:

- nb_w, nb_cumos, nb_fln, nb_flx, nb_fl (dependent or unknown fluxes),
- nb_ffn, nb_ffx, nb_ff (free fluxes),
- nb_fcn, nb_fcx, nb_fc (constrained fluxes),
- nb_ineq, nb_param, nb_fmn

Name vectors:

- nm_cumo, nm_fwrv, nm_fallnx, nm_fln, nm_flx, nm_fl, nm_par,
- nm_ffn, nm_ffx,
- nm_fcn, nm_fcx,
- nm_mcumo, nm_fmn

Numeric vectors:

- fwrv - all fluxes (fwd+rev)
- x - all cumomers (weight1+weight2+...)
- param - free flux net, free flux xch, scale label, scale mass, scale peak, metabolite concentrations
- fcn, fcx, fc - constrained fluxes
- bp - helps to construct the rhs of flux system
- xi -cumomer input vector
- fallnx - complete flux vector (constr+net+xch)
- bc - helps to construct fallnx
- li - inequality vector (mi%*%fallnx>=li)
- ir2isc - measure row to scale vector replicator
- ci - inequalities for param use (ui%*%param-ci>=0)
- measvec - measurement vector
- fmn - measured net fluxes

Matrices:

- Afl, qrAfl, invAfl,
- p2bfl - helps to construct the rhs of flux system
- mf, md - help to construct fallnx
- mi - inequality matrix (ftbl content)
- ui - inequality matrix (ready for param use)
- measmat - for measmat*x+memaone=vec of simulated not-yet-scaled measurements

Functions:

- lab_sim - translate param to flux and cumomer vector (initial approximation)
- cumo_cost - cost function (chi2)
- cumo_gradj - implicit derivative gradient

ftbl2xgmml

read a .ftbl file from a parameter and translate to .xgmml file. The generated xgmml file can be then imported into Cytoscape (www.cytoscape.org). Reactions involving two substrates or two products are represented by an additional almost invisible node while one-to-one reactions are just edges. Node and edge attributes are written in respective xml attributes. Compatibility: cytoscape v2.8.3 and v3.0

usage: ftbl2xgmml.py [-h|–help] mynetwork.ftbl [> mynetwork.xgmml]

OPTIONS -h, –help print this message and exit

param mynetwork the base of an ftbl file (mynetwork.ftbl)

returns mynetwork.xgmml – file of the network definition suitable for cytoscape

kvh

kvh.**dict2kvh**(*d*, *fp=sys.stdout*, *indent=0*)
Write a nested dictionary on the stream fp (stdout by default).

kvh.**escape**(*s*, *spch="|&;<>()$\"' tn*?[#~=%"*, *ech='\"*)
escape special characters in s. The special characters are listed in spch. Escaping is done by putting an ech string before them. Default spch and ech corresponds to quoting Shell arguments in accordance with http://www.opengroup.org/onlinepubs/009695399/utilities/xcu_chap02.html Example: os.system("ls %s" % escape(file_name_with_all_meta_chars_but_newline)); .. note:

1. Escaped <newline> is removed by a shell if not put in a single-quotted string (' ')
2. A single-quote character even escaped cannot appear in a single-quotted string

kvh.**kvh2dict**(*fp*, *strip=False*)
Read a kvh file from fp pointer then translate its tlist structure to a returned hierarchical dictionnary. Repeated keys at the same level of a dictionnary are silently overwritten

kvh.**kvh2obj**(*fp*, *strip=False*)
Read a kvh file from fp pointer then translate its tlist structure to a returned object hierarchy. Repeated fields at the same level of an object are silently overwritten

kvh.**kvh2tlist**(*fp, lev=[0], indent=[0], strip=False*)
Read a kvh file from fp stream descriptor and organize its content in list of tuples [(k1,v1), (k2,[(k2.1, v2.1)])] If fp is a string, it is used in open() operator

kvh.**kvh_get_matrix**(*fp*, *keys*)
Get matrix or vector whose key suite is in a list keys from a kvh file given in fp (file pointer of file name). For big kvh files, this function can be much faster than kvh2tlist()+kvh_getv_by_k() Return a matrix which is a list of lists (rows). The first item in each row is the row name. In case of matrix (i.e. "row_col" is present in kvh file), the very first row contain column names.

kvh.**kvh_getv_by_k**(*kvt*, *kl*) → None|String|kvh tlist
get value from kvt (kvh tlist) according to the key hierarchy defined in the list of keys kl. Return None if no key is found

kvh.**kvh_read_key**(*fp*, *strip=False*)

Read a string from the current position till the first unescaped , or the end of stream fp.

Returns tuple (key, sep), sep=None at the end of the stream

kvh.**kvh_read_val**(*fp*, *strip=False*)

Read a string from current position till the first unescaped

or the end of file. Return the read string.

kvh.**kvh_tlist2dict**(*tlist*)
Translate a tlist structure read from a kvh file to a hierarchical dictionnary. Repeated keys at the same level of a dictionnary are silently overwritten

kvh.**kvh_tlist2obj**(*tlist*)
Translate a tlist structure read from a kvh file to a hierarchical dictionnary. Repeated keys at the same level of a dictionnary are silently overwritten

kvh.**tlist2kvh**(*d*, *fp=sys.stdout*, *indent=0*)
Write a (hierarchichal) list of 2-tuples on the stream fp (stdout by default).

tools_ssg

tools_ssg.**aff**(*name*, *obj*, *ident=0*, *f=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>*)

print formatted object: name=obj

tools_ssg.**arr2pbm**(*A*, *fp*)
Write an image map of non-zero entries of matrix A to file pointer fp. Matrix A is an array

tools_ssg.**asort**(*d*)
sorts a dictionnary by value preserving key=value association the result is a list of tuples (key,value)

tools_ssg.**cumsum**(*l*, *tot=0*)

Returns an iterable of the length len(l)+1 with cumulated sum of items in l. First element in cumsum is equal to initial value of tot. Result depends on the meaning of "+" operator for l items and of tot type.

```
>>> list(cumsum("abc",tot=""))
['', 'a', 'ab', 'abc']
```

```
>>> list(cumsum(xrange(1,5)))
[0, 1, 3, 6, 10]
```

tools_ssg.**expandbit**(*i*, *pos*)
copy bits set to 1 in i to the result position given in the list pos. length of pos must be greater or equal to bitlength of i

tools_ssg.**icumo2iiso**(*icumo*, *size*)

Returns iterator on isotopomers composing a given icumo. size is carbon number

tools_ssg.**isstr**(*s*)

Returns True if the argument is a string

tools_ssg.**iterbit**(*i*, *size=0*)
iterator on bits in integer starting from 0-position. The iterator stops at highest non-zero bit

tools_ssg.**iternumbit**(*i*, *size=0*)
iterator on bits and its number in integer starting from 0-position. The iterator yields tuples (n,bit). If optional size is zero then it stops at highest non-zero bit. If not, it will stop at bit number size-1.

tools_ssg.**join**(*c*, *l*, *p=''*, *s=''*, *a=''*)
join the items of the list (or iterator) l separated by c. Each item is prefixed with p and suffixed with s. If the join result is empty for any reason, an alternative a is returned. p, s and a are optional

tools_ssg.**joint**(*c*, *l*, *p=''*, *s=''*, *a=''*)
join "true" items of the list (or iterator) l separated by c. Each item is prefixed with p and suffixed with s. If the join result is empty for any reason, an alternative a is returned. p, s and a are optional

tools_ssg.**list2count**(*l*, *incr=1*)
count values in a (short) list l incrementing the counter by optional incr.

Returns a dictionary {item:count}

tools_ssg.**read_table**(*f*) → dict(mat, col_names) read a plain text file f in a numpy mat. If some columns are not numerical, they are replaced by np.nan. If header=True, number of column names in the first row after skip must be the same as the number of values in each following row.

tools_ssg.**reverse**(*it*)
reverse order of an iterable

tools_ssg.**rstrbit**(*i*, *size=0*)

Returns the integer as reversed string binary representation. The lowest bit is on the left side

tools_ssg.**setbit32**(*i*, *nb*)
set a bit number nb (0 based) in an integer i

tools_ssg.**setcharbit**(*s*, *ch*, *i*)
set character ch in a string s everywhere a corresponding bit of i is set

tools_ssg.**ssign**(*i*, *sp='+'*, *sm='-'*)

Returns a string of i sign: sp (i>=0) or sm (i<0).

tools_ssg.**strbit**(*i*, *size=0*)

Returns the lowest part of integer as string binary representation

tools_ssg.**strbit2int**(*s*)
translate a string of 0's and 1's interpreted as bits to an integer all characters different from 0,1 are silently ignored.

tools_ssg.**strbit32**(*i*)

Returns a string of 0-1s (in chunk of 4) in an 32 bit integer

tools_ssg.**sumbit**(*i*)

Returns sum of bits in an integer

tools_ssg.**trd**(*l*, *d*, *p=''*, *s=''*, *a=''*)
translate items in an iterable l by a dictionary d, prefixing translated items by optional p and suffixing them by optional s. If an item is not found in the dictionnary alternative string a is used. If a==None, the item is left unchanged. No prefix or suffix are applied in both case.

Returns iterator

tools_ssg.**ulong**(*i*) → workarounded ulong

tools_ssg.**valval**(*o*, *keepNone=True*)

Returns an iterator over values of values, i.e. collapsing values of fisrt two nested lists in one list, for example.

tools_ssg.**wxlay2py**(*kvt, parent=[None]*)

Returns a string with python code generating wxWindow widget layout described in kvh tlist sturcture

CHAPTER

EIGHT

HOW TO . . .

. . . choose free fluxes?

You can define in FTBL all not constrained fluxes as dependent (put a letter `D` in the column `FCD` of the FTBL sections `FLUXES/NET` and `FLUXES/XCH`), run `influx_si` and see an error message that will suggest some candidates for free fluxes. For these fluxes, put a letter `F` in the column `FCD` and some numeric value in the next column `VALUE(F/C)` to provide a starting value for the fitting. Don't use 0 as starting value as it might lead to singular matrices in cumomer balances.

If you want to create an FTBL *de novo*, consider using application `txt2ftbl.py` included in `influx_si` package. Not only it translates an easily readable/writable text format into FTBL one, but it also automatically assigns some fluxes to be free.

. . . get statistical information for a given set of free fluxes without fitting measurements?

Put these values in the corresponding FTBL file as starting values for free fluxes and use `influx_si` with `--noopt` option.

. . . accelerate calculations?

You can relax stopping criterion and pass from 1.e-5 (by default) to, for example, 1.e-2 if this precision is sufficient for you. Use `optctrl:nlsic:errx` option in FTBL file (section `OPTIONS`) for this.

If you mean to accelerate Monte-Carlo simulations in Unix environment, you can use a hardware with many cores. In this case, the wall clock time can be reduced significantly. Note that distant nodes, even inside of the same cluster, are not used in the such kind of Monte-Carlo simulations.

Check that your system is not using swap (disk) memory. If it is the case, stop other applications running in parallel with `influx_si`. If possible extend the RAM on your hardware.

. . . extend upper limit for non linear iterations?

By default, this value is 50 which should be largely sufficient for most cases. If not, you can set another value via `optctrl:nlsic:maxit` option in the FTBL file (section `OPTIONS`). But most probably, you would like to check your network definition or to add some data or to change a substrate labeling, anyway to do something to get a well defined network instead of trying to make converge the fitting on some biologically almost meaningless situation.

. . . make FTBL file with synthetic data?

Follow for example steps outlined hereafter:

- edit FTBL file(s) with `NA` in measurements and realistic SD, name it e.g. `new_NA.ftbl`
- simulate data:

```
$ influx_s.py --noopt --addnoise new_NA
```

- prepare FTBL sections with simulated data:

```
$ res2ftbl_meas.py new_NA_res.kvh
```

It will create file (or files if there are parallel experiments) with synthetic data formatted for inclusion in FTBL file: `new_NA_sim1.ftbl`, `new_NA_sim2.ftbl`, etc.)

- copy/paste simulated data to a new file `new.ftbl` with numeric data instead of `NA`.
- use FTBL with synthetic data:

```
$ influx_s.py new.ftbl
```

... do custom post-treatment of Monte-Carlo iterations?

Let suppose you want to filter some of Monte-Carlo (MC) iterations based on their cost values. In `OPTIONS/posttreat_R` of your FTBL file add `save_all.R`. The file `save_all.R` can be found in `R` directory of `influx_si` distribution. Execution of `save_all.R` at the end of calculations will simply save all session variables in `mynetwork.RData` file (supposing that your FTBL file is named `mynetwork.ftbl`). In particular, you need `free_mc` matrix which contains free parameters (each column results from a given MC iteration). After that you can open an interactive R session in your working directory and run something similar to:

```
# preparations
load("mynetwork.RData")
source(file.path(dirx, "libs.R"))
source(file.path(dirx, "opt_cumo_tools.R"))
#source(file.path(dirx, "opt_icumo_tools.R")) # uncoment for influx_i use
tmp=sparse2spa(spa)

# doing something useful
# here, we calculate a vector of cost values, one per MC iteration
cost_mc=apply(free_mc, 2, function(p) cumo_cost(p, labargs))
# do something else ...
```

If, instead of cost values, you need for example a full set of net-xch fluxes then do

```
allflux_mc=apply(free_mc, 2, function(p) param2fl(p, labargs)$fallnx)
```

for residuals, do:

```
resid_mc=apply(free_mc, 2, function(p) lab_resid(p, FALSE, labargs)$res)
```

After that, you can filter or do whatever needed with obtained vectors and matrices.

CHAPTER

NINE

TROUBLESHOOTING

The software is provided "AS IS" without warranty of any kind explicit or implicit.

If you have some issue with `influx_si` you can try the following steps for solving them:

- partners and clients of MetaToul-RéseauxMétaboliques can benefit from advices of their dedicated stuff. If you are not a partner/client but would like to become one, for example to get help for your label experiment design and/or realization, you can contact platform MetaToul (cf. *Consulting and more*);
- you can search for similar problem discussion in the forum https://groups.google.com/forum/#!forum/influx_si. If you don't find your answer and wish to ask a new question you'll have to subscribe to this group.
- if you think that you face a bug, try the latest version of the software to see if this bug was already fixed. If it is still present, you can report it on https://github.com/sgsokol/influx/issues. Please note that we can't guarantee that any particular bug can be fixed in any particular release or can be fixed at all. It is possible, that we ask you to send us (in a private email not in influx_si@googlegroups.com) an ftbl file on which an error occur. It will be used only for purposes of bug reproducing and its identification. The received ftbl file will not be transmitted to any third party.
- if you have a problem with FTBL editing, you can read the documentation from 13CFlux and/or interpret error messages generated during FTBL parsing.
- if you have some difficulties in choosing free fluxes, define all not constrained fluxes as dependent (put a letter `D` in the column `FCD` of the FTBL sections `FLUXES/NET` and `FLUXES/XCH`) and see an error message that will suggest candidates for free fluxes. Another option is to use `--ffguess` flag that will automatically partition not constrained fluxes between free and dependent.
- if your resulting fluxes are badly defined (statistically or structurally), i.e. they have big confidence intervals or the Jacobian is rank deficient, you can try to play with input labeling (cf. IsoDesign software at http://metatoul.insa-toulouse.fr/metasys/software/isodes/) or try to collect some additional data on metabolites not yet measured. To have some insights on what part of the network is already well defined and which one still needs additional measurements, you can try to run `influx_si` with an option `--ln` (as *least norm*) (in addition to `--noopt` option) and examine standard deviation of the fluxes/concentrations in the resulting KVH file. Another possibility is to use parallel labeling experiments (cf. manual section *Parallel experiments*)

Once again, if you could not resolve your problem during these steps, see the next section *Consulting and more*.

CHAPTER

TEN

CONSULTING AND MORE

If you need a help in design, conducting and interpretation of label experiments, you can expose your problem in a brief email to platform MetaToul-Réseaux-Métaboliques (metatoul [at] insa-toulouse [dot] fr) located in Toulouse, France. A dedicated person will take contact with your to detail what can be done to help you and to draw up a quote. For more details about the platform MetaToul, you can visit their web site (english version is available).

If you need help in topics related to mathematics, `influx_si` software itself or custom feature for `influx_si`, you can ask a project creation with Mathematics Cell in Toulouse Biotechnology Institute (sokol (at) insa-toulouse (dot) fr).

You don't have to ask for a consulting for a simple bug submission. A bug submission can be directly made at github

If you wish to be informed about new influx_si releases and/or have discussions with influx_si users, you can subscribe to a dedicated email list [influx_si]

CHAPTER

ELEVEN

LICENSE FOR INFLUX_SI SOFTWARE

```
LICENCE for influx_si software

                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.
```

(continues on next page)

(continued from previous page)

```
  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
```

(continues on next page)

(continued from previous page)

```
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
```

(continues on next page)

(continued from previous page)

```
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.
```

(continues on next page)

(continued from previous page)

```
It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission.  For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
```

(continues on next page)

(continued from previous page)

```
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:
```

(continues on next page)

(continued from previous page)

```
  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.

WARRANTY

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE
PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED
IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS
IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT
NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE
PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

COPYRIGHT 2011-2019 INRA
```

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

c

C13_ftbl, 65

f

ftbl2code, 71
ftbl2mtf, 70
ftbl2netan, 71
ftbl2optR, 72
ftbl2xgmml, 74

k

kvh, 74

t

tools_ssg, 75
txt2ftbl, 69

INDEX

A

aff() (*in module tools_ssg*), 75
aglom() (*in module C13_ftbl*), 65
aglom_loop1() (*in module C13_ftbl*), 65
allprods() (*in module C13_ftbl*), 65
arr2pbm() (*in module tools_ssg*), 75
asort() (*in module tools_ssg*), 75

B

bcumo_decomp() (*in module C13_ftbl*), 65

C

C13_ftbl (*module*), 65
compile() (*in module txt2ftbl*), 69
conv_mid() (*in module C13_ftbl*), 65
copy() (*C13_ftbl.oset method*), 68
cumo_infl() (*in module C13_ftbl*), 65
cumo_iw() (*in module C13_ftbl*), 66
cumo_path() (*in module C13_ftbl*), 66
cumsum() (*in module tools_ssg*), 75

D

dict2kvh() (*in module kvh*), 74
dom_cmp() (*in module C13_ftbl*), 66
dsec2out() (*in module txt2ftbl*), 69
dtstamp() (*in module ftbl2mtf*), 71
dtstamp() (*in module txt2ftbl*), 69

E

enum_path() (*in module C13_ftbl*), 66
escape() (*in module kvh*), 74
expandbit() (*in module tools_ssg*), 75

F

formula2dict() (*in module C13_ftbl*), 66
frag_prod() (*in module C13_ftbl*), 66
ftbl2code (*module*), 71
ftbl2mtf (*module*), 70
ftbl2netan (*module*), 71
ftbl2optR (*module*), 72
ftbl2xgmml (*module*), 74
ftbl_netan() (*in module C13_ftbl*), 66
ftbl_parse() (*in module C13_ftbl*), 67

I

icumo2iiso() (*in module tools_ssg*), 75
infl() (*in module C13_ftbl*), 67
iso2cumo() (*in module C13_ftbl*), 67
iso2emu() (*in module C13_ftbl*), 67
isstr() (*in module tools_ssg*), 75
iterbit() (*in module tools_ssg*), 75
iternumbit() (*in module tools_ssg*), 75
itvl2li() (*in module txt2ftbl*), 69

J

join() (*in module tools_ssg*), 76
joint() (*in module tools_ssg*), 76

K

kvh (*module*), 74
kvh2dict() (*in module kvh*), 74
kvh2obj() (*in module kvh*), 74
kvh2tlist() (*in module kvh*), 74
kvh_get_matrix() (*in module kvh*), 74
kvh_getv_by_k() (*in module kvh*), 74
kvh_read_key() (*in module kvh*), 74
kvh_read_val() (*in module kvh*), 75
kvh_tlist2dict() (*in module kvh*), 75
kvh_tlist2obj() (*in module kvh*), 75

L

label_meas2matrix_vec_dev() (*in module C13_ftbl*), 67
labprods() (*in module C13_ftbl*), 67
list2count() (*in module tools_ssg*), 76
lowtri() (*in module C13_ftbl*), 67

M

mass_meas2matrix_vec_dev() (*in module C13_ftbl*), 67
mat2graph() (*in module C13_ftbl*), 68
mat2pbm() (*in module C13_ftbl*), 68
mecoparse() (*in module C13_ftbl*), 68

mkfunlabli() (*in module C13_ftbl*), 68
ms_frag_gath() (*in module C13_ftbl*), 68

N

netan2Abcumo_spr() (*in module ftbl2code*), 71
netan2R_cumo() (*in module ftbl2code*), 71
netan2R_fl() (*in module ftbl2code*), 71
netan2R_ineq() (*in module ftbl2code*), 71
netan2R_meas() (*in module ftbl2code*), 71
netan2Rinit() (*in module ftbl2code*), 71
ntimes() (*in module C13_ftbl*), 68

O

oset (*class in C13_ftbl*), 68

P

parse_cnstr() (*in module txt2ftbl*), 69
parse_linp() (*in module txt2ftbl*), 70
parse_mflux() (*in module txt2ftbl*), 70
parse_miso() (*in module txt2ftbl*), 70
parse_mmet() (*in module txt2ftbl*), 70
parse_opt() (*in module txt2ftbl*), 70
parse_tvar() (*in module txt2ftbl*), 70
peak_meas2matrix_vec_dev() (*in module C13_ftbl*), 68
proc_kinopt() (*in module C13_ftbl*), 68
proc_label_input() (*in module C13_ftbl*), 68
proc_label_meas() (*in module C13_ftbl*), 68
proc_mass_meas() (*in module C13_ftbl*), 68
proc_peak_meas() (*in module C13_ftbl*), 68
prod() (*in module C13_ftbl*), 69

R

rcumo_sys() (*in module C13_ftbl*), 69
read_table() (*in module tools_ssg*), 76
reverse() (*in module tools_ssg*), 76
rstrbit() (*in module tools_ssg*), 76

S

setbit32() (*in module tools_ssg*), 76
setcharbit() (*in module tools_ssg*), 76
src_ind() (*in module C13_ftbl*), 69
ssign() (*in module tools_ssg*), 76
strbit() (*in module tools_ssg*), 76
strbit2int() (*in module tools_ssg*), 76
strbit32() (*in module tools_ssg*), 76
sumbit() (*in module tools_ssg*), 76

T

t_iso2cumo() (*in module C13_ftbl*), 69
t_iso2m() (*in module C13_ftbl*), 69
t_iso2pos() (*in module C13_ftbl*), 69
tlist2kvh() (*in module kvh*), 75
tools_ssg (*module*), 75
topo_order() (*in module C13_ftbl*), 69
transpose() (*in module C13_ftbl*), 69
trd() (*in module tools_ssg*), 76
try_ext() (*in module txt2ftbl*), 70
tsv2df() (*in module txt2ftbl*), 70
txt2ftbl (*module*), 69
txt_parse() (*in module txt2ftbl*), 70

U

ulong() (*in module tools_ssg*), 76
update() (*C13_ftbl.oset method*), 68

V

valval() (*in module tools_ssg*), 76

W

werr() (*in module C13_ftbl*), 69
wout() (*in module C13_ftbl*), 69
wxlay2py() (*in module tools_ssg*), 77