Flux balance calculations with FASIMU

This manual documents version 2.1.6.

Introduction

FASIMU is an extremely flexible computation environment for different sorts of flux-balance analysis. The flexibility is traded off with simplicity of use. However, people who are acquainted with command-line interfaces will find it easy to use. On the pro side is that practically any algorithmic parameter can be adjusted and it is therefore especially suited for the scientific workflow: difficult problems can be tackled interactively and session protocols can be used to implement new flux-balance methods.

FASIMU is very open in two aspects: The source code is open and written in a high-level language which makes it easy to adapt and implement new functions. The other aspect is that intermediate results are stored in human-readable files which can easily be analyzed once the internal structure of the workflow is understood.

The interactive language used for FASIMU is one that most LINUX and many other UNIX and Windows users are familiar with: the bash-shell. Thus, anything working in bash can easily combined with FASIMU. The typical FASIMU session is in every respect identical to the interactive work in bash. In other words, FASIMU consists of available bash-functions. This concept is more flexible than the interactive languages of Mathematica or MATLAB as the integration with other programs is much more straightforward. For users of LINUX the learning curve is very steep for the syntax of the available commands. Writing algorithmic ideas based on FASIMU is technically writing additional bash functions. Programs in other languages can this way easily be combined with FASIMU; the only prerequisite is an executable program and a defined file exchange.

In flux balance analysis by far the most computational expensive part is the work of the optimizer. Thus, it is not much of an advantage to write the control of the algorithm in a low-level language. This is the reason that the implementation of FASIMU is consequently divided into two parts: the computationally difficult part is handled with an external solver (currently CPLEX, LINDO, lp_solve, or GLPK), the computationally easy but semantically complex part is written in a high-level language which is easiest to understand and modify.

Layer-structure

FASIMU consists of a two-layer structure: the outer layer is FASIMU itself and the lower layer is called FABASE. Where FABASE is focused on a single flux optimization, FASIMU supplies the iteration of flux optimizations with FABASE. FABASE can be used on its own when only single flux computations are required, FASIMU depends on FABASE. It is worth to keep in mind which function belongs to which level. The crucial item of the interface between the two layers is the shell variable “optimization_call”. It contains a FABASE function with full parameters which is called for any of the FASIMU functions which iterate flux calculations.

There is also another respect in which FASIMU sets a layer structure: on the metabolic model itself. When initially called, FASIMU modifies the model by setting a new boundary around the system and the addition of virtual exchange processes for any metabolite of the system across this boundary. The result is another FABASE-compatible model but with a few special properties. Usually, this works unattended in the background but it appears in the computed flux solutions. The duplicated metabolites can be recognized at the "_ex" ending. The virtual exchange processes hold a "_tr" suffix.

The logical structure of a FASIMU session

A FASIMU session should be started in a single empty directory. This directory will later contain the various input, output, and intermediate files of this particular session. For the analysis and proper reference it is advisable not to mix files from different sessions or edit the files in this place. FASIMU will probably work anyway but it is more difficult to assess if it is really doing what the scientist intended to do.

The first step is to copy all files related to the metabolic network into that directory. It is advisable to store the information to one model version in a separate directory. The metabolic model must be supplied in FA-Format, this is a proprietary extended version of CellNetAnalyzer-Format. There are transformation tools available for SBML and a simple Reaction-Scheme-Format.

The second step is the call of "source fasimu" which has several purposes. It transforms to model into a form compatible with FASIMU and makes the FASIMU functions available. Then it starts FABASE which supplies the FASIMU functions and writes some intermediate files which accelerate the preparation time for a single flux optimization call: the main parts of the input into the optimizer program is the prepared (in LPF-Files).

The third step (which also might be interchanged with the second) is the writing of the iteration scheme, a file with the fixed name "simulations.txt". One line defines a single flux computation.

The fourth step is setting the variable "optimization_call" which defines the algorithm of the flux computation. For deeper control on the algorithm the model files and LPF-Files in the working directory can be modified. To do this properly, a deeper understanding of the options FABASE is necessary.

The fifth step is the start of the simulations with "simulate". This function proceeds with all simulations, there are also functions which select a simulation etc.

The sixth step is to review the file "`evaluation.txt`" whether the reported results have been satisfactory. There is one line for each simulation holding a "+" if it obtained the desired result, "-" otherwise.

The seventh step is to view the flux distributions stored for all solutions in a file "`allout.txt`". Single flux distributions are selected with "`allout-select` <name of the simulation>". The I/O-fluxes are at the beginning and should be reviewed first whether the system boundary fluxes are as desired.

The eighth step is advisable if the flux distribution seems reasonable but too complex to check in text form: the visualization in BiNA, Cytoscape, or CellNetAnalyzer. Basically there is the function "`allout2bina`" which creates directories of input files for the flux analysis plugin of BiNA, or the function "`allout2valfiles`"

All these steps are described in greater detail below.

General guidelines of FASIMU's user interface

File contents

All files are whitespace-separated text files except where a two level separation is necessary (file `simulations`) in which case the primary separator is the tab and the secondary separator are space characters.

File extensions

All files to be supplied by the user are text files with a specific name such as `reactions` without an extension. Since these file are modified during the course of computation, their original content is kept in "file-name.original". All files produced by FASIMU being of interest to the user have the extension txt or out. Several files are produced during the workflow and are of lesser interest to the user. They are recognized by their extension:

txt: produced by FASIMU being of interest to the user

fgf: internally used by FASIMU

lpf: parts of the problem formulation

lp: CPLEX-LP format

ltx: LINDO-LP format

par: parameter files of LINDO and lp_solve

in: Input to external programs

out: Output of external programs

Function names

Function names (now) carry dashes and not underline characters (except if it contains the identifier lp_solve which is dictated by the name of the solver)

File parameter

Functions do not accept file parameters, instead the files used have fixed names. If several files for the same purpose are used throughout a session, the session script must rename them.

The respective options are switched on by the existence of these files, e.g. enzymes are considered if the file `enzymes` exists.

File overwrite

As a consequence of fixed file names, the invocation of FASIMU with `source fasimu` changes the input files for FABASE: reactions, metabolites, equilibriums, targetfluxes. The original files are kept in respective files with the extension .original. This concept has historical reasons (FABASE was first, FASIMU has been set on top of it) and will be changed in a future version of FASIMU such that no file will be modified.

The parameter files (`cplex-head.in` and files with the extension par) are only created by `source fabase / fasimu` if not present. Present files will only be overwritten if explicitly requested by the `restore-default-parameters` function. Also, the function `set-timeout` modifies these parameter files.

Installation

FASIMU requires the basic UNIX tools bash and gawk and an optimization solver. Also, a visualization package increases its usefulness.

Bash and gawk

On standard computer systems like UNIX, LINUX and Macintosh, bash and gawk are preinstalled. On MS-Windows systems bash and gawk are usually not installed. They can be conveniently installed by installing Cygwin.

On any standard UNIX system, e.g. LINUX, these are readily available. On UNIX systems, where these tools are not yet available, they can be compiled from the freely available source code, see http://www.gnu.org/software/bash/ and http://www.gnu.org/software/gawk. The version of bash must be at least 4.0 for the function prune-network to work, the rest of FASIMU also works for older versions.

The starting point for the below commands is a running bash shell.

Install FASIMU

Unzip fasimu.zip in a directory contained in the search path. Following UNIX conventions, the directory /usr/local/bin may be used. Alternatively, any directory such as ~/bin/ may be used. In this case the PATH variable should be extended by including the following line into the the start-up file ~/.bashrc of the command line interpreter:

```
export PATH="$PATH:~/bin"
```

Optimization solver

FASIMU was originally developed with the commercial solver ILOG CPLEX, but also the free solvers LP_SOLVE and GLPK can be used, additionally the LINDO API solver. However, some FASIMU functionality (those involving quadractic objectives and constraints) is not available with these alternative solvers.

CPLEX

CPLEX is a commercially available solver (see http://www.ilog.com/products/cplex/). FASIMU is tested with versions 9 and 10 but other versions should also work. There is a highly relevant difference between these two versions.

FASIMU uses the interactive solver of CPLEX which is called “cplex” and must be in the search PATH. If this is not already the case, there are two options to do this. Either, the directory containing the cplex executable can be added to the PATH, e.g.: “

```
export PATH=$PATH:/usr/ilog/cplex101/bin/x86-64_RHEL3.0_3.2/
```

Second option is to add a symbolic link in a directory in the search path, e.g.

```
ln          -s          /usr/ilog/cplex101/bin/x86-64_RHEL3.0_3.2/cplex
/usr/local/bin/cplex
```

LINDO API

FASIMU uses the interactive solver of LINDO API which is called “runlindo” and must be in the search PATH. If this is not already the case, there are two options to do this. Either, the directory containing the runlindo executable can be added to the PATH, e.g.: “

```
export PATH=$PATH:/usr/local/lindoapi/bin/linux64/
```

Second option is to add a symbolic link in a directory in the search path, e.g.

```
ln          -s          /usr/local/lindoapi/bin/linux64/runlindo
/usr/local/bin/runlindo
```

GLPK

GLPK is an open-source free solver (see http://www.gnu.org/software/glpk/) which is available for LINUX and Windows. It is ported to LINUX gentoo and is probably part of other major LINUX distributions. It is ported to cygwin and can therefore even run under MS-Windows."

FASIMU uses its interactive solver which is called “glpsol” and must be in the search PATH. It easiest to move the executable in /usr/local/bin which is the default installation.

lp_solve

lp_solve is another open-source free solver (see http://lpsolve.sourceforge.net/) which is available for LINUX and Windows. It is ported to LINUX gentoo and is probably part of any major LINUX distribution. lp_solve is tested in the version 5.5.

FASIMU uses its interactive solver which is called “lp_solve” which must be in the search PATH. It easiest to move the executable in `/usr/local/bin` which is the default installation. lp_solve requires the filter libxli_CPLEX.so which is included in the lp_solve distribution and should preferentially be placed in `/usr/local/lib`.

Visualization packages

BiNA

Download the latest version from http://sourceforge.net/projects/binafluxmodel/ which includes BiNA and the preinstalled flux analysis plugin. Set the shell variable `BINA_HOME` to the directory, the file `BiNA.jar` is located in.

Cytoscape

Download the latest version from http://www.cytoscape.org and install as documented at the side. Install the plugin FluxViz as documented in http://www.charite.de/sysbio/people/koenig/software/fluxviz/help/index.html#Installation

CellNetAnalyzer

CellNetDesigner is a commercial network analysis tool free for academic use, see http://www.mpi-magdeburg.mpg.de/projects/cna/cna.html. Although graphical output is not the specific purpose of this program, a combined use with FASIMU has some advantages.

The internal data format (the files `reactions` and `metabolites`) is identical in FASIMU and CellNetAnalyzer. Thus, it easy to transform a model developed with CellNetAnalyzer in FASIMU. The feedback of computed flux modes is easy.

FABASE-models, step 1

Getting a model from CellNetAnalyzer

The format of the files `reactions` and `metabolites` equals the internal format of CellNetAnalyzer, formerly FluxAnalyzer. Some parameters in these files have no meaning inside FABASE and are retained for compatibility only. Thus, a model developed with CellNetAnalyzer is a good starting point for a FABASE model. Computed flux modes can easily be transferred to CellNetAnalyzer since the format of the `.val` files is also compatible.

Getting a model from SBML

The easiest way to produce a FABASE model is to get or produce an SBML file and use the program sbml2fa which comes along FASIMU. It is capable to read SBML2.1 to SBML2.4 as it only reads the reaction network which did not change in the subversions. For the metabolites it uses the tags: id, name, compartment, boundaryCondition. The latter three may be missing, after a warning a default is set. For the reactions the tags id, name, and reversible are being used, the latter two might be missing. Additionally, a non-standard way to encode the equilibrium constant will be read: <listOfParameters><parameter id="equilibrium_constant" value="..."></listOfParameters>.

There are some issues where the SBML definition is not compatible with the definition of a FASIMU model. The first issue is that FASIMU encodes the compartment in a suffix to the metabolite identifier and name headed by the underscore character. Therefore underscore characters are not allowed in the compartment identifier (there are no problem in the metabolite names however). sbml2fa removes the underscore characters and also the space characters from the compartment names. If different compartments carry identifiers which differ only by some underscore or space characters (definitely not a sensible practice!) sbml2fa merges them to one. The second issue is closely related. If the metabolite identifier does not end with a _<compartment> already it is added. The same is done independently with metabolite names.

The second issue is that SBML allows space characters in metabolite names, but FASIMU does not (in fact, metabolite name serve as alternative identifiers in the simulation file). Therefore, `sbml2fam` replaces space characters by underscore characters in metabolite names.

Both issues result in the fact that SBMLs generated with FASIMU may have not the same identifiers as the input SBML. But if the following conditions are met, the SBML is FASIMU-proof, meaning SBMLs generated from FASIMU are compatible with the original SBML given to sbml2fa:

1. Compartment identifiers do not contain underscore characters
2. Metabolite identifiers do not contain space characters
3. Metabolite identifiers end with _ and the compartment identifier

4. Metabolite names end with _ and the compartment identifier

Getting a model from a reaction scheme

Another simple way to get a basic FABASE model is to use the program reaction2fa which also comes along FASIMU. It is fairly robust to read any text format reaction scheme which is available in the literature, e.g. the reaction schemes the earlier Palsson group publications uses. The basic format is a whitespace separated file where each line describes a single reaction. It starts with the reaction identifier, than the reaction using numbers (for the stoichiometric coefficients, 1 can be omitted), the identifiers of metabolites, the + sign and reaction arrows as “<=>” and “-->” resp. “<--“ for irreversible reactions. Neither the names of metabolites nor the boundary condition of metabolites can be set this way. This must be done with modifications of the `reactions` and `metabolites` file.

Writing a FABASE model from the scratch

This is easier than it might seem. For a basic FABASE model only two files are necessary, “reactions” and “metabolites” which are plain, white space separated text files. The elements surrounded by the white space (arbitrary sequences of space and tabulator characters) are called token.

`reactions`

The first token is the reaction identifier which must be unique. The next tokens describe the reaction, where stoichiometric coefficients are obligatory and the reaction arrow is strictly =. After the reaction two fixed tokens follow: “|” and “#”. The next two tokens refer to the flux bounds, the first one for the lower, the second one to the higher. “–Inf” is legal for the lower, “+Inf” for the upper bound. To set the reversibility of a reaction, it must be set in the flux bound, there is no indicative reaction arrow.

`metabolites`

The first token is the metabolite identifier which must be unique. The next token is the name which must not contain white space. Typically, space characters are replaced with underscore characters. The next token is arbitrary for the use within FABASE. The next token is the boundary condition, 1 for metabolites which may freely enter or leave the system and 0 for all other metabolites. At this point it must be noted that the boundary condition at this point is only used for direct FABASE calculations (which are not described in this manual) and is ignored for the use in FASIMU: the boundary condition is controlled in the simulations.txt file.

A FABASE model does not notice compartments, however it is recommended to mark the compartment (consisting of letters only) as a suffix separated with a “_” character.

Refining the FABASE Model

The following files are whitespace separated.

Equilibrium data

The first item of the file `equilibriums` is the reaction identifier, the second is the equilibrium constant. For a differing number of products and substrates, the equilibrium constant has a dimension, but the unit is omitted by dividing all concentrations to 1M. This data is used for two purposes: the first denotes the weight for the backward flux in flux minimization, the second one is for the thermodynamic realizability.

FABASE in itself does not use standard Gibbs free energy values to calculate the equilibrium constants, this is (currently) left to user and, by the way, this calculation depends also on the temperature.

Exclude reactions from the cost function

In the different computations based on flux minimization every reaction is used for the objective function. Including reactions in "`fluxmin-excluded`" is an easy way to exclude certain reactions (e. g. those which do not describe a cellular effort).

Concentration ranges

Concentration ranges must be supplied if TR-modes should be computed. To simplify specification of concentration ranges, concentrations may be given for classes of metabolites rather than for metabolites directly. The file `concentration-ranges-classes` assigns a metabolite to a class. The file `concentration-class-ranges` then assigns concentration ranges to classes. Each line must contain 6 tokens: The class identifier, the lower hard bound, the lower soft bound, the set point, the upper soft bound, and the upper hard bound. The hard bounds must never be exceeded. The soft bounds may be violated at the expense of an additional penalty. If a concentration value differs from its set point, a (usually low) penalty is given in the objective function (if this function in switched on). For most applications (when no concentration prediction is required) only the hard bounds are required but the other values must not be omitted in the file.

Exclude reactions from TR-computations: `TR-exclude`

Thermodynamic feasibility can only be assessed if the standard Gibb's free energies of a reaction is known. Therefore FASIMU/FABASE allows to selectively exclude reactions from the TR constraint: put the identifier of the reaction in the file `TR-exclude`. Often, Gibb's energies of reactions are computed as the difference of formation energies of products and reactants. If a formation energy is not known, the Gibb's energy of all reactions involving this metabolite is presumably also not known. Thus, another option is to enter a metabolite identifier in TR-exclude: this means that any reaction involving this metabolite is excluded from the TR criterion. Note that the identifier must include the compartment suffix. This is a consequence that TR resides at the FABASE-level which (as opposed to FASIMU) does not know about compartments. TR-excluded is a white-space separated file, the tokens are accepted no matter if they appear in the same line or in different lines.

Additional names of reactions

This is a white-space separated file: the first token on each line must be a reaction identifier, the rest of the line is regarded as the name of the reaction. The name of the reaction is for instance added at the end of each reaction flux line in allout.txt.

Enzymes and specific weights

In basic flux minimization every (forward) flux is considered to be of equal importance, i.e. the optimization function is the sum of (absolute values of) fluxes. However it is also possible that the fluxes have different weights in the optimization. In the FA-model they are written in the file “enzymes”. The name of the file is derived from the most straightforward use of this function: to implement a cost-minimizing principle of enzymes. The syntax is as follows. It is a white-space separated file. The first token on each line is the identifier of the enzyme. It is no problem if that identifier also appears as a reaction identifier, enzymes and reactions have different name spaces. Then, a coefficient follows, then a reaction identifier. The semantics is that the enzyme requires the cost (the coefficient) to catalyze a unit flux through that specific reaction. The coefficient can also be omitted, in which case it is considered to be 1. The coefficient zero is also allowed, it means that the respective reaction is assigned no cost with respect to this enzyme.

Several important notes for this function: 1. As opposed to other files in the FA-model it is not automatically used, the respective optimization function must require this: e.g. “compute-fluxmin e” or “compute-TR-FBA -F E”. 2. A reaction which is not referred to in “enzymes” is considered to have no cost, it is not minimized. 3. The information is only used for flux minimization computing scripts, other algorithms (biomass maximization or intake minimization= do not use it. 4. The use of this function requires that the reaction identifiers may must not consist only of digits, it is clear that they can be confused with the coefficients.

`fluxfix`

It is possible to set that a particular reaction flux is fixed to a linear combination of other fluxes. Such reactions can also be reformulated but for clarity of the network it might be useful to retain them as separate reactions and fix them separately. An example is that 3% of the oxygen used in the respiration chain is transformed to an superoxid ion.

`fluxfix` is a white-space separated file. The first token is the flux to fix. The rest of the line represents the linear combination, similar to the “enzyme” file: First the coefficient, then the reaction identifier. The coefficient can be omitted in which case it is assumed to be one. The reaction identifier can be omitted for the last token, which means that this value is added as a direct value. Thus, the function can also be used to fix reactions to a preset value.

This feature is used if the file with the name exists. Take care to delete or rename the file if it no longer needed.

`fluxbounds`

This (white-space separated) file sets flux boundaries in conjunction to the values set in `reactions`. The first token of each line is the reaction identifier. If it is the only token of this line the respective flux is fixed to zero. If one token follows the absolute value of the flux must be lower than the value of this token. I two tokens follow they represent the lower and upper bound. “Inf” and “-Inf” are allowed, representing unrestricted fluxes.

In conjunction with other bounds is interpreted in the normal way: the maximum of all applicable lower bounds is the lower bound and the minimum of all upper bounds is the upper bound.

This feature is used if the file with the name exists. Take care to delete or rename the file if it no longer needed.

`setfluxes`

This (white-space separated) file sets fluxes to specified values. The first token of each line is the reaction identifier. If it is the only token of this line the respective flux is fixed to zero. If one token follows the fluxed is fixed to this value. This feature is used if the file with the name exists. Take care to delete or rename the file if it no longer needed.

There are two differences to the file `targetfluxes` described below: (i) it is not modified by functions of FASIMU and (ii) the fluxes are always fixed to the respective values whereas there are not strictly set to the `targetfluxes` when the fitness maximization feature is used.

There is no difference to `fluxfix` when it is only used to fix it to values and not to other fluxes. The `setfluxes` overwrites the value given by `fluxfix`.

FABASE files used to implement FASIMU: `targetfluxes/fluxbounds`

The file "`targetfluxes`" provides a way to fix the flux rate of a reaction (first token) to a value (second token). The file "`fluxbounds`" borders the flux value. An upper limit is given as one token, whereas a range requires two tokens. The simulation objective (see `simulations` below) is implemented with `targetfluxes`, the simulation constraints are implemented with `fluxbounds`.

Since FASIMU automatically generates these files prior invocation of a FABASE function, these options are only available when FABASE is used directly without FASIMU.

FASIMU-compatible Models

For compatibility, conventions of CellNetAnalyzer have been adopted for FABASE and FASIMU: Space characters are not allowed in identifiers. Reaction identifiers should have at least one letter. Every metabolite must have an assigned compartment where the compartment is attached as suffix to the metabolite identifier preceded by underscore. Compartment identifiers consist only of letters and the compartment “ex” is not allowed..

Starting FASIMU, step 2

As mentioned fasimu is invoked by

```
source fasimu
```

There are some options to this call which are indicated by the following identifiers which follow the word fasimu in the command line.

`cplex, cplex9, cplex10, lp_solve, glpk` selects the specific solver. This is necessary if the automatic selection of the solver fails or selects not the desired solver. The automatic recognition of the version of cplex calls `cplex` but that might fail if this is a single license installation and another user is using it at the moment. In this case `cplex9` is used for versions up to 9 and `cplex10` for version from 10 on.

`names_in_allout`. Normally the identifiers of the metabolites are written in the file allout.txt which also. If you prefer the names instead, select this option.

`nomodelprepare` Here, only the FASIMU functions are defined but any action on the model is omitted. This is useful if only a file allout.txt or evaluation.txt from a previous run shall be analyzed or used with the FASIMU functions, for instance for the visualization.

`debug` Some debug messages in the course of the fasimu invocation.

The simulations control file, step 3

The general structure is as follows:

*Each line describes a single simulation

*Each line contains four tab-separated fields: name, objective, constraint, evaluator, comment. Each of the fields may contain several tokens, separated by the space character.

First column: Name

primary key, must be a unique, simulations are identified by its name. may contain space characters

Second field: Objective(s) Targetflux(es)

space separated tokens. A token is either a reaction-ID, a metabolite-ID, or a decompartimentalized metabolite-ID (in this case it is defaulted to the compartment indicated by the variable default_compartment (which is ext by default), the default for outside the system). Each token can be preceded by a real valued coefficient ("1" is omitted, this makes sense only if multiple tokens are used to relatively quantify the target fluxes).Tokens in this field are used without qualifiers (see third column), i.e. targets are always defined as reactions working in forward direction or metabolite export across the system boundary.

Third column: System boundary tokens

Each token is headed by [- + = %], followed by an identifier as above.

	Reaction	Comment	Metabolite	Comment
+	forward direction	definition of forward/ backward direction: see below	product	export of this metabolite across the system boundary is allowed
-	backward direction	definition of forward/ backward direction: see below	substrate	import of this metabolite across the system boundary is allowed
=	not applicable	allowing non-zero flux through reactions is the default	product or substrate	import or export across the system boundary are allowed
%	forbidden	flux through reactions must be zero	not applicable	zero flux through exchange reaction across the system boundary is the default

The rules dictated by the above constraints are combined with flux bounds given in the file `reactions`.

In this column file-content-replacement may avoid repeating long lists several times: When the name of a text file appears in this column, its name is replaced by the file content.

Another syntax is possible in the constraints section:

Syntax	Comment

<identifier> <= <number>	Upper bound for the respective reaction or system exchange
<number> <= <identifier>	Lower bound for the respective reaction or system exchange
<number> <= <identifier> <= <number>	Range for the respective reaction or system exchange

There must a space characters surrounding the <= characters. Instead of <= also < can be written but the result is identical because in the optimization software (and in fact for the underlying theory of optimization) strictly “less than” is not implemented.

Fourth column: evaluator

This is used to ensure a quick overview whether a simulation is successful (indicated by a + sign in the second column of `evaluation.txt`). It can be either zero (this means this simulation is intended to fail), or one (or several) token(s) as above in which case it is checked whether the reaction associated with this token carries a nonzero flux in the flux solution.

A metabolite ID can be

* its model identifier with or without compartment

* metabolite name with or without compartment

If more than one evaluator is given the simulation is considered to be satisfied if *all* evaluators are satisfied. However, the comment in column 3 of `evaluation.txt` indicates which of the tokens are tested successfully.

Fifth column: comment

This is technically not used in FASIMU but it is a good idea to describe the simulation semantically here.

Controlling FABASE, step 4

There is one function for all sorts of flux optimization called `compute-FBA`. The variable “optimization_call” should be set to one of the functions below together with parameters, e.g. `optimization_call=”compute-FBA –T A –F e –s 1 –w 0.0001”`.

The simplest flux minimization: compute-FBA/compute-FBA -F

A flux distribution obeying the flux-balance condition is computed. It minimized the fluxes as a general rule, but there are some variants, controlled by the single parameter. The default is an implementation of {Holzhütter, 2004 #55} where the forward fluxes have the weight 1 and the backward fluxes have the weight equal to the equilibrium constant (given in the file `equilibriums`). If the parameter `–F s` is given the weights of the forward and backwards flux are divided by $(1+K_{eq}^2)^{1/2}$, according to {Holzhütter, 2004 #56}. The parameter setting `–F 1` follows the same idea, both weights are divided by $(1+K_{eq})$. The parameter setting `–F 2` is again very similar, both weights are divided by $(1+K_{eq}^2)$. For the parameter `-F 0`, the backward flux has the same weight as the forward flux, regardless of the value in equilibriums. This is equivalent to the case where no equilibrium data is given at all.

For the two options `–F e` and `–F E` the file enzymes must be available. Here, weights are not given for the fluxes themselves but for the enzyme with respect to a reaction. The general principle is also given in {Holzhütter, 2006 #54}. The difference between e and E is that for e both directions of the reaction receive the same weight where for E the backward flux is multiplied with equilibrium constant, similar to the default.

Allowing restricted fitness: compute-FBA -f

This implements the generalized flux minimization principle {Holzhütter, 2006 #54}. Fluxes must not strictly obey the given objective (in FASIMU, the objective tokens in `simulations`) but are allowed to deviate. Thus, the optimization maximizes the fitness as the first priority and then minimizes the fluxes (as above) as the secondary goal.

The fitness function ranges from zero to one, where one represents maximal fitness. This figure is printed in the output files.

The option -f takes 2 parameters. The first one is the type of the fitness function, ranging from 0 to 3. Default is 0, which is called unadjusted type, because larger fluxes have a higher impact on the fitness. The fitness function is $1-(\Sigma\Delta v_i^2/\Sigma L_i^2)^{1/2}$. The fitness function of type 1 (linearly adjusted type) is $1-(\Sigma(\Delta v_i^2/L_i)/\Sigma L_i)^{1/2}$. The fitness function of type 2 (quadratically adjusted type) is $1-(\Sigma(\Delta v_i^2/L_i^2)/n)^{1/2}$. The fitness function of type 3 (linear type) is $1-(\Sigma(|\Delta v_i|/L_i)/n)$. To apply this type it is necessary to ensure that the directions of the target fluxes are correctly fixed. Currently, for the optimization solvers LINDO, lp_solve and GLPK only type 3 is possible because it requires quadratic terms.

The second parameter is the weight with respect to the weight of the fluxes. Technically, the optimizer does not supply a hierarchic optimization, thus, a large weight has to be given to the fitness score. The default is 10^6.

Minimizing a selected target: compute-FBA -m

This function allows to minimize one or more reaction fluxes, usually applied to the inward transport process of substrates. The parameters to this function are the reaction identifiers.

Maximizing a selected target: compute-FBA -b

This option allows to maximize one or more reaction fluxes, usually applied to the biomass synthesis process or an important product {Edwards, 2001 #15}. The parameters to this function are the reaction identifiers.

Thermodynamic feasibility computations: `compute-FBA -T`

This option switches on the criterion that a positive flux through a reaction must be accompanied by a negative Gibb's free energy the latter being dependent on standard Gibb's energies and ranges of allowable concentrations. The so-called criterion of thermodynamic realizability (TR) is described in detail in {Hoppe, 2007 #57}. This option requires equilibrium and concentration ranges to be set (see above).

TR is a constraint, the flux objective is given by the above options `-F`, `-m`, and `-b`, again with the default of `-F 0`.

The TR computation not only predicts fluxes but also metabolite concentrations, stored in the file `concentrations.txt`. With the option `-d` given to `compute-FBA` a more comprehensive report on concentrations is stored in the file TR-FBA-concentration-doc.txt comprising also the applied ranges. For more refined concentration predictions reasonable soft bounds (switched on with the "-s <weight>" option) and set points ("-w <weight>")

should be supplied for the concentrations. Typical call is “`compute-FBA -T –s 1 –w 0000.1`”.

For the TR criterion a type identifier can be given which controls the way, zero fluxes and thermodynamic potentials are treated. Basically, a non-zero flux and a non-zero potential must have the same sign. (Thermodynamic potential here is the negative Gibb’s energy change divided by RT.) But it is not clearly defined if one of the values is zero. There are two ways: in the relaxed form (a or A) a zero potential is compatible with any flux value, and a zero flux value is compatible with any potential. In the strict form (b or B) a zero potential is only compatible with a zero potential and vice-versa. The differences in the biochemical interpretation will not be covered here. But from the computational side: types b and B are much harder to compute (on more binary variable for each reaction) and the system is much more likely to be infeasible. The uppercase letters refer to an implementation of TR with conditional clauses which is more robust, numerically stable, and accurate in the computation but is only available in the solver CPLEX version 10 and higher. The lowercase letters refer to the so-called bigM implementation which is the standard for other solvers.

A trailing “r” in the type identifier changes the way reaction marked as irreversible are dealt with. By default, there is still the binary variable. For a zero flux, the potential can have either sign. By the setting “r” there is no conditional variable but the sign of the potential is fixed to the sign of the irreversibility constraint. If the flux is non-zero, that is not a difference. However, for a zero flux, the constraint on the potential is still active, where in the default way, there is also a potential with the opposite sign possible. Again, the biochemical implication will not be discussed here. From the computational side, the setting “r” accelerates the computation but also increases the likelihood that the system is unfeasible.

Type	Zero flux	Zero potential	Irreversible reaction	comment
A	q arbitrary	any flux		Conditional clause implementation
B	q=0	v=0		Conditional clause implementation
a	q arbitrary	any flux		bigM implementation
b	$-\epsilon<q<\epsilon$	$-\epsilon<v<\epsilon$		bigM implementation
Ar	q arbitrary	any flux	potential sign fixed	Conditional clause implementation
Br	q=0	v=0	potential sign fixed	Conditional clause implementation
ar	q arbitrary	any flux	potential sign fixed	bigM implementation
br	$-\epsilon<q<\epsilon$	$-\epsilon<v<\epsilon$	potential sign fixed	bigM implementation

There is another parameter worth to mention: -t <seconds> which restricts the computation time of a single optimization.

Perform simulations, step 5

The function call `simulations` performs all simulations described in `simulations.txt` and stores the complete result in allout.txt and a short overview in `evaluation.txt`. To just compute a single simulation identified by its name call `simulations_single "<name>"`

where the quotes can be omitted if the name contains no space characters. To store the solutions requires considerably disk space for large networks, `simulations_noallout` performs the computations but suppresses to write the solutions.

Evaluation file evaluation.txt, step 6

This file is tab separated:

1. Name of the simulation
2. + or - depending if the computed result is in accord with the expectation
3. A comment on the evalution

There are several bash functions which provide convenient views on the result:

- `negeval` gives only lines in evaluation.txt which have not the desired result.
- `inacceval` scans the file for warnings on the inaccuracy of results (switched on by the -c option of compute-FBA).

Analyzing allout.txt, step 7

The file contains all information on the solutions: flux values, concentration values if applicable in a human readable form. Each solution begins with a line with many # characters and the name of the simulation. Then a message on computation and possible warnings follow.

After that the reaction fluxes follow in a tab separated format, the first item is the reaction identifier, the second item is the flux rate, the third is the equilibrium constant used, the fourth is the reaction equation, the fifth are possible annotations of the reactions (given in reaction-names). The reactions are normally written with the identifier of the metabolites. However, it can also be written using the names of the identifier by invoking fasimu with:

```
source fasimu names_in_allout
```

Zero fluxes are omitted throughout, although occasionally fluxes with flux rates close to zero may appear in large networks which are numerically difficult. The first fluxes denoted in the solution are the fluxes across the systems boundary (which are not fluxes in the metabolic reaction systems), identifiable by the trailing “`_tr`” in the reaction identifier and the trailing “_ex” in the metabolite identifier in the reaction equation. Then the regular fluxes follow.

If applicable, i.e. if `compute-FBA -T` was used, concentrations of the metabolites involved in the solution follow, in a tab-separated format: the first item is the metabolite identifier, the second is the concentration value, and the third are the concentration ranges used in the computations which might include soft bounds and set points if they have been used.

Prepare files for the visualization, step 8

FASIMU itself has no visualization capabilities on its own, however it includes plugins for the easy integration of computed flux modes in other packages.

Visualization in Cytoscape

For the visualization in {Killcoyne, 2009 #72} in combination with the flux analysis plugin, the call of `allout2valfiles` transforms the solutions recorded in allout.txt in separate files in a directory `val`.

Visualization in BiNA

For the visualization in BiNA {Küntzer, 2006 #83} in combination with the flux analysis plugin (http:/binafluxmodel.sourceforge.net/), the call of `allout2bina` transforms the solutions recorded in allout.txt in separate files in a directory BiNA. The names of the files are the respective simulation names. There are two parameters to this function. The metabolites (without a compartment identifier) following "-a" are defined to be alias metabolites (see BiNA documentation), metabolites which are not drawn as one node in the graph but as many nodes possible distributed throughout the graph. E.g.

```
allout2bina -a ATP ADP NADH NAD
```

defines these metabolites to be aliased in all occurrences. BiNA is capable to define alias function for each reaction separately but this has to be modified manually, see the description of the BiNA-flux analysis file format.

The parameter -c defines RGB color values for the color of metabolite nodes depending on the compartment. The syntax is as follows: <compartment>[<numR>,<numG>,<numB>]. This sequence must be written without space characters. The respective values range from 0 to 255. In the -c option an arbitrary number of such items may be given. Example:

```
allout2bina -c cyto[245,45,226] mito[66,212,244]
```

Note that if all three numbers are low the color is very dark and the black letters of the metabolite name may not be readable.

A previous directory BiNA will be removed by this function. If the file `allout.txt` contains more than one flux distribution with the same name (this happens if the same simulation is run over again with simulate-single), a underscore character and a number (starting from 0) is appended to the name of the simulation to avoid overwriting.

Visualization in CellNetDesigner

For the visualization in {Klamt, 2007 #74} , the files stored in the `val` directory by `allout2valfiles` can also be directly be used for the visualization in CellNetDesigner by the ReadMode function. Of course, CellNetDesigner requires a ready-made image of the network.

Customization of the external solvers

Large networks in combination with computational expensive constraints such as

- Thermodynamic realizability (requires Boolean variables, thus, computing a flux distribution is now at least a mixed-boolean linear problem, and the number of variables increases: for each metabolite one Boolean and one real-valued variable)
- Set points for concentration values (-w option) turn the problem into a quadratic objective problem

- Use of the fitness maximization, types 0...2 also turns the problem into a quadratic constraint problem
- Real numbers and large numbers as stoichiometry factors (increase the numerical difficulty of the problem, which might cause apparently wrong solutions)

may push the optimization on the brink of their capability. However, all solvers can be customized. Choosing the right parameter switch may put a previously unsolvable problem into reach. You should inspect the file “`solver.out`” which is the output of the solver program stored after each computation call by FASIMU and the instruction manual of the respective solver.

The function

```
set-timeout <seconds>
```

modifies the configuration of the active solver to stop the computation when the given time is reached and output the best intermediate result.

CPLEX

As FASIMU has been developed with CPLEX, a few default settings have been found to ideally harmonize with difficult FBA optimizations. The parameters are stored in the file `cplex-tail.in` which you can modify to change the values of parameters. The file will not be overwritten by a call of “`source fasimu`”; the changes will be in effect for all further computations in this directory. The defaults can be restored with:

```
restore-defaults-cplex
```

LINDO

Lindo parameters are stored in “`lindo.par`”. Again, this file will not be overwritten by a subsequent call of “`source fasimu`”; the defaults can be restored with:

```
restore-defaults-lindo
```

lp_solve

Parameters for lp_solve are stored in “`lp_solve.par`”. This file will not be overwritten by a subsequent call of “`source fasimu`”; the defaults can be restored with:

```
restore-defaults-lp_solve
```

GLPK

GLPK parameters are stored in the bash variable “`glpk`”. To apply different parameters you have to adjust this variable. The variable is not changed by a subsequent call of “`source fasimu`”; the default can be restored with:

```
restore-defaults-glpk
```

Further functions in FABASE

Expression data: the Shlomi algorithm

The function `compute-shlomi` implements the algorithm to predict the active subnetwork depending on expression profile {Shlomi, 2008 #114}. The expression profile is given in the file expressions, which is white-space separated. The first token is the reaction identifier, the second is an expression value, which can be a float number, a binary or a three valued measure. Its interpretation is controlled with the threshold values given -l and -u. For compute-shlomi -l 0.4 -u 0.6 transcript values lower than 0.4 are considered as off, higher than 0.6 as on, and values in between are considered as grey zone and are excluded from the algorithm. Reactions not referred to in expressions are also excluded from the algorithm – this equivalent to the treatment of grey zone values.

Well-formed equilibriums

For thermodynamic computations it is favorable if the standard reaction free energies are compatible to each other, in other words, they obey the Wegscheider condition. Experimental values from different sources obviously do not exactly comply with this rule. Values derived from formation energies by prediction methods {Jankowski, 2008 #69} should do so but this is not always the case due to computational errors. The function `well-formed-equilibriums` attempts to force this condition with changes to the data given in `equilibriums`. It is done with an error minimization which is either linear (-l option) or quadratic (-q only available for solver CPLEX). The latter usually spreads the modification values on more reactions. This algorithm is described in {Hoppe, 2007 #57}. There are two files modifying the algorithm: If a reaction is included in the file `trusted-equilibriums` it will not be modified. If a reaction is included in the file `untrusted-equilbriums` it will primarily be changed. The corrected set of equilibrium constants will be written to `corrected-equilibriums` and a modification report is written to `eq-corr-report.txt`.

Further functions in FASIMU

prune-network

The function checks for each reaction whether a positive or negative flux is possible. This depends on the selection of the optimization protocol (variable $optimization_call) and the contents of the file stdexch.txt which defines the system boundary. See {Hoffmann, 2007 #38} for details on the method. A new network is created in the subdirectory sub. The blocked reactions are removed from the network. If one direction is blocked it is marked as irreversible.

If an error message “bash: declare: -A: invalid option” is shown, please update your bash to a version higher than 4.

make-FVA-simulationsfile

This function prints a simulations file applicable to perform a flux-variability analysis {Reed, 2004 #102; Llaneras, 2007 #87} which should be directed to `simulations`. There are two other elements to be taken care for a meaningful FVA. The first is the file with the predefined name `stdexch.txt` which defines the constraints valid in all defined simulations and it should be used to allow input of substrates and output of waste products and metabolic objectives. The second is the definition of flux boundaries. If there would be no flux restrictions probably many of the fluxes were unbounded in which case the FVA would be meaningless. It is recommended to either restrict all input fluxes or all output fluxes.

FVA-valfiles-chart

This function interprets the result files of a FVA to produce a chart, list of reactions with their respective minimum and maximum. It uses the files in the directory `val` thus, it is necessary to call `allout2valfiles` beforehand. See tutorial for the normal work flow.

check-essentiality

This functions checks the essentiality of reactions with respect to the defined simulations. First, the actual simulations (defined by the `simulations` file and the variable `optimization_call`) are performed. Based on the results a new set of simulations is defined implementing a single knock-out for each reaction contained in the solution of the respective. Note that this is much more efficient than to check every reaction – a reaction can only be essential if it appears in a reference solution.

The result is written concisely in essentiality-report.txt. Each line refers to a simulation in the original set. The lines are tab-separated, the first token is the identifier of the simulation and the second is either

1. the comment `failed` if the original simulation failed
2. the comment no `essentials` if they are no essential reactions, or
3. `essentials:` followed by a space separated list of reaction identifiers referring to the essential reactions.

Other intermediate files can also be inspected:

1. evaluation.txt contains the result of the reference run,
2. simulations_essentiality contains the newly defined simulations,
3. evaluation_essentiality.txt contains their results.

Modify model files

The recommended way to handle your models

I do not recommend to modify the files in the directory in which you are working with fasimu. The program is capable to regenerate intermediate files and for advanced users it makes sense to do so (see below). But to start with I strongly recommend that you keep the original models in separate directories and start fasimu in a fresh directory. Say, you keep and modify your model in a directory `~/model` then proceed like that:

```
cd ~
cp –R model model-compute
cd model-compute
source fasimu
<do the computations>
<copy the result files to a designated directory>
cd ..
rm –r model-compute
```

If you are acquainted with fasimu you will certainly design your own work flow.

Modification functions

If you choose to modify the model files you must not change the files `reactions`, `metabolites`, `equilibriums`, `TR-excluded`, `fluxmin-excluded`, `concentration-ranges-classes` but the files with the extension .original instead. After the modification you call `source fasimu`. You may change the file `simulations` any time you wish. The changes come into effect the next time you call `simulate` or `simulate-single`.

For other files you have to call update functions as follows:

Modified file	Function for the safe integration of the modification
enzymes	update-enzymes
fluxfix	update-fluxfix
concentration-class-ranges	update-concentration-ranges

These update functions are a feature of FABASE. There are other update functions in FABASE but they are not compatible with the work in FASIMU.

Getting more help

There is also a FASIMU tutorial which may by helpful to you.

Functions help page

Some functions have their own help pages available with the “-h” option: compute-FBA, compute-shlomi, well-formed-equilibriums

FASIMU has an online help page: fasimu-help.

Source code

FASIMU is written in the high-level languages bash and gawk so if you are fairly familiar with the syntax of these languages it may be worthwhile to look in the source code. For some functions additional information is available in the comments of the source code. The functions related to FABASE are defined in /usr/local/bin/fabase, related to FASIMU in /usr/local/bin/fasimu (for the recommended installation places).

FASIMU project website

You might wish inspect the project’s website at

`http://www.bioinformatics.org/fasimu`

for the latest updates and other information. There is also a project page at

`http://www.bioinformatics.org/project/?group_id=1004`

leading to bug tracking and a public forum.

License

FASIMU is published under the GNU public license (GPL) which can be viewed in a FASIMU session with

```
fasimu-license
```

There is also a license statement at the beginning of each of the source files. This document is likewise under the GPL.

References

Edwards, J.S., Ibarra, R.U. and Palsson, B.O. (2001) In silico predictions of Escherichia coli metabolic capabilities are consistent with experimental data, *Nat Biotechnol*, **19**, 125--130.

Hoffmann, S., Hoppe, A. and Holzhütter, H.-G. (2007) Pruning genome-scale metabolic models to consistent ad functionem networks, *Genome Inform*, **18**, 308--319.

Holzhütter, H.-G. (2004) The principle of flux minimization and its application to estimate stationary fluxes in metabolic networks, *Eur J Biochem*, **271**, 2905--2922.

Holzhütter, H.-G. (2006) The generalized flux-minimization method and its application to metabolic networks affected by enzyme deficiencies, *Biosystems*, **83**, 98--107.

Holzhütter, S. and Holzhütter, H.-G. (2004) Computational design of reduced metabolic networks, *Chembiochem*, **5**, 1401--1422.

Hoppe, A., Hoffmann, S. and Holzhütter, H.-G. (2007) Including metabolite concentrations into flux balance analysis: thermodynamic realizability as a constraint on flux distributions in metabolic networks, *BMC Syst Biol*, **1**, 23.

Jankowski, M.D., Henry, C.S., Broadbelt, L.J. and Hatzimanikatis, V. (2008) Group contribution method for thermodynamic analysis of complex metabolic networks, *Biophys J*, **95**, 1487--1499.

Killcoyne, S., Carter, G.W., Smith, J. and Boyle, J. (2009) Cytoscape: a community-based framework for network modeling, *Methods Mol Biol*, **563**, 219--239.

Klamt, S., Saez-Rodriguez, J. and Gilles, E.D. (2007) Structural and functional analysis of cellular networks with CellNetAnalyzer, *BMC Syst Biol*, **1**, 2.

Küntzer, J., Blum, T., Gerasch, A., Backes, C., Hildebrandt, A., Kaufmann, M., Kohlbacher, O. and Lenhof, H.P. (2006) BN++ - A Biological Information System, *J Integr Bioinformatics*, **3**, 34.

Llaneras, F. and Picó, J. (2007) An interval approach for dealing with flux distributions and elementary modes activity patterns, *J Theor Biol*, **246**, 290--308.

Reed, J.L. and Palsson, B.Ø. (2004) Genome-scale in silico models of E. coli have multiple equivalent phenotypic states: assessment of correlated reaction subsets that comprise network states, *Genome Res*, **14**, 1797--1805.

Shlomi, T., Cabili, M.N., Herrgård, M.J., Palsson, B.Ø. and Ruppin, E. (2008) Network-based prediction of human tissue-specific metabolism, *Nat Biotechnol*, **26**, 1003--1010.