

Documentation:
<http://opencobra.github.io/cobratoolbox>

```

> Checking if git is installed ... Done.
> Checking if the repository is tracked using git ... Done.
> Checking if curl is installed ... Done.
> Checking if remote can be reached ... Done.
> Initializing and updating submodules ... Done.
> Adding all the files of The COBRA Toolbox ... Done.
> Define CB map output... set to svg.
> Retrieving models ... Done.
> TranslateSBML is installed and working properly.
> Configuring solver environment variables ...
  - [----] ILOG_CPLEX_PATH : --> set this path manually after installing the solver ( see instructions )
  - [*----] GUROBI_PATH: /opt/gurobi702/linux64/matlab
  - [----] TOMLAB_PATH : --> set this path manually after installing the solver ( see instructions )
  - [----] MOSEK_PATH : --> set this path manually after installing the solver ( see instructions )
Done.
> Checking available solvers and solver interfaces ... Done.
> Setting default solvers ... Done.
> Saving the MATLAB path ... Done.
  - The MATLAB path was saved as ~/pathdef.m.

> Summary of available solvers and solver interfaces

```

Support	LP	MILP	QP	MIQP	NLP	

cplex_direct	full			0	0	0
dqqMinos	full			1	-	-
glpk	full			1	1	-
gurobi	full			1	1	1
ibm_cplex	full			0	0	-
matlab	full			1	-	1
mosek	full			0	0	-
pdco	full			1	-	1
quadMinos	full			1	-	-
tomlab_cplex	full			0	0	0
qpng	experimental			-	-	1
tomlab_snopt	experimental			-	-	-
gurobi_mex	legacy			0	0	0
lindo_old	legacy			0	-	-
lindo_legacy	legacy			0	-	-
lp_solve	legacy			1	-	-
opti	legacy			0	0	0

Total	-			7	2	3
					1	2

+ Legend: - = not applicable, 0 = solver not compatible or not installed, 1 = solver installed.

```

> You can solve LP problems using: 'dqqMinos' - 'glpk' - 'gurobi' - 'matlab' - 'pdco' - 'quadMinos' - 'qpng'
> You can solve MILP problems using: 'glpk' - 'gurobi'
> You can solve QP problems using: 'gurobi' - 'pdco' - 'qpng'
> You can solve MIQP problems using: 'gurobi'
> You can solve NLP problems using: 'matlab' - 'quadMinos'

> Checking for available updates ...
> The COBRA Toolbox is up-to-date.

```

```
warning off MATLAB:subscripting:noSubscriptsSpecified
```

PROCEDURE

Generate a network

A constraint-based metabolic model contains the stoichiometric matrix with reactions and metabolites [1].

S is stoichiometric representation of metabolic networks corresponding to the reactions in the biochemical pathway. In an each column of the S is a biochemical reaction and in each row is a precise metabolite. There is a stoichiometric coefficient of zero, which means that metabolite not participate in that distinct reaction. The coefficient also can be positive when the appropriate metabolite is produced, or negative for every metabolite depleted [1].

```
ReactionFormulas = {'glc-D[e] -> glc-D[c]',...
  'glc-D[c] + atp[c] -> H[c] + adp[c] + g6p[c]',...
  'g6p[c] <=> f6p[c]',...
  'atp[c] + f6p[c] -> H[c] + adp[c] + fdp[c]',...
  'fdp[c] + h2o[c] -> f6p[c] + pi[c]',...
  'fdp[c] -> g3p[c] + dhap[c]',...
  'dhap[c] -> g3p[c]'};
ReactionNames = {'GLCt1', 'HEX1', 'PGI', 'PFK', 'FBP', 'FBA', 'TPI'};
lowerbounds = [-20, 0, -20, 0, 0, -20, -20];
upperbounds = [20, 20, 20, 20, 20, 20, 20];
model = createModel(ReactionNames, ReactionNames, ReactionFormulas, ...
  'lowerBoundList', lowerbounds, 'upperBoundList', upperbounds);
```

Warning: Metabolite glc-D[e] not in model - added to the model

Warning: Metabolite glc-D[c] not in model - added to the model

GLCt1 glc-D[e] <=> glc-D[c]

Warning: Metabolite atp[c] not in model - added to the model

Warning: Metabolite H[c] not in model - added to the model

Warning: Metabolite adp[c] not in model - added to the model

Warning: Metabolite g6p[c] not in model - added to the model

HEX1 glc-D[c] + atp[c] -> H[c] + adp[c] + g6p[c]

Warning: Metabolite f6p[c] not in model - added to the model

PGI g6p[c] <=> f6p[c]

Warning: Metabolite fdp[c] not in model - added to the model

PFK atp[c] + f6p[c] -> H[c] + adp[c] + fdp[c]

Warning: Metabolite h2o[c] not in model - added to the model

Warning: Metabolite pi[c] not in model - added to the model

FBP fdp[c] + h2o[c] -> f6p[c] + pi[c]

Warning: Metabolite g3p[c] not in model - added to the model

Warning: Metabolite dhap[c] not in model - added to the model

FBA fdp[c] <=> g3p[c] + dhap[c]

TPI dhap[c] <=> g3p[c]

We can now have a look at the different model fields created. The stoichiometry is stored in the S field of the model, which was described above. Since this is commonly a sparse matrix (i.e. it does contain a lot of zeros, to display it it is useful to use the full representation)

```
full(model.S)
```

ans =

-1	0	0	0	0	0	0
1	-1	0	0	0	0	0
0	-1	0	-1	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	0	0
0	1	-1	0	0	0	0
0	0	1	-1	1	0	0

0	0	0	1	-1	-1	0
0	0	0	0	-1	0	0
0	0	0	0	1	0	0

Some descriptive fields always present are `model.mets` and `model.rxns` which represent the metabolites and the reactions respectively.

```
model.mets
```

```
ans =
'glc-D[e]'
'glc-D[c]'
'atp[c]'
'H[c]'
'adp[c]'
'g6p[c]'
'f6p[c]'
'fdp[c]'
'h2o[c]'
'pi[c]'
'g3p[c]'
'dhap[c]'
```

```
model.rxns
```

```
ans =
'GLCt1'
'HEX1'
'PGI'
'PFK'
'FBP'
'FBA'
'TPI'
```

Fields in the COBRA model are commonly column vectors, which can be an important detail when writing functions manipulating these fields.

There are a few more fields present in each COBRA model:

`model.lb`, indicating the lower bounds of each reaction, and `model.ub` indicating the upper bound of a reaction.

```
[{'Reaction ID', 'Lower Bound', 'Upper Bound'};... % this displays an array with
model.rxns, num2cell(model.lb), num2cell(model.ub)] % reaction names and flux bounds.
```

```
ans =
'Reaction ID'    'Lower Bound'    'Upper Bound'
'GLCt1'          [          -20]  [           20]
'HEX1'           [           0]  [           20]
'PGI'            [          -20]  [           20]
'PFK'            [           0]  [           20]
'FBP'            [           0]  [           20]
'FBA'            [          -20]  [           20]
'TPI'            [          -20]  [           20]
```

```
printFluxBounds(model); %This is a convenience function which does
```

Reaction ID	Lower Bound	Upper Bound
GLCt1	-20.000	20.000
HEX1	0.000	20.000
PGI	-20.000	20.000
PFK	0.000	20.000
FBP	0.000	20.000
FBA	-20.000	20.000
TPI	-20.000	20.000

%pretty much the same as the line above

Before we start to modify the model, it might be useful to store some of the current properties of the model

```
mets_length = length(model.mets);
rxns_length = length(model.rxns);
```

Creating, adding and handling reactions

If we want to add a reaction to the model or modify an existing reaction we are using function `addReaction`.

We will add some more reactions from glycolysis.

- The formula approach

```
model = addReaction(model, 'GAPDH',...
    'reactionFormula', 'g3p[c] + NAD[c] + 2 pi[c] -> NADH[c] + H[c] + 13bpg[c]');
```

Warning: Metabolite NAD[c] not in model - added to the model
Warning: Metabolite NADH[c] not in model - added to the model
Warning: Metabolite 13bpg[c] not in model - added to the model
GAPDH 2 pi[c] + g3p[c] + NAD[c] -> H[c] + NADH[c] + 13bpg[c]

```
model = addReaction(model, 'PGK',...
    'reactionFormula', '13bpg[c] + adp[c] -> atp[c] + 3pg[c]');
```

Warning: Metabolite 3pg[c] not in model - added to the model
PGK adp[c] + 13bpg[c] -> atp[c] + 3pg[c]

```
model = addReaction(model, 'PGM', 'reactionFormula', '3pg[c] <=> 2pg[c]');
```

Warning: Metabolite 2pg[c] not in model - added to the model
PGM 3pg[c] <=> 2pg[c]

Display of stoichiometric matrix after adding reactions. Note the enlarge link when you move your mouse over the output to display the full matrix:

```
full(model.S)
```

ans =

```
-1    0    0    0    0    0    0    0    0    0
 1   -1    0    0    0    0    0    0    0    0
```

0	-1	0	-1	0	0	0	0	1	0
0	1	0	1	0	0	0	1	0	0
0	1	0	1	0	0	0	0	-1	0
0	1	-1	0	0	0	0	0	0	0
0	0	1	-1	1	0	0	0	0	0
0	0	0	1	-1	-1	0	0	0	0
0	0	0	0	-1	0	0	0	0	0
0	0	0	0	1	0	0	-2	0	0

```
% one extra column is added(for added reaction) and 5 new
% rows(for NADH, NAD, 13bpg, 2pg and 3pg metabolites)
```

The following functions are used when we want to search reactions sequence in the model and change the order of the selected reaction.

```
rxnID = findRxnIDs(model, model.rxns);
model = moveRxn(model, 8, 1);
```

While the latter does not modify the structure as such it can help in keeping a model tidy.

- The list approach

The addReaction function has ability to recognize duplicate reactions when an order of metabolites and an abbreviation of the reaction are different.

```
model = addReaction(model, 'GAPDH2',...
    'metaboliteList', {'g3p[c]', 'NAD[c]', 'pi[c]', '13bpg[c]', 'NADH[c]', 'H[c]' },...
    'stoichCoeffList', [-1; -1; -2; 1; 1; 1], 'reversible',false);
```

Warning: Model already has the same reaction you tried to add: GAPDH

Since the second call should not have added anything we will check if the number of the reaction increased by the three reactions we added (and not by the one duplicated) and the number of metabolites was incremented by five (13bpg, NAD, NADH, 23bpg and 2pg).

```
assert(length(model.rxns) == rxns_length + 3);
assert(length(model.mets) == mets_length + 5);
```

Adding Exchange, Sink and Demand reactions

Specific type of reactions in the constraint-based models are reactions that are using and recycling accumulated metabolites, or producing required metabolites in the model.

1. *Exchange reactions* - Reactions added to the model to move metabolites across the created *in silico* compartments. Those compartments represent intra- and intercellular membranes.
2. *Sink reactions* - The metabolites, produced in reactions that are outside of an ambit of the system or in unknown reactions, are supplied to the network with reversible sink reactions.
3. *Demand reactions* - Irreversible reactions added to the model to consume metabolites that are deposited in the system.

There are two ways to implement that kind of reactions:

1. Use addReaction with the documented function call:

```
model = addReaction(model, 'EX_glc-D[e]', 'metaboliteList', {'glc-D[e]'} ,...  
                    'stoichCoeffList', [-1]);
```

```
EX_glc-D[e] glc-D[e] <=>
```

In the bigger networks we can find our exchange reactions with the following functions:

```
[selExc, selUpt] = findExcRxns(model, 0, 1) % determines whether a reaction is a
```

```
selExc =  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
1
```

```
selUpt =  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0
```

```
% general exchange reaction and whether  
% its an uptake.
```

2. Use a utility function to create a particular reaction type: addExchangeRxn, addSinkReactions, addDemandReaction.

```
model = addExchangeRxn(model, {'glc-D[e]', 'glc-D[c]'})
```

Warning: Reaction with the same name already exists in the model, updating the reaction

```
EX_glc-D[e] glc-D[e] <=>  
EX_glc-D[c] glc-D[c] <=>
```

```
model =  
    rxns: {12×1 cell}  
        S: [17×12 double]  
        lb: [12×1 double]  
        ub: [12×1 double]  
        c: [12×1 double]  
    mets: {17×1 cell}  
        b: [17×1 double]
```

```
rules: {12×1 cell}
genes: {0×1 cell}
osense: -1
csense: [17×1 char]
rxnGeneMat: [12×0 double]
rxnNames: {12×1 cell}
subSystems: {12×1 cell}
metNames: {17×1 cell}
grRules: {12×1 cell}
```

```
model = addSinkReactions(model, {'13bpg[c]', 'NAD[c]'})
```

```
sink_13bpg[c] 13bpg[c] <=>
sink_NAD[c] NAD[c] <=>
model =
```

```
rxns: {14×1 cell}
S: [17×14 double]
lb: [14×1 double]
ub: [14×1 double]
c: [14×1 double]
mets: {17×1 cell}
b: [17×1 double]
rules: {14×1 cell}
genes: {0×1 cell}
osense: -1
csense: [17×1 char]
rxnGeneMat: [14×0 double]
rxnNames: {14×1 cell}
subSystems: {14×1 cell}
metNames: {17×1 cell}
grRules: {14×1 cell}
```

```
model = addDemandReaction(model, {'dhap[c]', 'g3p[c]'})
```

```
DM_dhap[c] dhap[c] ->
DM_g3p[c] g3p[c] ->
```

```
model =
rxns: {16×1 cell}
S: [17×16 double]
lb: [16×1 double]
ub: [16×1 double]
c: [16×1 double]
mets: {17×1 cell}
b: [17×1 double]
rules: {16×1 cell}
genes: {0×1 cell}
osense: -1
csense: [17×1 char]
rxnGeneMat: [16×0 double]
rxnNames: {16×1 cell}
subSystems: {16×1 cell}
metNames: {17×1 cell}
grRules: {16×1 cell}
```

Setting ratio between the reactions and changing reactions boundary

It is important to emphasize that previous knowledge base informations should be taken into account. Most of them could disrupt future analysis of the model.

For instance, if it is familiar that flux through one reaction is X times the flux through another reaction, it is recommended to specify that in your model.

E.g. $1 \text{ } v \text{ EX_glc} - D[c] = 2 \text{ } v \text{ EX_glc} - D[e]$

```
model = addRatioReaction (model, {'EX_glc-D[c]', 'EX_glc-D[e]'}, [1; 2]);
```

Altering Reaction bounds

In order to respect the transport and exchange potential of a particular metabolite, or to resemble the different conditions in the model, we frequently need to set appropriate limits of the reactions.

```
model = changeRxnBounds(model, 'EX_glc-D[e]', -18.5, 'l');
```

Modifying Reactions

The addReaction function also is a good choice when modifying reactions. By supplying a new stoichiometry, the old will be overwritten. For example further up, we added awrong stoichiometry for the GAP-Dehydrogenase with a phosphate coefficient of 2. (easily visiple by printing the reaction)

```
printRxnFormula(model, 'rxnAbbrList', 'GAPDH');
```

```
GAPDH 2 pi[c] + g3p[c] + NAD[c] -> H[c] + NADH[c] + 13bpg[c]
```

We can correct this by simply calling ddReaction again with the corrected stoichiometry. In essence parts which are not provided are taken from the old reaction, and only the new ones overwrite the existing data

```
model = addReaction(model, 'GAPDH',...  
    'metaboliteList', {'g3p[c]', 'NAD[c]', 'pi[c]', '13bpg[c]', 'NADH[c]', 'H[c]'},...  
    'stoichCoeffList', [-1; -1; -1; 1; 1; 1]);
```

Warning: Reaction with the same name already exists in the model, updating the reaction

```
GAPDH pi[c] + g3p[c] + NAD[c] -> H[c] + NADH[c] + 13bpg[c]
```

We might also want to add a gene rule to the reaction. this can either be done using

```
model = changeGeneAssociation(model, 'GAPDH', 'G1 and G2');
```

```
New gene G1 added to model  
New gene G2 added to model
```

```
printRxnFormula(model, 'rxnAbbrList', {'GAPDH'}, 'gprFlag', true);
```

```
GAPDH pi[c] + g3p[c] + NAD[c] -> H[c] + NADH[c] + 13bpg[c] G1 and G2
```

The other option to achieve this is to use `addReaction` and the `geneRule` parameter

```
model = addReaction(model, 'PGK', 'geneRule', 'G2 or G3', 'printLevel', 0);
```

Warning: Reaction with the same name already exists in the model, updating the reaction
New gene G3 added to model

```
printRxnFormula(model, 'gprFlag', true);
```

```
GAPDH pi[c] + g3p[c] + NAD[c] -> H[c] + NADH[c] + 13bpg[c] G1 and G2
GLCt1 glc-D[e] <=> glc-D[c]
HEX1 glc-D[c] + atp[c] -> H[c] + adp[c] + g6p[c]
PGI g6p[c] <=> f6p[c]
PFK atp[c] + f6p[c] -> H[c] + adp[c] + fdp[c]
FBP fdp[c] + h2o[c] -> f6p[c] + pi[c]
FBA fdp[c] <=> g3p[c] + dhap[c]
TPI dhap[c] <=> g3p[c]
PGK adp[c] + 13bpg[c] -> atp[c] + 3pg[c] G2 or G3
PGM 3pg[c] <=> 2pg[c]
EX_glc-D[e] glc-D[e] <=> 2 Ratio_EX_glc-D[c]_EX_glc-D[e]
EX_glc-D[c] glc-D[c] + Ratio_EX_glc-D[c]_EX_glc-D[e] <=>
sink_13bpg[c] 13bpg[c] <=>
sink_NAD[c] NAD[c] <=>
DM_dhap[c] dhap[c] ->
DM_g3p[c] g3p[c] ->
```

Remove reactions and metabolites

In order to detach reactions from the model, the following function has been used:

```
model = removeRxn(model, {'EX_glc-D[c]', 'EX_glc-D[e]', 'sink_13bpg[c]', ...
                          'sink_NAD[c]', 'DM_dhap[c]', 'DM_g3p[c]'});

assert(rxn_length + 3 == length(model.rxn));
% The reaction length has been reevaluated
```

Remove metabolites

```
model = removeMetabolites(model, {'3pg[c]', '2pg[c]'}, false);
```

For instance, in previous code the many metabolites from 'GAPDH' were deleted, but the reaction is still present in the model (since there are more metabolites left). The false indicates, that empty reactions should not be removed.

To delete metabolites and reactions with zero rows and columns, the following function can be used:

```
model = removeTrivialStoichiometry(model)
```

```
model =
    rxns: {9x1 cell}
         S: [15x9 double]
         lb: [9x1 double]
         ub: [9x1 double]
         c: [9x1 double]
    mets: {15x1 cell}
         b: [15x1 double]
    rules: {9x1 cell}
```

```

    genes: {3x1 cell}
    osense: -1
    csense: [15x1 char]
    rxnGeneMat: [9x3 double]
    rxnNames: {9x1 cell}
    subSystems: {9x1 cell}
    metNames: {15x1 cell}
    grRules: {9x1 cell}
    note: 'EX_glc-D[c] and EX_glc-D[e] are set to have a ratio of 1:2.'

```

```
model = removeRxn(model, {'GAPDH', 'PGK'});
```

Search for duplicates and comparison of two models

Since genome-scale metabolic models are expanding every day [2], the need for comparison and merge of them is also spreading.

The elementary functions for the model manipulation, besides main actions, simultaneously perform the structural analysis and comparison (e.g. `addReaction`). Likewise, there are additional functions that are only dealing with analysing similarities and differences within and between the models.

- Checking for reaction duplicates by reaction abbreviation, by using method 'S' that will not detect reverse reactions, and method 'FR' that will neglect reactions direction:

```
[model, removedRxn, rxnRelationship] = checkDuplicateRxn(model, 'S', 1, 1);
```

```

Checking for reaction duplicates by stoichiometry ...
no duplicates found.

```

Adding duplicate reaction to the model:

```

model = addReaction(model, 'GLCt1_duplicate_reverse', ...
    'metaboliteList', {'glc-D[e]', 'glc-D[c]'}, ...
    'stoichCoeffList', [1 -1], 'lowerBound', 0, 'upperBound', 20, 'checkDuplicate', 0);

```

```
GLCt1_duplicate_reverse glc-D[c] -> glc-D[e]
```

```
fprintf('>> Detecting duplicates using S method\n');
```

```
>> Detecting duplicates using S method
```

```

method = 'S';
%will not be removed as does not detect reverse reaction
[model, removedRxn, rxnRelationship] = checkDuplicateRxn(model, method, 1, 1);

```

```

Checking for reaction duplicates by stoichiometry ...
no duplicates found.

```

```

assert(rxnLength + 1 == length(model.rxnLength));
% The reaction length has been reevaluated

fprintf('>> Detecting duplicates with using FR method\n');

```

```
>> Detecting duplicates with using FR method
```

```
method = 'FR';  
%will be removed as detects reverse reaction  
[model, removedRxn, rxnRelationship] = checkDuplicateRxn(model, method, 1, 1);
```

```
Checking for reaction duplicates by stoichiometry (up to orientation) ...
```

```
Keep: GLCt1_glc-D[e] <=> glc-D[c]
```

```
Duplicate: GLCt1_duplicate_reverse_glc-D[c] -> glc-D[e]
```

```
assert(rxns_length == length(model.rxns));
```

- Function `checkCobraModelUnique` marks the reactions and metabolites that are not unique in the model.

```
model = checkCobraModelUnique(model, false)
```

```
model =  
    rxns: {7×1 cell}  
        S: [12×7 double]  
        lb: [7×1 double]  
        ub: [7×1 double]  
        c: [7×1 double]  
    mets: {12×1 cell}  
        b: [12×1 double]  
    rules: {7×1 cell}  
    genes: {3×1 cell}  
    osense: -1  
    csense: [12×1 char]  
    rxnGeneMat: [7×3 double]  
    rxnNames: {7×1 cell}  
    subSystems: {7×1 cell}  
    metNames: {12×1 cell}  
    grRules: {7×1 cell}  
    note: 'EX_glc-D[c] and EX_glc-D[e] are set to have a ratio of 1:2.'
```

Changing the model's objective

Simulating different conditions in the model is often necessary in favor of performing calculations that investigate a specific objective. One of the elementary objectives is optimal growth [3]. Model can be modified to get different conditions with changing the model objective:

```
modelNew = changeObjective(model, 'GLCt1', 0.5);  
  
% multiple rxns, default coefficient (1)  
modelNew = changeObjective(model, {'PGI'; 'PFK'; 'FBP'});
```

The direction of reactions

When reaction is reversible and we want to remove it, it is necessary to first change the reverse sign and then remove reaction afterwards.

```
fprintf('>> Converting to Irreversible\n');
```

>> Converting to Irreversible

```
load('testModelManipulation.mat', 'model', 'modelIrrev');  
[testModelIrrev, matchRev, rev2irrev, irrev2rev] = convertToIrreversible(model);
```

Following function is comparing the differences and similarities between two models:

```
assert(isSameCobraModel(modelIrrev, testModelIrrev));
```

REFERENCES

1. Orth, J. D. et al. What is flux balance analysis? *Nat. Biotechnol.*, 28(3), 245–248 (2010).
2. Feist, A. M., Palsson, B. The Growing Scope of Applications of Genome-scale Metabolic Reconstructions: the case of *E. coli*. *Nature Biotechnology*, 26(6), 659–667 (2008).
3. Feist, A. M., Palsson, B. O. The Biomass Objective Function. *Current Opinion in Microbiology*, 13(3), 344–349 (2010).