Testing basic properties of a metabolic model (aka sanity checks)

Author(s): Ines Thiele, Ronan M. T. Fleming, Systems Biochemistry Group, LCSB, University of Luxembourg.

Reviewer(s): Almut Heinken, LCSB, University of Luxembourg.

In this tutorial, we show how test for basic modeling properties of a metabolic model. The tutorial was developed during the construction of the generic human metabolic model, Recon 3D [1] and can be applied to Recon 3D derived condition- and cell-type specific models, to the previous generic human reconstruction, Recon2, as well as other metabolic models.

Content:

The tests include:

- leak test
- production of protons from nothing as well as from water, and/or oxygen alone
- production of matter when atp hydrolysis reaction is allowed to work but all uptakes are closed
- production of too much ATP from glucose under aerobic condition
- duplicated reactions
- empty colmuns in the model.rxnGeneMat
- the single gene deletion analysis runs smoothly
- ATP yield from different carbon sources
- metabolic objective functions
- flux consistency
- demand reactions with negative lower bound (should not occur based on definition of demand reactions)
- consistency of model.rev, which defines reaction reversibility, and the set values for the lower bounds on reactions.

All results are stored in a table ('TableChecks').

EQUIPMENT SETUP

If necessary, initialize the cobra toolbox:

```
initCobraToolbox(false) % false, as we don't want to update
```

For solving linear programming problems in FBA analysis, certain solvers are required:

```
changeCobraSolver ('glpk', 'all', 1);
%Uncomment if you don't want to use glpk.
%changeCobraSolver ('ibm_cplex', 'all', 1);
%changeCobraSolver ('tomlab_cplex', 'all', 1);
```

This tutorial can be run with `'glpk'` package as linear programming solver, which does not require additional instalation and configuration. However, for the analysis of large models, such as Recon 3, it is not recommended to use `'glpk'` but rather industrial-strength solvers, such as the `'gurobi'` package.

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

PROCEDURE

Before proceeding with the simulations, the path for the model needs to be set up. In this tutorial, the used model is the generic model of human metabolism, Recon 3 [1]. If Recon 3 is not available, please use Recon 2.

```
modelFileName = 'Recon2.0model.mat'; %Replace if you want to load Recon3D
modelDirectory = getDistributedModelFolder(modelFileName); %Look up the
folder for the distributed Models.
modelFileName= [modelDirectory filesep modelFileName]; % Get the full path.
Necessary to be sure, that the right model is loaded
model = readCbModel(modelFileName);
```

Model Harmonization

Replace reaction abbreviation for the ATP hydrolysis (DM_atp_c_) and Biomass reaction used differently in various models.

```
model.rxns(find(ismember(model.rxns,'ATPM')))={'DM_atp_c_'};
model.rxns(find(ismember(model.rxns,'ATPhyd')))={'DM_atp_c_'};
model.rxns(find(ismember(model.rxns,'DM_atp(c)')))={'DM_atp_c_'};
model.rxns(find(ismember(model.rxns,'EX_biomass_reaction')))={'biomass_reacti
on'};
model.rxns(find(ismember(model.rxns,'EX_biomass_maintenance')))={'biomass_mai
ntenance'};
model.rxns(find(ismember(model.rxns,'EX_biomass_maintenance_noTrTr')))={'biom
ass_maintenance_noTrTr'};
```

Set lower bound of the biomass reaction to 0.

```
model.lb(find(ismember(model.rxns,'biomass_reaction')))=0;
model.lb(find(ismember(model.rxns,'biomass_maintenance_noTrTr')))=0;
model.lb(find(ismember(model.rxns,'biomass_maintenance')))=0;
```

Harmonize different use of brackets.

```
model.rxns = regexprep(model.rxns,'\(','\[');
model.rxns = regexprep(model.rxns,'\)','\]');
model.rxns = regexprep(model.rxns,'Ex_','EX_');
model.rxns = regexprep(model.rxns,'Sink_','sink_');
model.rxns = regexprep(model.rxns,'-','_');
```

Define some parameters that we will need.

```
cnt = 1;
tol = 1e-6;
```

Define the closed model. Here, we will set to zero the lower bounds of all reactions that represent exchange and siphon ('sink') reactions, or that contain only one entry in the column of the S matrix. The upper bound of those reactions is set to 1000 (i.e., infinity). Note that this overwrites any constraints on those reactions that may be present in a condition- and cell-type specific model.

```
modelClosed = model;
modelexchanges1 = strmatch('Ex_',modelClosed.rxns);
modelexchanges4 = strmatch('EX_',modelClosed.rxns);
modelexchanges2 = strmatch('DM_',modelClosed.rxns);
modelexchanges3 = strmatch('sink_',modelClosed.rxns);
selExc = (find( full((sum(abs(modelClosed.S)==1,1) ==1) &
(sum(modelClosed.S~=0) == 1))))';

modelexchanges =
unique([modelexchanges1;modelexchanges2;modelexchanges3;modelexchanges4;selEx
c]);
modelClosed.lb(find(ismember(modelClosed.rxns,modelClosed.rxns(modelexchanges
))))=0;
modelClosed.ub(find(ismember(modelClosed.rxns,modelClosed.rxns(modelexchanges
))))=1000;
modelClosedOri = modelClosed;
```

Start with tests.

Perform leak test, i.e., whether the closed model can produce any exchanged metabolite, as defined in the model, from nothing.

```
modelClosed = modelClosedOri;
[LeakRxns,modelTested,LeakRxnsFluxVector] =
fastLeakTest(modelClosed,modelClosed.rxns(selExc),'false');
TableChecks{cnt,1} = 'fastLeakTest 1';
if length(LeakRxns)>0
    warning('model leaks metabolites!')
    TableChecks{cnt,2} = 'Model leaks metabolites!';
else
    TableChecks{cnt,2} = 'Leak free!';
end
cnt = cnt + 1;
```

Test if something leaks when demand reactions for each metabolite in the model are added. Note that this step is time consuming.

```
modelClosed = modelClosedOri;
[LeakRxnsDM,modelTestedDM,LeakRxnsFluxVectorDM] =
fastLeakTest(modelClosed,modelClosed.rxns(selExc),'true');

TableChecks{cnt,1} = 'fastLeakTest 2 - add demand reactions for each
metabolite in the model';
if length(LeakRxnsDM)>0
```

```
    TableChecks{cnt,2} = 'Model leaks metabolites when demand reactions are
added!';
else
    TableChecks{cnt,2} = 'Leak free when demand reactions are added!';
end
cnt = cnt + 1;
```

Test if the model produces energy from water!

```
modelClosed = modelClosedOri;
modelClosedATP = changeObjective(modelClosed,'DM_atp_c_');
modelClosedATP = changeRxnBounds(modelClosedATP,'DM_atp_c_',0,'l');
modelClosedATP = changeRxnBounds(modelClosedATP,'EX_h2o[e]',-1,'l');
FBA3=optimizeCbModel(modelClosedATP);
TableChecks{cnt,1} = 'Exchanges, sinks, and demands have  lb = 0, except
h2o';
if abs(FBA3.f) > 1e-6
    TableChecks{cnt,2} = 'model produces energy from water!';
else
    TableChecks{cnt,2} = 'model DOES NOT produce energy from water!';
end
cnt = cnt + 1;
```

Test if the model produces energy from water and oxygen!

```
modelClosed = modelClosedOri;
modelClosedATP = changeObjective(modelClosed,'DM_atp_c_');
modelClosedATP = changeRxnBounds(modelClosedATP,'DM_atp_c_',0,'l');
modelClosedATP = changeRxnBounds(modelClosedATP,'EX_h2o[e]',-1,'l');
modelClosedATP = changeRxnBounds(modelClosedATP,'EX_o2[e]',-1,'l');

FBA6=optimizeCbModel(modelClosedATP);
TableChecks{cnt,1} = 'Exchanges, sinks, and demands have  lb = 0, except h2o
and o2';
if abs(FBA6.f) > 1e-6
    TableChecks{cnt,2} = 'model produces energy from water and oxygen!';
else
    TableChecks{cnt,2} = 'model DOES NOT produce energy from water and
oxygen!';
end
cnt = cnt + 1;
```

Test if the model produces matter when atp demand is reversed!

```
modelClosed = modelClosedOri;
modelClosed = changeObjective(modelClosed,'DM_atp_c_');
modelClosed.lb(find(ismember(modelClosed.rxns,'DM_atp_c_'))) = -1000;
FBA = optimizeCbModel(modelClosed);
TableChecks{cnt,1} = 'Exchanges, sinks, and demands have  lb = 0, allow
DM_atp_c_ to be reversible';
```

```
if abs(FBA.f) > 1e-6
    TableChecks{cnt,2} = 'model produces matter when atp demand is
reversed!';
else
    TableChecks{cnt,2} = 'model DOES NOT produce matter when atp demand is
reversed!';
end
cnt = cnt + 1;
```

Test if the model has flux through h[m] demand !

```
modelClosed = modelClosedOri;
modelClosed = addDemandReaction(modelClosed,'h[m]');
modelClosed = changeObjective(modelClosed,'DM_h[m]');
modelClosed.ub(find(ismember(modelClosed.rxns,'DM_h[m]'))) = 1000;
FBA = optimizeCbModel(modelClosed,'max');
TableChecks{cnt,1} = 'Exchanges, sinks, and demands have  lb = 0, test flux
through DM_h[m] (max)';
if abs(FBA.f) > 1e-6
    TableChecks{cnt,2} = 'model has flux through h[m] demand (max)!';
else
    TableChecks{cnt,2} = 'model has NO flux through h[m] demand (max)!';
end
cnt = cnt + 1;
```

Test if the model has flux through h[c] demand !

```
modelClosed = modelClosedOri;
modelClosed = addDemandReaction(modelClosed,'h[c]');
modelClosed = changeObjective(modelClosed,'DM_h[c]');
modelClosed.ub(find(ismember(modelClosed.rxns,'DM_h[c]'))) = 1000;
FBA = optimizeCbModel(modelClosed,'max');
TableChecks{cnt,1} = 'Exchanges, sinks, and demands have  lb = 0, test flux
through DM_h[c] (max)';
if abs(FBA.f) > 1e-6
    TableChecks{cnt,2} = 'model has flux through h[c] demand (max)!';
else
    TableChecks{cnt,2} = 'model has NO flux through h[c] demand (max)!';
end
cnt = cnt + 1;
```

Test if the model produces too much atp demand from glucose under aerobic condition. Also consider using the tutorial testModelATPYield to test if the correct ATP yield from different carbon sources can be realized by the model.

```
modelClosed = modelClosedOri;
modelClosed = changeObjective(modelClosed,'DM_atp_c_');
modelClosed.lb(find(ismember(modelClosed.rxns,'EX_o2[e]'))) = -1000;
modelClosed.lb(find(ismember(modelClosed.rxns,'EX_h2o[e]'))) = -1000;
modelClosed.ub(find(ismember(modelClosed.rxns,'EX_h2o[e]'))) = 1000;
```

```
modelClosed.ub(find(ismember(modelClosed.rxns,'EX_co2[e]'))) = 1000;
FBAOri = optimizeCbModel(modelClosed,'max');

TableChecks{cnt,1} = 'ATP yield ';
if abs(FBAOri.f) > 31 % this is the theoretical value
     TableChecks{cnt,2} = 'model produces too much atp demand from glc!';
else
   TableChecks{cnt,2} ='model DOES NOT produce too much atp demand from
glc!';
end
cnt = cnt + 1;
```

Test metabolic objective functions with open sinks. Note this step is time consuming and may only work reliably on Recon 3D derived models due to different usage of abbreviations.

```
TableChecks{cnt,1} = 'Test metabolic objective functions with open sinks';
if 1 % perform test function
    [TestSolution,TestSolutionNameOpenSinks, TestedRxnsSinks, PercSinks] =
Test4HumanFctExt(model,'all');
    TableChecks{cnt,2} = strcat('Done. See variable
TestSolutionNameOpenSinks for results. The model passes
', num2str(length(find(abs(TestSolution)>tol))),' out of ',
num2str(length(TestSolution)), 'tests');
else
    TableChecks{cnt,2} = 'Not performed.';
end
cnt =  cnt + 1;
```

Test metabolic objective functions with closed sinks (lb). Note this step is time consuming and may only work reliably on Recon 3D derived models due to different usage of abbreviations.

```
TableChecks{cnt,1} = 'Test metabolic objective functions with closed sinks
(lb)';
if 1 % perform test functions
    [TestSolution,TestSolutionNameClosedSinks, TestedRxnsClosedSinks,
PercClosedSinks] = test4HumanFctExt(model,'all',0);
    TableChecks{cnt,2} = strcat('Done. See variable
TestSolutionNameClosedSinks for results. The model passes
', num2str(length(find(abs(TestSolution)>tol))),' out of ',
num2str(length(TestSolution)), 'tests');
else
    TableChecks{cnt,2} = 'Not performed.';
end
cnt =  cnt + 1;
```

Compute ATP yield. This test is identical to the material covered in the tutorial testModelATPYield.

```
TableChecks{cnt,1} = 'Compute ATP yield';
if 1 % test ATP yield
    [Table_csources, TestedRxns, Perc] = testATPYieldFromCsources(model);
```

```
    TableChecks{cnt,2} = 'Done. See variable Table_csources for results.';
else
    TableChecks{cnt,2} = 'Not performed.';
end
cnt = cnt + 1;
```

Check for duplicated reactions in the model.

```
TableChecks{cnt,1} = 'Check duplicated reactions';
method='FR';
removeFlag=0;
[modelOut,removedRxnInd, keptRxnInd] =
checkDuplicateRxn(model,method,removeFlag,0);
if isempty(removedRxnInd)
    TableChecks{cnt,2} = 'No duplicated reactions in model.';
else
    TableChecks{cnt,2} = 'Duplicated reactions in model.';
end
cnt = cnt + 1;
```

Check empty columns in 'model.rxnGeneMat'.

```
TableChecks{cnt,1} = 'Check empty columns in rxnGeneMat';
E = find(sum(model.rxnGeneMat)==0);
if isempty(E)
    TableChecks{cnt,2} = 'No empty columns in rxnGeneMat.';
else
    TableChecks{cnt,2} = 'Empty columns in rxnGeneMat.';
end
cnt = cnt + 1;
```

Check that demand reactions have a lb >= 0.

```
TableChecks{cnt,1} = 'Check that demand reactions have a lb >= 0';
DMlb = find(model.lb(strmatch('DM_',model.rxns))<0);
if isempty(DMlb)
    TableChecks{cnt,2} = 'No demand reaction can have flux in backward
direction.';
else
    TableChecks{cnt,2} = 'Demand reaction can have flux in backward
direction.';
end
cnt = cnt + 1;
```

Check whether singleGeneDeletion runs smoothly.

```
TableChecks{cnt,1} = 'Check whether singleGeneDeletion runs smoothly';
try
    [grRatio,grRateKO,grRateWT,hasEffect,delRxns,fluxSolution] =
singleGeneDeletion(model);
```

```
    TableChecks{cnt,2} = 'singleGeneDeletion finished without problems.';
catch
    TableChecks{cnt,2} = 'There are problems with singleGeneDeletion.';
end
cnt = cnt + 1;
```

Check for flux consistency.

```
TableChecks{cnt,1} = 'Check for flux consistency';
param.epsilon=1e-4;
param.modeFlag=0;
%param.method='null_fastcc';
param.method='fastcc';
printLevel = 1;
[fluxConsistentMetBool,fluxConsistentRxnBool,fluxInConsistentMetBool,fluxInCo
nsistentRxnBool,model] = findFluxConsistentSubset(model,param,printLevel);
if isempty(find(fluxInConsistentRxnBool))
    TableChecks{cnt,2} = 'Model is flux consistent.';
else
    TableChecks{cnt,2} = 'Model is NOT flux consistent';
end
cnt = cnt + 1;
```

Display all results.

```
TableChecks
```

Save all results.

```
resultsFileName = 'TestResults';
save(strcat(resultsFileName,'.mat'));
```

References

[1] Brunk, E. et al. Recon 3D: A resource enabling a three-dimensional view of gene variation in human metabolism. (submitted) 2017.