# Metabotools tutorial II - Integration of quantitative metabolomic data

## Authors: Maike K. Aurich, Sylvain Arreckx, Systems Biochemistry Group, LCSB, University of Luxembourg.

## Reviewer(s): Anne Richelle, Lewis Lab at University of California, San Diego.

### INTRODUCTION

In this tutorial, we  generate a contextualized model by integrating quantitative extracellular metabolomic data [1]. We will afterwards analyze the behavior of the models to variations in flux by using phenotypic phase plane analysis.

Before running a section in the tutorial, read the corresponding sections in the MetaboTools protocol and supplemental tutorial (Data sheet 3, http://journal.frontiersin.org/article/10.3389/fphys.2016.00327/full).

### PROCEDURE

Clear workspace and initialize the COBRA Toolbox

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

## Step 0 - Define the output location and set the LP solver

```
global CBTDIR % set path to cobratoolbox (pathToCOBRA)
outputPath = pwd;% ouputPath = 'ADD YOUR PATH TO YOUR OUTPUT FOLDER'
solver = 'gurobi';  % can be gurobi or 'ibm_cplex'
solverOK = changeCobraSolver(solver, 'LP');
if solverOK == 1
    display('The LP solver is set.');
else
    error('The LP solver is not set.')
end
```

Load and check that the input model is correclty loaded

```
tutorialPath = fileparts(which('tutorial_metabotools2.m'));
if exist([tutorialPath filesep 'starting_model.mat'], 'file') == 2  % 2
means it's a file.
    starting_model = readCbModel([tutorialPath filesep
'starting_model.mat']);
    display('The model is loaded.');
else
    error('The ''starting_model'' could not be loaded.');
end
```

Check output path and writing permission

```
if ~exist(outputPath) == 7
    error('Output directory in ''outputPath'' does not exist. Verify that
you type it correctly or create the directory.');
end

% Make and save dummy file to test writing to output directory
A = rand(1);
try
    save([outputPath filesep 'A']);
catch ME
    error('Files cannot be saved to the provided location: %s\nObtain rights
to write into %s directory or set ''outputPath'' to a different directory.',
outputPath, outputPath);
end
```

## Step 1 - Shaping the model's environment using setMediumConstraints

## Define the model bounds using setMediumConstraints

Constrain the model using the data related to medium composition. To this end, define the set of exchange reactions for which exometabolomic data are available. In this example, no data are available

```
medium_composition = {};
met_Conc_mM = [];
```

Define constraints on basic medium components (i.e., metabolites that are uptake from the medium but not captured by the measured data)

```
mediumCompounds = {'EX_h(e)', 'EX_h2o(e)', 'EX_hco3(e)', 'EX_nh4(e)',
'EX_o2(e)', 'EX_pi(e)', 'EX_so4(e)'};
ions = {'EX_ca2(e)', 'EX_cl(e)', 'EX_co(e)', 'EX_fe2(e)', 'EX_fe3(e)',
'EX_k(e)', 'EX_na1(e)', 'EX_i(e)', 'EX_sel(e)'};
mediumCompounds = [ions mediumCompounds];
mediumCompounds_lb = -100;
```

Define also additional constraints to limit the model behaviour (e.g., secretion of oxygen, essential amino acids that need to be taken up)

```
customizedConstraints = {'EX_co2(e)', 'EX_o2(e)', 'EX_his_L(e)',
'EX_ile_L(e)', 'EX_leu_L(e)', ...
                         'EX_lys_L(e)', 'EX_phe_L(e)', 'EX_thr_L(e)',
'EX_trp_L(e)', 'EX_val_L(e)', ...
                         'EX_met_L(e)', 'EX_ascb_L(e)', 'EX_btn(e)',
'EX_chol(e)', 'EX_fol(e)', ...
                         'EX_pnto_R(e)', 'EX_retn(e)', 'EX_thm(e)',
'EX_vitd2(e)', 'EX_vitd3(e)', 'EX_retinol(e)'};
customizedConstraints_ub = [2000, 0, 2000, 2000, 2000, 2000, 2000, 2000,
2000, 2000, 2000, 2000, ...
```

```
                            2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000,
2000];
customizedConstraints_lb = [-100, -1000, -10, -10, -10, -10, -10, -10, -10,
-10, -10, -1, -1, -1, ...
                            -1, -1, -1, -1, -1, -1, -1];
```

Apply the medium constraints previously defined using *setMediumConstraints*. Note that you can also provide information related to the cell concentration (*cellConc*), the cell weight (*cellWeight*), the time (*t*), the current value and the new value for infinite constraints (respectively *current_inf* and *set_inf*).

```
cellConc = [];
cellWeight = [];
t = [];
set_inf = 2000;
current_inf = 1000;
close_exchanges = 0;
[modelMedium, basisMedium] = setMediumConstraints(starting_model, set_inf,
current_inf, medium_composition, met_Conc_mM, ...
                                                  cellConc, t, cellWeight,
mediumCompounds, mediumCompounds_lb, customizedConstraints, ...
                                                  customizedConstraints_ub,
customizedConstraints_lb, close_exchanges);

clearvars -EXCEPT modelMedium tol solver outputPath tutorialPath solverQuant
```

## Step 2 : Generate an individual exchange profiles for each sample

Generate individual uptake and secretion profiles from fluxes data using *prepIntegrationQuant*. Note that negative flux values are interpreted as uptake and positive values as secretion. Moreover, the function removes from each sample the metabolite uptakes or secretions that cannot be realized by the model due to missing production or degradation pathways, or blocked reactions. If only secretion is not possible, only secretion is eliminated from the sample profile whereas uptake will still be mapped.

```
load([tutorialPath filesep 'tutorial_II_data.mat']);
model = modelMedium;
test_max = 500;
test_min = 0.0001;
variation = 20;

prepIntegrationQuant(model, metData, exchanges, samples, test_max, test_min,
outputPath);
clearvars -EXCEPT modelMedium samples tol solver outputPath tutorialPath
solverQuant
```

Use *checkExchangeProfiles*  generate a summary of the number of uptake and secretion exchanges per samples.

```
nmets = 70;
```

```
[mapped_exchanges, minMax, mapped_uptake, mapped_secretion] =
checkExchangeProfiles(samples, outputPath, nmets);
clearvars -EXCEPT modelMedium samples tol solver mapped_exchanges
outputPath tutorialPath solverQuant
save([outputPath filesep 'Result_checkExchangeProfiles']);
```

## Step 3 : Generate contextualized model

Use the function *setQuantConstraints* to integrate the uptake and secretion profiles and generate condition-specific metabolic models for each sample. The function allows the definition of metabolites that should not be consumed (*no uptake*) or not secreted (*no secretion*). Note that the solver ='ibm_cplex' is required for this step.

```
solverQuant = 'ibm_cplex';
changeCobraSolver(solverQuant, 'LP');

minGrowth = 0.008;% lower bound to the biomass reaction obj
obj = 'biomass_reaction2';

no_secretion = {'EX_o2(e)'};
no_uptake = {'EX_o2s(e)', 'EX_h2o2(e)'};
medium = {};% reactions that should be excluded from minimization of
exchanges
tol = 1e-6;
model = modelMedium;
epsilon = 1e-4;

addExtraExch = {'EX_tdchola(e)', 'Ex_5hoxindoa[e]'};% metabolite exchanges
that are added to the upper and lower bounds
addExtraExch_value = 1;% flux values that are added to the upper and lower
bounds
[ResultsAllCellLines, OverViewResults] = setQuantConstraints(model, samples,
tol, minGrowth, obj, no_secretion, ...
                                                            no_uptake,
medium, addExtraExch, addExtraExch_value, outputPath);
clearvars -EXCEPT modelMedium samples ResultsAllCellLines OverViewResults
tol solver mapped_exchanges outputPath tutorialPath
```

## Step 4 : Analyze added exchanges

Use the function *statisticsAddedExchanges* to get a table summarizing the exchange reactions that were added to the models. The function *mkTableOfAddedExchanges* generates a table that summarizes the reactions added to individual models (*Added all*).

```
changeCobraSolver(solver, 'LP');

[Ex_added_all_unique] = statisticsAddedExchanges(ResultsAllCellLines,
samples);
```

```
clearvars -EXCEPT modelMedium samples ResultsAllCellLines OverViewResults
Ex_added_all_unique tol solver mapped_exchanges outputPath tutorialPath

[Added_all] = mkTableOfAddedExchanges(ResultsAllCellLines, samples,
Ex_added_all_unique);

save([outputPath filesep 'statistics']);
clearvars -EXCEPT modelMedium samples ResultsAllCellLines OverViewResults
tol solver mapped_exchanges outputPath tutorialPath
```

## Step 5 : Analyze the sets of essential genes

Use the function *analyzeSingleGeneDeletion* to predict and analyze the sets of essential genes across a set of models.

```
cutoff = 0.05;
[genes, ResultsAllCellLines, OverViewResults] =
analyzeSingleGeneDeletion(ResultsAllCellLines, outputPath, samples, cutoff,
OverViewResults);
clearvars -EXCEPT modelMedium samples ResultsAllCellLines OverViewResults
Ex_added_all_unique genes tol solver mapped_exchanges outputPath tutorialPath
```

The output table (genes) consists of six columns specifying (1) the gene ID, (2) the category, (3) the number of models for which the gene is essential (growth ratio < 0.05), (4) the number of models for which the gene is not essential (growth ratio > 0.95), (5) the number of models in which the growth (ratio between wild-type and knock-out model) is reduced (0.05 > growth ratio > 0.95), and (6) the number of models that do not have the gene.

## Step 6 : Check reaction essentiality

Use the function checkEffectRxnKO to investigate which individual gene-associated reaction makes the model infeasible (i.e., reactions associated with a gene need to carry flux).

```
samples_to_test = samples; %(sub-)set of models
fill = 'NAN'; %Define a placeholder for an empty cell
genes_to_test = {'55293.1'}; %set of genes

[FBA_Rxns_KO, ListResults] = checkEffectRxnKO(samples_to_test, fill,
genes_to_test, samples, ResultsAllCellLines);

clearvars -EXCEPT modelMedium samples ResultsAllCellLines OverViewResults
Ex_added_all_unique genes FBA_Rxns_KO ListResults tol solver
mapped_exchanges outputPath tutorialPath
```

## Step 7 : Generate an intersect model and an union model

The function *makeSummaryModels* generates a union model (*unionModel*) and an intersect model (*intersectModel*). Additionally, it writes out the set of reactions (*diffRxns*) and exchange reactions (*diffExRxns*) that distinguish the union and the intersect model for further analysis.

```
mk_union = 1;
mk_intersect = 1;
mk_reactionDiff = 1;
starting_model = readCbModel([tutorialPath filesep 'starting_model.mat']);
model = starting_model;

[unionModel, intersectModel, diffRxns, diffExRxns] =
makeSummaryModels(ResultsAllCellLines, samples, model, mk_union,
mk_intersect, mk_reactionDiff);
clearvars -EXCEPT modelMedium samples ResultsAllCellLines OverViewResults
Ex_added_all_unique genes FBA_Rxns_KO ListResults unionModel intersectModel
diffRxns diffExRxns tol solver mapped_exchanges outputPath model tutorialPath

save([outputPath filesep 'summary']);
```

## Step 8 : Predict differences in metabolite production or consumption

Use the function predictFluxSplits to predict the production or consumption (defined using *dir*) of metabolites of interest (*met2test* = *atp*) for each model defined by *samples*. Define the objective function (*obj*) that should be used to generate the flux vector from which the reaction contributions for metabolite production/consumption are calculated. Furthermore, an ATP yield (*ATPyield*) can be predicted, which relates the total ATP production to a user-defined carbon source (default: *carbon source* = 'EX glc(e)'). The ATP yield can be used as a metric to distinguish metabolic phenotypes. Reported are the reactions carrying highest flux for production/consumption of the metabolite of interest (*maximum contributing rxn, maximum contributing flux*).This analysis requires a quadratic programming (*QP*) solver.

```
changeCobraSolver(solver, 'QP');
obj = 'DM_atp_c_';
carbon_source = {'EX_glc(e)'};
samples = samples(1:4, 1);
dir = 1;
```

## Example A - ATP production

```
% exclude transport reactions from flux split analysis
transportRxns = {'ATPtm'; 'ATPtn'; 'ATPtx'; 'ATP1ter'; 'ATP2ter';
'EX_atp(e)'; 'DNDPt13m';...
                 'DNDPt2m'; 'DNDPt31m'; 'DNDPt56m'; 'DNDPt32m'; 'DNDPt57m';
'DNDPt20m';...
                 'DNDPt44m'; 'DNDPt19m'; 'DNDPt43m'; 'ADK1'; 'ADK1m'};
ATPprod = {'ATPS4m'; 'PGK'; 'PYK'; 'SUCOASm'};
met2test = {'atp[c]', 'atp[m]', 'atp[n]', 'atp[r]', 'atp[x]'};

[BMall, ResultsAllCellLines, metRsall, maximum_contributing_rxn,
maximum_contributing_flux, ATPyield] = predictFluxSplits(model,...
    obj, met2test, samples,ResultsAllCellLines, dir,  transportRxns,
ATPprod, carbon_source);

PHs = [samples maximum_contributing_rxn(:, 1)];
```

```
maximum_contributing_flux_ATP = maximum_contributing_flux;
clear ATPprod transportRxns met2test maximum_contributing_rxn
```

## Example B - NADH production

```
met2test = {'nadh[c]', 'nadh[m]', 'nadh[n]', 'nadh[x]', 'nadh[r]'};
transportRxns = {'NADHtpu'; 'NADHtru'; 'NADtpu'};

[BMall, ResultsAllCellLines, metRsall, maximum_contributing_rxn,
maximum_contributing_flux_NADH] = predictFluxSplits(model,...
    obj, met2test, samples, ResultsAllCellLines, dir, transportRxns);

PHs = [PHs maximum_contributing_rxn(:, 1)];
clear transportRxns met2test maximum_contributing_rxn
```

## Example C - FADH2 production

```
transportRxns = {'FADH2tru'; 'FADH2tx'};
met2test = {'fadh2[c]', 'fadh2[m]', 'fadh2[n]', 'fadh2[x]', 'fadh2[r]'};

[BMall, ResultsAllCellLines, metRsall, maximum_contributing_rxn,
maximum_contributing_flux_FADH2] = predictFluxSplits(model,...
    obj, met2test, samples, ResultsAllCellLines, dir,  transportRxns);
PHs = [PHs maximum_contributing_rxn(:, 1)];
clear transportRxns met2test
```

## Example D - NADPH production

```
transportRxns = {'NADPHtru'; 'NADPHtxu'};
met2test = {'nadph[c]', 'nadph[m]', 'nadph[n]', 'nadph[x]', 'nadph[r]'};

[BMall, ResultsAllCellLines, metRsall, maximum_contributing_rxn,
maximum_contributing_flux_NADPH] = predictFluxSplits(model,...
    obj, met2test, samples, ResultsAllCellLines, dir, transportRxns);

PHs = [PHs maximum_contributing_rxn(:, 1)];
clear transportRxns met2test
save([outputPath filesep 'fluxSplits']);
```

## Step 9 - Illustrate the phenotypes (PHs) on 3Dplot

The function make3Dplot allows illustration of the results of the previous analysis. The colors specify different phenotypes.

```
diff_view = 1;
fonts = 18;

make3Dplot(PHs, maximum_contributing_flux_ATP, fonts, outputPath, diff_view);
```

## Step 10 - Perform Phase Plane Analysis

Use the function *performPPP* to investigate the behavior of the models to variations in flux through a pair of exchange reactions. The following example illustrates the testing of the robustness of the models to variation of glucose uptake & oxygen uptake, and glutamine secretion & oxygen uptake (defined in *mets*). The range of flux values to be tested is defined by the number of steps and step size (*step num* and *step size*). The direction of exchange is defined individually for each exchange (*direct*).

```
mets = {'EX_glc(e)', 'EX_o2(e)'; 'EX_gln_L(e)', 'EX_o2(e)'; 'EX_lac_L(e)',
'EX_o2(e)'};
step_size = [40, 40; 20, 40; 40, 40];
step_num = [28, 26; 21, 26; 42, 26];
direct = [-1, -1; -1, -1; 1, -1];

[ResultsAllCellLines] = performPPP(ResultsAllCellLines, mets, step_size,
samples, step_num, direct);

save([outputPath filesep 'PPP']);
```

Use illustrate_ppp to illtustrate the results of the phase plane analysis

```
label = {'Glucose uptake (fmol/cell/hr)'; 'Oxygen uptake (fmol/cell/hr)';
'Growth rate (hr-1)'};
mets = {'EX_glc(e)'; 'EX_o2(e)'};
fonts = 12;
samples = {'IGROV1'};
illustrate_ppp(ResultsAllCellLines, mets, outputPath, samples, label, fonts,
tol);
```

**REFERENCE**

1. Jain M, Nilsson R, Sharma S, Madhusudhan N, Kitami T, et al. (2012) Metabolite Profiling Identifies a Key Role for Glycine in Rapid Cancer Cell Proliferation. Science 336: 1040–1044.