COBRA Toolbox Minitutorial

Markus Herrgard (mherrgar@ucsd.edu)

1. Setting up the COBRA Toolbox

Install SBML Toolbox and the linear programming solver that is going to be used with the toolbox. Make sure that both the SBML Toolbox and the LP solver are functional and accessible in the Matlab path. Edit initCobraToolbox to change the default LP solver and run the initCobraToolbox script to set the correct path to the COBRA Toolbox folders. The LP solver can be changed during the use of the Toolbox by the following function (solver changed to Glpk in this example):

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

2. Reading and writing SBML models

Read in E. coli iJR904 model in SBML format (assuming the model SBML file is in the current working directory):

```
model = readCbModel('Ec_iJR904_GlcMM')
```

Alternatively read in the model using a diaglog box that allows choosing the file name and location:

```
model = readCbModel;
```

Write the model to a SBML file (you will be prompted for the file name and location):

```
writeCbModel(model,'sbml');
```

Write the model to a Excel file named 'test.xls':

```
writeCbModel(model,'xls','test.xls');
```

3. Changing model parameters and basic FBA calculations

Solve FBA problem (maximize default objective in model):

```
solution = optimizeCbModel(model, 'max',false,false)
```

Print out solution vector (only nonzero exchange fluxes):

```
printFluxVector(model, solution.x, true, true);
```

Solve FBA problem, but this time return the minimum 1-norm solution:

```
solution2 = optimizeCbModel(model, 'max',false,true)
```

Compare the two solutions:

```
printFluxVector(model, [solution.x solution2.x], true, false);
```

Change carbon souce to succinate:

model2 = changeRxnBounds(model, {‘EX_glc(e)’,'EX_fru(e)'}, [0 -9], 'l'); solution = optimizeCbModel(model2); printFluxVector(model2, solution.x, true, true);

Change to anaerobic conditions:

model3 = changeRxnBounds(model, 'EX_o2(e)', 0, 'l'); solution = optimizeCbModel(model3); printFluxVector(model3, solution.x, true, true);

Change objective to maximizing ethanol production:

model4 = changeObjective(model, 'EX_etoh(e)',1); solution = optimizeCbModel(model4); printFluxVector(model4, solution.x, true, true);

4. Dynamic FBA

Change glucose and oxygen uptake rates:

model = changeRxnBounds(model, {'EX_glc(e)','EX_o2(e)'}, [-10 -18], 'l');

Set up parameters for dynamic FBA simulation:

substrateRxns = {'EX_glc(e)'}; initConcentrations = 10; initBiomass = .035; timeStep = .25; nSteps = 20; plotRxns = {'EX_glc(e)','EX_ac(e)'};

Run dynamic FBA:

dynamicFBA(model,substrateRxns,initConcentrations,... initBiomass,timeStep,nSteps,plotRxns);

Dynamic FBA will run until the most limiting of the nutrients is exhausted (typically carbon source). At this point the function will terminate and plot the results of the simulation.

5. Robustness analysis

Read in S. cerevisiae iND750 model:

model = readCbModel('Sc_iND750_GlcMM')

Run robustness analysis of the PGK reaction:

robustnessAnalysis(model, 'PGK', 20);

Run robustness analysis of the PGL reaction:

robustnessAnalysis(model, 'PGL', 20);

6. Single and double gene deletion analysis

Run model-wide single gene deletion analysis using FBA:

grRatioFBA = singleGeneDeletion(model,'FBA');

Run model-wide single gene deletion analysis using linear MOMA:

grRatioMOMA= singleGeneDeletion(model,'lMOMA');

Plot results for both FBA and linear MOMA analysis:

plot(1:length(grRatioFBA),[sort(grRatioMOMA) sort(grRatioFBA)],'.');

Run model-wide double gene deletion analysis using FBA:

grRatioDouble = doubleGeneDeletion(model,'FBA');

Find epistatic interactions and plot the distribution of the number of interactions:

interactions = findEpistaticInteractions(model,grRatioDouble); nInteractions = sum(interactions); plot(sort(nInteractions(nInteractions > 0)),'-');

7. Flux variability analysis

Read in E. coli iJR904 model:

model = readCbModel('Ec_iJR904_GlcMM');

Run flux variability analysis:

[minFlux,maxFlux] = fluxVariability(model,90);

The minimum and maximum fluxes for each reaction are returned in minFlux and maxFlux. Plot results for glycolysis and the pentose phosphate pathway:

rxnNames = {'PGI','PFK','FBP','FBA','TPI','GAPD','PGK','PGM','ENO',... 'PYK','PPS','G6PDH2r','PGL','GND','RPI','RPE','TKT1','TKT2','TALA'};

rxnID = findRxnIDs(model,rxnNames); printLabeledData(model.rxns(rxnID),[minFlux(rxnID) maxFlux(rxnID) maxFlux(rxnID) ... - minFlux(rxnID)],true,3);

8. Uniform random sampling of metabolic solution space

Set a constraint for minimum growth rate (90% of the optimal):

sol = optimizeCbModel(model); growthRate = sol.f; model = changeRxnBounds(model,'BiomassEcoli',0.9*growthRate,'l');

Prepare the model for sampling and sample the solution space of the model using default settings (1 million steps):

[modelSampling,samples] = sampleCbModel(model,'Ec_iJR904_GlcMM_flux');

The pre-processed model is returned in modelSampling and a subsample of all the flux samples is returned in variable samples. Plot histograms and pairwise scatterplots for glycolytic reactions:

```
    rxnNames = {'PGI','PFK','FBP','FBA','TPI','GAPD','PGK',...
'PGM','ENO','PYK','PPS'};

    sampleScatterMatrix(rxnNames,modelSampling,samples);
```

9. Finding correlated reaction sets using sampling

Identify sets based on sampling results created in step 8:

```
    [sets,setNumber] = identifyCorrelSets(modelSampling,samples);
```

Select the largest five sets:

```
    selectRxns = setNumber > 0 & setNumber <=5; outputRxnList =
modelSampling.rxns(selectRxns); outputSetNumber = setNumber(selectRxns);
```

Export in Cytoscape .sif format (show only metabolites that are used in less than 40 reactions):

```
    outputNetworkCytoscape(modelSampling,
'iND750_correlSets',outputRxnList,outputSetNumber,40);
```