Flux Balance Analysis (FBA) and its variants

Author(s): Ronan M.T. Fleming, Leiden University;Vanja Vlasov, LCSB, University of Luxembourg; Thomas Pfau, Systems Biology Group, LSRU, University of Luxembourg

Reviewer(s): Ines Thiele, Catherine Clancy, National University of Ireland, Galway. Thomas Pfau, Systems Biology Group, LSRU, University of Luxembourg.

INTRODUCTION

Flux balance analysis (FBA) evaluates the metabolic flux distribution[1], and is one of the most used modelling approaches for metabolic systems.

The applications of FBA for molecular systems biology include prediction of the growth rates, uptake rates, knockout lethality and product secretion. In FBA, the solution space is constrained by the assumption of a steady-state, under which each internal metabolite is consumed at the same rate as it is produced.

For the quantitative estimation of the metabolic fluxes, linear programming (LP) can be used to solve the stoichiometric matrix for a given objective function under different constraints. The constraints of the problem depict the space of all eligible possibilities from which an optimal solution can be selected;

$$\begin{array}{ll} \min\limits_{v} & c^T v \\ \text{s.t.} & Sv = b, \\ & l \leq v \leq u, \end{array}$$

Equation 1: Formula of standard FBA.

where $c \in \Re^n$ is a parameter vector that linearly combines one or more reaction fluxes to form what is termed the objective function, and where a $b_i < 0$, or $b_i > 0$, represents some fixed output, or input, of the ith molecular species. $S \in \Re^{m \times n}$ is a stoichiometric matrix for m molecular species and n reactions, and b is a vector of known metabolic exchanges. The output of FBA is a particular flux distribution, v, which maximises or minimises the objective function and stands between upper and lower bounds, u and l, respectively.

There are multiple different variants of FBA, some of which will be demonstrated here:

1. **Standard FBA**
2. **Regularised FBA**
3. **Sparse FBA**
4. **Metabolite dilution FBA (mdFBA)**
5. **Geometric FBA**
6. **Parsimonious enzyme usage Flux Balance Analysis (pFBA)**
7. **Dynamic FBA**

8. **Relaxed FBA**
9. **Flux enrichment analysis (FEA)**

EQUIPMENT SETUP

Initialise The Cobra Toolbox and set the solver.

If necessary, initialise the cobra toolbox:

```
if 0
    initCobraToolbox(false) % false, as we don't want to update
end
```

For solving LP problems in a FBA analysis, certain solvers are required and can be set using the `changeCobraSolver` function:

```
% solverOK = changeCobraSolver(solverName, solverType, printLevel, unchecked)
```

The present tutorial can run with the GLPK package, which does not require additional installation and configuration. Although, for the analysis of large models is recommended to use the GUROBI package.

Setup the appropriate solver for the machine you are using by removing the "%" (comment) sign for only the desired solver.

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

```
> The solver compatibility is not tested with MATLAB R2019b.
> Solver for LP problems has been set to glpk.
> The solver compatibility is not tested with MATLAB R2019b.
> Solver for MILP problems has been set to glpk.
> Solver glpk not supported for problems of type MIQP. No solver set for this problemtype
> Solver glpk not supported for problems of type NLP. No solver set for this problemtype
> Solver glpk not supported for problems of type QP. Currently used: qpng
```

```
% changeCobraSolver('tomlab_cplex','all');
% changeCobraASolver('ibm_cplex','all');
% changeCobraSolver ('gurobi', 'all');
```

Model Setup

This tutorial will use the generic model of the human cellular metabolism[2], Recon 2.0. Other COBRA models, including Recon 3, may also be run with this tutorial. For information on metabolites structures and reactions, and to download the latest COBRA model releases, visit the Virtual Metabolic Human database (VMH, http://vmh.life).

Before proceeding with the simulations, load the model into the workspace:

```
global CBTDIR
%modelFileName = 'Recon2.0model.mat';
modelFileName = 'Recon3DModel_301.mat';
```

```
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
if 0
    model = readCbModel(modelFileName);
else
    load('Recon3DModel_301.mat')
    model = Recon3DModel;
end
```

In this tutorial we assume, that the cellular objectives include energy production or optimisation of uptake rates and by-product secretion for various physiological functions of the human body.

PROCEDURE

1. Standard FBA

Standard FBA predicts an optimal solution for a cellular objective within a given set of constraints on a metabolic network (see Equation 1). Constraints on the network are set by assigning limits on the uptake, consumption or production of metabolites in reactions.

Timing:

The time to determine a FBA solution depends on the size of the genome-scale model and is commonly less than a second for a medium sized model.

Calculating maximal ATP energy production under aerobic conditions:

For each new simulation, the original model will be copied to a new variable. This preserves the constraints of the original model to perform further simulations with new constraints. Additionally, this method of renaming the model avoids confusion while performing multiple simulations at the same time.

```
modelaerobic = model;
```

The ATP demand reaction, i.e., `DM_atp_c_` within the model is a reaction that involves hydrolysis of ATP to ADP, Pi and proton in the cytosol.

```
 printRxnFormula(model, 'DM_atp_c_');
```

```
DM_atp_c_    h2o[c] + atp[c]     ->    h[c] + adp[c] + pi[c]
```

We will set this reaction as our objective with the `'changeObjective'` command. Maximising the flux through the ATP demand reaction will result in the network producing a maximal amount of ATP (up to the limit of the reaction).

```
modelaerobic = changeObjective (modelaerobic, 'DM_atp_c_');
```

The glucose and oxygen, in this case, are provided in high amounts for calculating the flux through ATP demand.

The `'changeRxnBounds'` function changes the flux constraints of the lower ('`l`'), upper ('u'), or both the bounds ('b'), of the specified reaction. Here, we will change the maximal uptake of glucose to 20 μmol/min/gDW and of oxygen to 1000 μmol/min/gDW. The uptake of oxygen is effectively unconstrainted (i.e. infinity).

```
% modelaerobic = changeRxnBounds (modelaerobic, 'EX_glc_D[e]', -20, 'l'); %
For Recon 3.0 uncomment these lines and
%modelaerobic = changeRxnBounds (modelaerobic, 'EX_o2[e]', -1000, 'l'); %
comment the lines below.
modelaerobic = changeRxnBounds (modelaerobic, 'EX_glc_D[e]', -20, 'l');
modelaerobic = changeRxnBounds (modelaerobic, 'EX_o2[e]', -1000, 'l');
```

The function `optimizeCbModel` calculates one of the optimal solutions for a (maximum or minimum) objective reaction within the defined solution space. In the above example, the maximal flux through the `DM_atp_c_` is desired.

```
FBAaerobic = optimizeCbModel (modelaerobic, 'max')
```

```
FBAaerobic = struct with fields:
         full: [10600×1 double]
          obj: 1000
        rcost: [10600×1 double]
         dual: [5835×1 double]
        slack: [5835×1 double]
       solver: 'glpk'
    algorithm: 'default'
         stat: 1
     origStat: 5
         time: 0.7090
        basis: []
            f: 1000
            x: [10600×1 double]
            v: [10600×1 double]
            w: [10600×1 double]
            y: [5835×1 double]
            s: [5835×1 double]
```

- **Anticipated results**

When oxygen and all carbon sources (internal and external) are provided the flux through ATP demand reaction can reach its maximum rate of 1000 μmol/min/gDW.

- **Troubleshooting**

If there are multiple carbon sources available in the model, it may be necessary to specify more constraints in order to examine the effect of a single carbon source on ATP production.

To avoid this issue, all external carbon sources need to be closed with the exception of the single carbon source of interest.

```
%Closing the uptake of all energy and oxygen sources
[exchBool,uptBool] = findExcRxns(model);
uptakes = model.rxns(uptBool);
```

```
% If you use Recon3.0 model, then:
% modelalter = model;
% modelalter = changeRxnBounds(modelalter, uptakes, 0, 'b');
% modelalter = changeRxnBounds(modelalter, 'EX_HC00250[e]', -1000, 'l');

% The alternative way to do that, in case you were using another large
model,
% that does not contain defined Subsystem is
% to find uptake exchange reactions with following codes:
% [selExc, selUpt] = findExcRxns(model);
% uptakes1 = model.rxns(selUpt);

% Selecting from the exchange uptake reactions those
% which contain at least 1 carbon in the metabolites included in the
reaction:
 subuptakeModel = extractSubNetwork(model, uptakes);
 hiCarbonRxns = findCarbonRxns(subuptakeModel,1);
% Closing the uptake of all the carbon sources
 modelalter = model;
 modelalter = changeRxnBounds(modelalter, hiCarbonRxns, 0, 'b');
% Closing other oxygen and energy sources. Use the following lines for
recon2, or uncomment the lines below for recon3
 exoxygen = {'EX_adp[e]',   'EX_amp[e]',    'EX_atp[e]',    'EX_co2[e]',
'EX_coa[e]',    'EX_fad[e]',    'EX_fe2[e]',...
    'EX_fe3[e]',    'EX_gdp[e]',    'EX_gmp[e]',    'EX_gtp[e]',
'EX_h[e]',    'EX_h2o[e]',    'EX_h2o2[e]',...
    'EX_nad[e]',    'EX_nadp[e]',    'EX_no[e]',    'EX_no2[e]',
'EX_o2s[e]'};
modelalter = changeRxnBounds (modelalter, exoxygen, 0, 'l');
```

Calculating maximum ATP energy production under anaerobic and glucose only conditions:

```
modelanaerobic = modelalter;
% modelaerobic = changeRxnBounds (modelaerobic, 'EX_glc_D[e]', -20, 'l'); %
For Recon 3.0 uncomment these lines and
%modelaerobic = changeRxnBounds (modelaerobic, 'EX_o2[e]', -1000, 'l'); %
comment the lines below.
modelanaerobic = changeRxnBounds(modelanaerobic, 'EX_glc_D[e]',-20,'l');
modelanaerobic = changeRxnBounds (modelanaerobic, 'EX_o2[e]', 0, 'l');
modelanaerobic = changeObjective(modelanaerobic,'DM_atp_c_');
FBAanaerob = optimizeCbModel(modelanaerobic,'max')
```

```
FBAanaerob = struct with fields:
         full: [10600×1 double]
          obj: 1.0000e+03
        rcost: [10600×1 double]
         dual: [5835×1 double]
        slack: [5835×1 double]
       solver: 'glpk'
    algorithm: 'default'
         stat: 1
     origStat: 5
         time: 0.6735
        basis: []
            f: 1.0000e+03
            x: [10600×1 double]
            v: [10600×1 double]
            w: [10600×1 double]
            y: [5835×1 double]
            s: [5835×1 double]
```

- **Anticipated results**

Compared to the aerobic condition, anaerobic condition with only glucose as an energy source has reduced flux through ATP demand (82 μmol/min/gDW), signifying the need to oxygen to run the oxidative phosphorylation. The results are dependant on the model you are using. For Recon 3.0, under anaerobic conditions with only glucose as an energy source, the flux for ATP demand is 40 μmol/min/gDW.

2. Quadratically regularised FBA

Regularised FBA calculates the optimal solution of a linear objective function, and finds the unique optimal flux vector that minimises the Euclidean norm of the flux vector, that is

$$\begin{aligned} \min_{v} \quad & c^T v + \eta ||v|| \\ \text{s.t.} \quad & Sv = b, \\ & l \leq v \leq u, \end{aligned}$$

If the scalar parameter η is small, then $c^T v$ dominates and the same optimal objective as in FBA is attained, with the advantage that a unique flux vector is returned - one that minimises the the sum of the squares of the flux vector.

In practice, for genome-scale, mono-scale models (e.g. not ME models) the following value for the scalar parameter works fine

```
eta = 1e-6;
```

This is a quadratic optimisation problem, so a 'QP' solver is required. The COBRA Toolbox v3 comes shipped with the solver PDCO: Primal-Dual interior method for Convex Objectives (further info here: link).

```
[solverOK, solverInstalled] = changeCobraSolver('pdco','QP');
```

Solve a quadratically regularised FBA problem:

```
FBAsolution = optimizeCbModel(model,'max',eta);
```

2. Sparse FBA

Sparse FBA calculates the optimal solution of a linear objective function and finds the smallest set of reactions that can carry flux to achieve the objective. Sparse FBA minimises the number of active reactions by keeping same maximal objective;

$$
\begin{aligned}
\min_{v} \quad & ||v||_0 \\
\text{s.t.} \quad & Sv = b, \\
& l \leq v \leq u, \\
& c^T v = \rho^*
\end{aligned}
$$

Equation 2: Formula of Sparse FBA.

where the last constraint is optional and represents the requirement to satisfy an optimal objective value ρ^* derived from any solution to a FBA problem. This approach is used to check for minimal sets of reactions that either should be active or should not be active in a flux balance model that is representative of a biochemical network.

```
% [vSparse, sparseRxnBool, essentialRxnBool]  = sparseFBA(model,
osenseStr,...
%  checkMinimalSet, checkEssentialSet, zeroNormApprox)
```

As an optional input, there are different appoximation types of zero-norm (only available when `minNorm = 'zero'`). Default is `cappedL1`.

```
% Other types of zero-norm:
%  * 'cappedL1' : Capped-L1 norm
%  * 'exp'      : Exponential function
%  * 'log'      : Logarithmic function
%  * 'SCAD'     : SCAD function
%  * 'lp-'      : :math:`L_p` norm with :math:`p < 0`
%  * 'lp+'      : :math:`L_p` norm with :math:`0 < p < 1`
%  * 'l1'       : L1 norm
%  * 'all'      : try all approximations and return the best result
```

Timing:

The time to determine a `sparseFBA()` solution depends on the size of the genome-scale model and is taking from < 1 second for a 1,000 reaction model, to < 2 seconds for a model with more than 10,000 reactions.

Calculating maximal ATP energy production under anaerobic and glucose only conditions:

```
modelspar = modelalter;
% For Recon3.0 model
% modelspar = changeRxnBounds (modelspar, 'EX_glc_D[e]', -20, 'l');
% modelspar = changeRxnBounds (modelspar, 'EX_o2[e]', 0, 'l');
modelspar = changeRxnBounds(modelspar, 'EX_glc_D[e]',-20,'l');
modelspar = changeRxnBounds (modelspar, 'EX_o2[e]', 0, 'l');
modelspar = changeObjective(modelspar, 'DM_atp_c_');
[vSparse, sparseRxnBool, essentialRxnBool] = sparseFBA(modelspar, 'max');
```

```
---FBA---
```

- **Anticipated results:**

Commonly, a sparse FBA solution will have much smaller number of active reactions compared to a standard FBA on the same model with same objective function. The outputs `sparseRxnBool` and `essentialRxnBool` return vectors with 1 and 0's, with sparse and essential reactions respectively.

Display the sparse flux solution, but only the non-zero fluxes. This is convenient with the `printFluxVector` function.

```
nonZeroFlag = 1;
printFluxVector(model, vSparse, nonZeroFlag)
```

```
10FTHFtm                        250
4ABUTtm                        -250
ABTArm                         -250
ACONTm                          250
AKGDm                           250
ATPtm                           250
CSm                             250
DCK1m                          -250
FOLt2                           500
GALt1r                          250
GLUDxm                          250
GLYtm                          -500
GTHDH                           500
H2CO3Dm                         250
H2Otm                           250
HPYRRy                          500
ICDHxm                          250
MTHFCm                         -250
MTHFD2m                        -250
NH4t3r                         -750
PCm                             250
PDHm                            250
SUCOASm                        -250
THFtm                          -500
TRIOK                           500
EX_HC00250[e]                  -500
EX_HC02160[e]                   250
EX_HC02161[e]                  -250
r0021                          -500
r0027                           250
r0081                           500
r0160                           500
r0178                          -250
r0193                          -500
r0377                           250
r0838                           250
r0940                          -500
r1411                           250
r1434                           500
RE2605C                         500
RE2898C                         500
EX_oh1[e]                      -500
FOLt2le                        -500
DM_Lcystin                      250
r0295                           500
MLTHFtm                         250
sink_thf[c]                    1000
DM_atp_c_                      1000
ENO                            1000
EX_h[e]                         750
```

```
EX_h2o[e]                      250
FBA                            250
GAPD                          1000
GHMT2r                        -500
MTHFC                          250
MTHFD                          250
PGI                            250
PGK                          -1000
PGM                          -1000
PGMT                           250
TPI                            250
UDPG4E                        -250
EX_nh4[e]                     -750
GALK                           250
PFK                            250
PYK                           1000
r0392                         -500
UGLT                           250
MLTHFte3                      1000
EX_mlthf[e]                  -1000
HMR_6607                      -500
HMR_8475                      -250
DM_4abut[c]                    250
DCMPtm                        -250
```

3. Metabolite dilution flux balance analysis (mdFBA)

This is a variant of FBA for predicting metabolic flux distributions by accounting for growth-associated dilution of all metabolites in a context-dependent manner[3].

A solution from the function `mdFBA` supports that all metabolites used in any reaction of the solution can either be produced by the network or taken up from the surrounding medium.

Timing:

Since this is a MIXED Integer Problem it can take a long time to solve.

Calculating ATP energy production under aerobic condition using mdFBA:

In this function, there is an optional output `newActives`, that represent reactions that are only active in this analysis.

```
% The valid solution can be produced with the Recon 3.0 model
% modelmd = model;
% modelmd = changeRxnBounds(modelmd, 'EX_glc_D[e]',-20,'l');
% modelmd = changeRxnBounds (modelmd, 'EX_o2[e]', -1000, 'l');
% modelmd = changeObjective(modelmd, 'DM_atp_c_');

% [sol, newActives] = mdFBA(modelmd)
```

- **Troubleshooting:**

When a model does not have a feasible solution, add the input: `'getInvalidSolution', true`.

```
% clear modelmd
modelnosol = modelalter;
modelnosol = changeObjective(modelnosol, 'DM_atp_c_');
[sol, newActives] = mdFBA(modelnosol,  'getInvalidSolution', true)
```

```
sol =

     []


newActives =

   0×0 empty cell array
```

Sometimes when an FBA of a model with the same objective function and constraints is run many times, or using different LP logarithm, we may get different set of solutions for individual reactions. In other words, there are different sets of `'FBAsolution.v'` values (fluxes of the reactions) and still get the same objective function value `'f'`. That is, the opitmal solution vector is not unique. This can create difficulty when investigating the changes to fluxes between two different conditions. In this case a unique solution is required to compare the changes to fluxes.

This issue can be solved with either of the following the methods

- `geometricFBA`, which provides a standard, central and reproducible solution, or
- `pFBA`, which provides a solution based on the minimal fluxes through the model, and classify each gene according to how it contributes to the optimal solution.

4. Geometric FBA

The geometric FBA solves the smallest frame that contains all sets of optimal FBA solutions and posts a set of multiple linear programming problems[4].

This FBA analysis applies iterations, where by each iteration reduces the permissible solution space. After a finite number of iterations, it resolves one single solution of the flux distribution.

```
% USAGE:
% flux = geometricFBA(model, varargin)
```

Timing:

The time to determine a geometric FBA solution depends on the size of the genome-scale model and the number of iterations. For a model with more than 10,000 reactions and several iterations takes ≥ 30minutes.

Calculating ATP energy production under anaerobic conditions using geometric FBA:

```
modelgeo = modelalter;
% For Recon3.0 model
% modelgeo = changeRxnBounds (modelgeo, 'EX_glc_D[e]', -20, 'l');
%modelgeo = changeRxnBounds (modelgeo, 'EX_o2[e]', 0, 'l');
modelgeo = changeRxnBounds(modelgeo, 'EX_glc_D[e]',-20,'l');
modelgeo = changeRxnBounds (modelgeo, 'EX_o2[e]', 0, 'l');
modelgeo = changeObjective(modelgeo, 'DM_atp_c_');
```

```
% WARNING: Depending on the size of the model running this function might
take very long;
% FBAgeo = geometricFBA (modelgeo, 'flexRel', 1e-3);
```

Display the unique fluxes from reactions, that are non-zero in the geometric FBA solution.

```
% for i=1:length(FBAgeo)
%     if abs(FBAgeo(i)) > 1e-3
%         fprintf('%1.3f \t %s\n', FBAgeo(i), modelgeo.rxns{i})
%     end
% end
```

Troubleshooting:

When the algorithm has convergence problems, change one of the optional inputs, `flexRel`, into e.g. `1e-3`. The default is 0 when there is flexibility to flux bounds.

Enter the optional parameters as parameter name followed by parameter value, for example:

`flux = geometricFBA(model, 'epsilon', 1e-9)`

5. Parsimonious enzyme usage Flux Balance Analysis (pFBA)

The pFBA method was developed to achieve higher flux levels when more enzymes are required[5].

After performing the FBA to find the optimal value for the objective function, pFBA gets the answer of an another linear program to determine the flux distribution that minimises the total flux through all metabolic reactions in the model.

Timing:

The time to determine a pFBA solution depends on the size of the genome-scale model and is taking from < 1 minute for a 1,000 reaction model, to 5 minutes for a model with more than 10,000 reactions.

The function is:

```
% [GeneClasses RxnClasses modelIrrevFM] = pFBA(model, varargin)
```

Where 'varagin' includes required inputs:

```
% * 'geneoption' - 0 = minimize the sum of all fluxes in the network,
%                  1 = only minimize the sum of the flux through
%                  gene-associated fluxes (default),
%                  2 = only minimize the sum of the flux through
%                  non-gene-associated fluxes
%
% * 'map' - map structure from readCbMap.m (no map written if empty)
%
% * 'mapoutname' - File Name for map
%
% * 'skipclass' - 0 = classify genes and reactions (default).
%                    1 = Don't classify genes and reactions. Only return
%                        model with the minimium flux set as upper bound.
```

Given outputs in this function are:

```
% OUTPUTS:
% GeneClasses:  Structure with fields for each gene class
% RxnsClasses:  Structure with fields for each reaction class
% modelIrrevFM: Irreversible model used for minimizing flux with
%               the minimum flux set as a flux upper bound
```

Calculating ATP energy production under anaerobic conditions using pFBA:

```
modelp = modelalter;
%Recon3.0 model
modelp = changeRxnBounds (modelp, 'EX_glc_D[e]', -20, 'b');
modelp = changeRxnBounds (modelp, 'EX_o2[e]', 0, 'b');
modelp = changeObjective(modelp, 'DM_atp_c_');
[GeneClasses RxnClasses modelIrrevFM] = pFBA(modelp,...
    'geneoption', 0, 'skipclass', 1)
```

```
netFlux    fluxMeasure     ->

GeneClasses =

     []


RxnClasses =

     []
modelIrrevFM = struct with fields:
                      S: [5836×14226 double]
                   mets: {5836×1 cell}
                      b: [5836×1 double]
                 csense: [5836×1 char]
                   rxns: {14226×1 cell}
                     lb: [14226×1 double]
                     ub: [14226×1 double]
                      c: [14226×1 double]
                 osense: -1
                  genes: {2248×1 cell}
                  rules: {14226×1 cell}
             metCharges: [5836×1 int64]
            metFormulas: {5836×1 cell}
              metSmiles: {5836×1 cell}
               metNames: {5836×1 cell}
              metHMDBID: {5836×1 cell}
         metInChIString: {5836×1 cell}
              metKEGGID: {5836×1 cell}
           metPubChemID: {5836×1 cell}
            description: 'Recon3DModel.mat'
                grRules: {14226×1 cell}
             rxnGeneMat: [14226×2248 double]
    rxnConfidenceScores: [14226×1 double]
               rxnNames: {14226×1 cell}
               rxnNotes: {14226×1 cell}
           rxnECNumbers: {14226×1 cell}
          rxnReferences: {14226×1 cell}
```

```
               rxnKEGGID: {14226×1 cell}
              subSystems: {14226×1 cell}
              metCHEBIID: {5836×1 cell}
                metPdMap: {5836×1 cell}
             metReconMap: {5836×1 cell}
                 modelID: 'Recon3DModel'
                  rxnCOG: {14226×1 cell}
        rxnKeggOrthology: {14226×1 cell}
             rxnReconMap: {14226×1 cell}
                 version: 'Recon3D_01'
              PleaseCite: 'Brunk et al, Nat Biotech, 2018; doi:10.1038/nbt.4072'
                   match: [14226×1 double]
         reversibleModel: 0
```

Display minimal fluxes of the reactions that are required for producing energy only from only glucose media. This is convenient with the `printFluxVector` function.

```
nonZeroFlag = 1;
printFluxVector(modelIrrevFM, modelIrrevFM.lb, nonZeroFlag)
```

```
DM_atp_c_                        1000
EX_glc_D[e]_r                      20
netFlux                     3.192e+04
```

6. Dynamic FBA

The dynamic FBA is an extension of standard FBA that accounts for cell culture dynamics, implementing both dynamic (nonlinear programming) and static (LP) optimisation of an objective function and applying constraints to the rates of change of flux in addition to the standard FBA constraints[6].

The dynamic FBA method implemented in this function is essentially the same as the method described by Varma A. and B. O. Palsson[7].

```
modeldinamic = model;
% For Recon3.0 model
% modeldinamic = changeRxnBounds (modeldinamic, 'EX_glc_D[e]', -20, 'l');
% modeldinamic = changeRxnBounds (modeldinamic, 'EX_o2[e]', -1000, 'l');
% modeldinamic = changeRxnBounds (modeldinamic, 'EX_ac[e]', -1000, 'l');

modeldinamic = changeRxnBounds (modeldinamic, 'EX_glc_D[e]', -20, 'b');
modeldinamic = changeRxnBounds (modeldinamic, 'EX_o2[e]', -1000, 'l');
modeldinamic = changeRxnBounds (modeldinamic, 'EX_ac[e]', -1000, 'l');
% For Recon3.0 model
% smi = {'EX_glc_D[e]' 'EX_ac[e]'};
smi = {'EX_glc_D[e]' 'EX_ac[e]'};
% exchange reaction for substrate in environment

smc = [10.8]; % Glucose, Acetate concentration (all in mM)

Xec = 0.001; % initial biomass
dt = 1.0/1000.0; % time steps
time = 1.0/dt; % simulation time
```

```
[concentrationMatrix, excRxnNames, timeVec,...
    biomassVec] = dynamicFBA(modeldinamic, smi, smc, Xec, dt, time, smi );
```

```
ans = 1806
Step number    Biomass
Dynamic FBA analysis in progress ...
0%      [                                        ]
```

7. Relaxed FBA

Find the minimal set of relaxations on bounds and steady-state constraint to make an infeasible FBA problem feasible.

See tutorial_relaxedFBA.mlx

8. Flux enrichment analysis (FEA)

The flux enrichment analysis calculates the likelihood that a set of fluxes would belong to a subsystem or pathway.

Timing:

The time to calculate the FEA is < 1 second for any size of a model.

```
modelfea = model;
res = optimizeCbModel(modelfea,'max');
% say you are interested in enriching the active reactions
activeReactions = find(res.x)
% You can also look for e.g. positive/negative/zeros flux reactions,
% that depends pretty much on the question.
% Now you look for the enrichement of reactions per subsystems
resultCell = FEA(modelfea, activeReactions, 'subSystems')
```

REFERENCES

[1] Orth, J. D., Thiele I., and Palsson, B. Ø. What is flux balance analysis? *Nat. Biotechnol.,* 28(3), 245–248 (2010).

[2] Thiele, I., et al. A community-driven global reconstruction of human metabolism. *Nat. Biotechnol.,* 31(5), 419–425 (2013).

[3] Benyamini, T, Folger, O., Ruppin, E., Schlomi, T. Flux balance analysis accounting for metabolite dilution. *Genome Biology*., 11(4):R43 (2010).

[4] Smallbone, K., and Simeonidis, E. Flux balance analysis: A geometric perspective. *J Theor Biol*., 258: 311-315 (2009).

[5] Lewis, N.E., et al. Omic data from evolved E. coli are consistent with computed optimal growth from genome-scale models. *Mol Syst Biol*., 6:390 (2010).

[6] Mahadevan, R., Edwards, J.S., Doyle, F.J. Dynamic Flux Balance Analysis of Diauxic Growth in Escherichia coli. *Biophys J.,* 83(3):1331-1340 (2002).

[7] Varma A. and Palsson, B. Ø. Stoichiometric flux balance models quantitatively predict growth and metabolic by-product secretion in wild-type Escherichia coli W3110. *App Environ Microbiol*., 60(10):3724-3731 (1994).

[8]. Laurent Heirendt & Sylvain Arreckx, Thomas Pfau, Sebastian N. Mendoza, Anne Richelle, Almut Heinken, Hulda S. Haraldsdottir, Jacek Wachowiak, Sarah M. Keating, Vanja Vlasov, Stefania Magnusdottir, Chiam Yu Ng, German Preciat, Alise Zagare, Siu H.J. Chan, Maike K. Aurich, Catherine M. Clancy, Jennifer Modamio, John T. Sauls, Alberto Noronha, Aarash Bordbar, Benjamin Cousins, Diana C. El Assal, Luis V. Valcarcel, Inigo Apaolaza, Susan Ghaderi, Masoud Ahookhosh, Marouen Ben Guebila, Andrejs Kostromins, Nicolas Sompairac, Hoai M. Le, Ding Ma, Yuekai Sun, Lin Wang, James T. Yurkovich, Miguel A.P. Oliveira, Phan T. Vuong, Lemmer P. El Assal, Inna Kuperstein, Andrei Zinovyev, H. Scott Hinton, William A. Bryant, Francisco J. Aragon Artacho, Francisco J. Planes, Egils Stalidzans, Alejandro Maass, Santosh Vempala, Michael Hucka, Michael A. Saunders, Costas D. Maranas, Nathan E. Lewis, Thomas Sauter, Bernhard Ø. Palsson, Ines Thiele, Ronan M.T. Fleming, **Creation and analysis of biochemical constraint-based models: the COBRA Toolbox v3.0**, Nature Protocols, volume 14, pages 639–702, 2019 doi.org/10.1038/s41596-018-0098-2.