Dynamic Flux Balance Analysis using DFBAlab

Authors:

Jose Alberto Gomez

Prof. Paul I. Barton pib@mit.edu

Process Systems Engineering Laboratory

77 Massachusetts Avenue

Cambridge, MA, USA 02139

Dynamic Flux Balance Analysis using DFBAlab

Abstract

Bioprocesses are of critical importance in several industries such as the food and pharmaceutical industries. Despite their importance and widespread application, bioprocess models remain rather simplistic and based on unstructured models. These simple models have limitations, making it very difficult to model complex bioprocesses. With dynamic flux balance analysis (DFBA) more comprehensive bioprocess models can be obtained. DFBA simulations are difficult to carry out because they result in dynamic systems with linear programs embedded. Therefore, the use of DFBA as a modeling tool has been limited. With DFBAlab, a MATLAB code that performs efficient and reliable DFBA simulations, the use of DFBA as a modeling tool has become more accessible. Here, we illustrate with an example how to implement bioprocess models in DFBAlab.

Keywords: Dynamic Flux Balance Analysis, Flux Balance Analysis, Bioprocess modeling, Linear Programming, Dynamic Systems

1. Introduction

Dynamic Flux Balance Analysis (DFBA) [1], [2] is a bioprocess modeling framework that relies on genome-scale metabolic network reconstructions (GENREs) of microorganisms. It is the dynamic extension of Flux Balance Analysis (FBA) [3], which has become popular with the advent of high-throughput genome sequencing. In fact, the number and level of detail of genome-scale metabolic network reconstructions has rapidly increased since 1999 (see Figure 1 in [4]). Despite the ever increasing availability of new and better metabolic network reconstructions, DFBA modeling remains challenging, and therefore its use has been limited.

Traditionally, bioprocess modeling relies on unstructured models to calculate the growth rates of microorganisms. This approach has significant limitations that make it impossible to simulate very complex bioprocesses. These limitations are countered by FBA by considering genome-scale metabolic networks of the microorganisms involved. FBA models the growth and metabolic fluxes rates of microorganisms as solutions of the following linear program (LP):

$$\begin{aligned} &\max v_{growth} \\ &\text{s.t.} \quad \mathbf{Sv} = \mathbf{0}, \\ &\qquad \mathbf{v}^{LB} \le \mathbf{v} \le \mathbf{v}^{UB}, \end{aligned}$$

where $\mathbf{S}$ is the stoichiometric matrix, $\mathbf{v}$ is the fluxes vector, $\mathbf{v}^{UB}$ and $\mathbf{v}^{LB}$ are bounds on the metabolic fluxes given by thermodynamics, the extracellular environment and/or genetic modifications, and v_{growth} sums all growth associated fluxes. Given mass balance and thermodynamic constraints on a metabolic network, FBA finds a solution that satisfies these constraints and maximizes growth. If the LP becomes infeasible, it may indicate a lack of sufficient substrates and nutrients to provide the minimum maintenance energy for the respective microorganism to survive.

DFBA combines process models described by an ordinary differential equation (ODE) system, a differential-algebraic equation (DAE) system or a partial differential-algebraic equation (PDAE) [5], [6] system with FBA to model bioprocesses. These models can be expressed as dynamic systems with LPs embedded [7], [8] which are challenging to simulate. The embedded LP poses difficulties in the form of non-unique solutions and premature LP infeasibilities. Fortunately, these complications have been addressed by efficient DFBA simulators, DSL48LPR in FORTRAN [7] and DFBAlab in MATLAB [9]. Both are free for academic research and can be found in the following webpage: http://yoric.mit.edu/software. The rest of this chapter will talk exclusively about how to use DFBAlab to perform DFBA simulations.

DFBAlab uses lexicographic optimization and the phase I of the simplex algorithm to deal with nonunique solutions and LP infeasibilities, respectively. Lexicographic optimization is a strategy that enables obtaining unique exchange fluxes. This strategy requires defining an objective function for each exchange flux of interest. More information can be found in [9] and [7].

2. Materials

1. MATLAB: DFBAlab is compatible with MATLAB 7.12.0 and newer versions.
2. DFBAlab: DFBAlab is a MATLAB code that performs DFBA simulations efficiently and reliably. It is free for academic research and can be downloaded at http://yoric.mit.edu/software. DFBAlab contains three folders named 'Functions', 'Examples_Direct', and 'Examples_DAE'. The folder named 'Functions' must be added to the MATLAB path. The Examples folders contain the examples described in [9]. It is recommended to use 'Examples_DAE' since these examples run much faster than 'Examples_Direct'.
3. Relevant GENREs: To perform DFBA simulations, GENREs of the relevant microorganisms are needed. An extensive collection of GENREs can be found at the following webpage: http://systemsbiology.ucsd.edu/InSilicoOrganisms/OtherOrganisms. There are some protocols and methods that generate these reconstructions automatically. A list can be found at Table 1 of [10]. GENREs are usually published in the Systems Biology Markup Language (SBML) [11].
4. LP Solvers Gurobi or CPLEX: As mentioned before, DFBA calculates the growth rates and exchange fluxes rates of microorganisms by solving LPs. Therefore, LP solvers are needed. DFBAlab is currently compatible with CPLEX [12] and Gurobi [13]. Academic licenses are available for both LP solvers.
5. Cobra Toolbox: The Cobra Toolbox [14] is a package that runs in MATLAB that uses constraint-based modeling to predict the behavior or microorganisms. It can be obtained from https://opencobra.github.io/. The Cobra toolbox has a function that transforms a metabolic network

in SBML format into a '.mat' format, which is the input used by DFBAlab. For this function to work, the SBML package must be obtained from http://sbml.org/Main_Page.

3. Methods

Here, we describe step by step how to model a DFBA problem using DFBAlab. The examples in 'Examples_Direct' and 'Examples_DAE' require the same modeling effort. Only the implementation in 'Examples_DAE' will be discussed because the implementation in 'Examples_Direct' is very similar. Examples in both folders contain three key files: main.m, DRHS.m, and RHS.m. In addition, the examples in 'Examples_DAE' contain a file called evts.m. In addition, the Examples folders contain a folder called 'Example-ModelImport'. This folder contains a very simple script that converts an SBML file into a '.mat' file that can be used in MATLAB simulations. Next, we show the relevant parts of these files and the inputs required from the user. In addition, we use Example 3 in [9] to illustrate the use of DFBAlab. In this example, *Chlamydomonas reinhardtii* and *Saccharomyces cerevisiae* grow together in a pond open to the atmosphere. For simplicity, this pond is modeled as a single continuously stirred-tank reactor (CSTR). The model results in a DAE system as pH balances are considered. In the Notes section, Notes 1 and 2 describe some common problems regarding the use of DFBAlab.

a. Converting a GENRE in SBML format into '.mat' format

When published, GENREs are usually in SBML format. The file 'ModelImport.m' in the 'Example-ModelImport' folder in Examples calls the Cobra toolbox and converts GENREs in SBML format into '.mat' format. This function is really easy to use. Just modify line 4 of the code,

```
NAME1=readCBModel('NAME2',DB,'SBML')
```

and replace NAME1 for the name you want to give your model, replace NAME2 with the name of the SBML file, and replace DB with the numerical value you want infinity to be replaced with. Usually, a value of DB = 1000 works fine. The output will be a model with the name 'NAME1.mat'. Every microorganism that needs to be modeled in DFBAlab must be transformed to a '.mat' file.

b. Inputs for the main.m file

The 'main.m' file sets up the simulation. In this section, each part of the 'main.m' file will be described and the required inputs will be specified. It is important to notice that the structure variable INFO contains important information that is then passed to other MATLAB files and functions. This structure can be used to pass other important parameter information to 'DRHS.m', 'RHS.m', and 'evts.m'. The first part of main.m clears the workspace and specifies the number of models that will be used in the simulation.

```
clear all
INFO.nmodel = NM;
```

Here, NM stands for the number of models in the simulation. For example, consider a dynamic model of a plug flow reactor (PFR) with two different species growing. The PFR can be discretized in N spatial slices to transform it from a partial differential equation (PDE) system to an ODE system. Then, $NM = 2*N$. The next part of the 'main.m' file loads the models. Here, the models must already be in '.mat' format after going through the 'ModelImport.m' code:

```
load NAMEMODEL.mat
model{i} = NAMEMODEL;
DB(i) = db;
INFO.DB = DB;
```

Each species must be loaded into the file using the load command and replacing NAMEMODEL for the actual name of the model. Then, for $i = 1, \ldots, nmodel,$ the cell model stores each relevant metabolic

network model. The array DB stores the default bound specified in 'ModelImport.m' for each model. In this way, DFBAlab can change this bound information back to infinity. This array is stored in the INFO structure. The next part defines the *exID* cell, which carries the exchange fluxes information:

```
exID{i}=[ex1,ex2,…,exj];
INFO.exID = exID;
```

Again $i = 1, \ldots, nmodel$, and *ex1*, *ex2*, *exj* correspond to the indices of the exchange fluxes that are needed in the 'DRHS.m' file for model i. For example, let us assume that model 1 corresponds to an *E. coli* model. To model this bioprocess, we may care about the exchange fluxes of glucose, oxygen, and carbon dioxide. Then in $model\{1\}.rxns$ the indices associated to the exchange fluxes reactions for glucose, oxygen, and carbon dioxide must be found. These are the indices that must be inserted in place of *ex1*, *ex2*, *exj*. The order of these indices is relevant for the 'RHS.m' file as explained later.

Next, the cell C corresponding to the cost vectors is defined:

```
minim = 1;
maxim = -1;
% Maximize growth
C{i}(j).sense = maxim;
C{i}(j).rxns = [cin];
C{i}(j).wts = [1];
```

DFBAlab relies on lexicographic optimization to perform efficient DFBA simulations [9]. Each cost vector requires three entries: sense, reactions and weights. The information in $C\{i\}(j)$ corresponds to model i and cost vector j. Usually, the first cost vector for any model will correspond to maximization of biomass. Other relevant biological objectives can be listed as subsequent cost vectors. All fluxes that appear in the right-hand side of the ODE system must be listed as cost vectors in order to guarantee a unique solution of the simulation. If you want the corresponding cost vector to be maximized, enter *maxim* in sense, otherwise enter *minim*. Then in $C\{i\}(j).rxns$ list the indices of the reactions

corresponding to this cost vector. Finally in $C\{i\}(j).wts$ enter the coefficients corresponding to this cost vector. For example,

```
C{i}(j).sense = maxim;
C{i}(j).rxns = [1108,103];
C{i}(j).wts = [1,-1];
INFO.C = C;
```

is equivalent to defining cost vector j for model i as *maximize* $v_{1108} - v_{103}$. The cost vectors are stored in the INFO structure. Next the initial conditions, integration time and DFBAlab options are set.

```
Y0 = [ICVECTOR];
% Time of simulation
tspan = [ti,tf];

% LP Objects construction parameters
INFO.LPsolver = 0; % CPLEX = 0, Gurobi = 1.
INFO.tol = tol1; % Feasibility, optimality and convergence tolerance for
Cplex (tol>=1E-9).
                 % It is recommended it is at least 2 orders of magnitude
                 % tighter than the integrator tolerance.
                 % If problems with infeasibility messages, tighten this
                 % tolerance.
INFO.tolPh1 = tol2; % Tolerance to determine if a solution to phaseI equals
zero.
                   % It is recommended to be the same as INFO.tol.
INFO.tolevt = tol3; % Tolerance for event detection. Has to be greater
                   % than INFO.tol. Recommended to be 2 times the integration
tolerance.
```

$Y0$ is a column vector containing the initial conditions. *tspan* contains the integration time interval: *ti* corresponds to the initial time and *tf* corresponds to the final time. Next, some DFBAlab parameters are established: *INFO.LPsolver* selects between Gurobi (enter 1) and CPLEX (enter 0). Next, the LP tolerance is set at *INFO.tol*. This tolerance has to be larger than 10^{-9}. Next, *INFO.tolPh1* corresponds to the threshold value under which the penalty function will be considered equal to zero. Here, we recommend using *INFO.tol* for this value. Finally, *INFO.tolevt* corresponds to the event detection

tolerance that triggers when the FBA LP must be solved again. Here, we recommend using two times the absolute tolerance of the integration method. Next, the integration options are set:

```
M = [MASS];
options = odeset('AbsTol',tol4,'RelTol',tol5,'Mass',M,'Events',@evts);
```

If we are integrating a DAE system, a mass matrix must be defined. Using odeset, other MATLAB integration options can be set such as absolute and relative tolerances, nonnegativity constraints, and event detection. DFBAlab always uses event detection if you use the Examples in 'Examples_DAE'. The following parts of the 'main.m' file do not require any further inputs.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[model,INFO] = ModelSetupM(model,Y0,INFO);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if INFO.LPsolver == 0
    [INFO] = LexicographicOpt(model,INFO);
elseif INFO.LPsolver == 1
    [INFO] = LexicographicOptG(model,INFO);
else
    display('Solver not currently supported.');
end

tic
tint = 0;
TF = [];
YF = [];
while tint<tspan(2)
% Look at MATLAB documentation if you want to change solver.
% ode15s is more or less accurate for stiff problems.
    [T,Y] = ode15s(@DRHS,tspan,Y0,options,INFO);
    TF = [TF;T];
    YF = [YF;Y];
    tint = T(end);
    tspan = [tint,tspan(2)];
    Y0 = Y(end,:);
    if tint == tspan(2)
        break;
    end

% Update b vector
[INFO] = bupdate(tint,Y0,INFO);
%Determine model with basis change
    value = evts(tint,Y0,INFO);
    ind = find(value<=0);
    fprintf('Basis change at time %d. ',tint);
```

```
    k = 0;
    ct = 0;
    while ~isempty(ind)
        k = k + 1;
        ct = ct + size(model{k}.A,1);
        ind2 = find(ind<=ct);
        if ~isempty(ind2)
           INFO.flagbasis = k;
           fprintf('Model %i. \n',k);
           % Perform lexicographic optimization
           if INFO.LPsolver == 0
               [INFO] = LexicographicOpt(model,INFO);
           elseif INFO.LPsolver == 1
               [INFO] = LexicographicOptG(model,INFO);
           else
               display('Solver not currently supported.');
           end
           ind(ind2)=[];
        end
    end
end

display(toc);
```

The function ModelSetupM takes the model cell and the $INFO$ structure to transform the LPs into standard form, which is key for efficient integration with DFBAlab. The functions LexicographicOpt and LexicographicOptG solve the LPs with CPLEX and Gurobi, respectively. Then comes the integration loop using the numerical integrator ode15s, used for stiff systems which is the case for DFBA systems. The results of integration are stored in vectors YF for the states and TF for the times. After this, any plots can be generated using the information in YF and TF.

c. Example inputs for main.m

The following MATLAB code is commented in the relevant sections.

```
clear all
INFO.nmodel = 2; % Number of models: one for algae and one for yeast.
load iND750.mat % This is the yeast model
model{1} = iND750;
DB(1) = 1000; % This is the default bound for the yeast model (1000 =
Infinity)
load iRC1080.mat % This is the algae model
model{2} = iRC1080;
DB(2) = 1000; % This is the default bound for the algae model (1000 =
Infinity)
INFO.DB = DB;
```

```
exID{1}=[428,458,407,420]; % These are the exchange fluxes with variable
bounds for the yeast model. The bounds are defined in RHS.m.
exID{2}=[26,28,24,25,1,2,3,4,5,6,7,8,9,10,11,12,13,62,63,64,65,27,81,47]; %
These are the exchange fluxes with variable bounds for the algae model. The
bounds are defined in RHS.m.
INFO.exID = exID;

% Next, we define the cost vectors for yeast and algae. For both cases,
first, we maximize growth, and then we maximize or minimize each one of the
exchange fluxes that appear in the ODE system.
minim = 1;
maxim = -1;
% Yeast
% Maximize growth
C{1}(1).sense = maxim;
C{1}(1).rxns = [1266];
C{1}(1).wts = [1];
% Glucose
C{1}(2).sense = maxim;
C{1}(2).rxns = [428];
C{1}(2).wts = [1];
% O2
C{1}(3).sense = maxim;
C{1}(3).rxns = [458];
C{1}(3).wts = [1];
% CO2
C{1}(4).sense = maxim;
C{1}(4).rxns = [407];
C{1}(4).wts = [1];
% Ethanol
C{1}(5).sense = maxim;
C{1}(5).rxns = [420];
C{1}(5).wts = [1];

% Algae
% Maximize growth
C{2}(1).sense = maxim;
C{2}(1).rxns = [63];
C{2}(1).wts = [1];
% Acetate
C{2}(2).sense = minim;
C{2}(2).rxns = [28];
C{2}(2).wts = [1];
% O2
C{2}(3).sense = maxim;
C{2}(3).rxns = [24,81];
C{2}(3).wts = [1,1];
% CO2
C{2}(4).sense = minim;
C{2}(4).rxns = [25];
C{2}(4).wts = [1];

INFO.C = C;

% Initial conditions
% Y1 = Volume (L)
```

```
% Y2 = Biomass Yeast (gDW/L)
% Y3 = Biomass Algae (gDW/L)
% Y4 = Glucose (mmol/L)
% Y5 = O2 (mmol/L)
% Y6 = Total Carbon (mmol/L)
% Y7 = Ethanol (mmol/L)
% Y8 = Acetate (mmol/L)
% Y9 = Total Nitrogen (mmol/L)
% Y10 = Penalty
% Y11 = NH4+
% Y12 = NH3
% Y13 = CO2
% Y14 = HCO3-
% Y15 = CO3 -2
% Y16 = H +

Y0 = [140 1.1048 1.8774 0.0140, 0.00065156 1.2211 8.2068 0.0237 0.1643 0
0.1643 2.4476E-5 1.0568 0.1643 2.5842E-6 2.6701E-6]';

% Time of simulation
tspan = [0,24];

% LP options and tolerances
INFO.LPsolver = 0; % CPLEX = 0, Gurobi = 1.
INFO.tol = 1E-9; % Feasibility, optimality and convergence tolerance for LP
solver (tol>=1E-9). It is recommended it is at least 1 order of magnitude
tighter than the integration tolerance.
INFO.tolPh1 = INFO.tol; % Tolerance to determine if a solution to phaseI
equals zero. Usually, the best value here is to set it equal to INFO.tol.
INFO.tolevt = 2E-6; % Tolerance for event detection. Has to be greater than
INFO.tol. Usually a value of two times the integration tolerance works fine.

% This is a DAE system because of the pH balances. The first 10 states are
the differential states and the last 6 correspond to the algebraic states.
M = [eye(10) zeros(10,6); zeros(6,16)];
options = odeset('AbsTol',1E-6,'RelTol',1E-6,'Mass',M,'Events',@evts);

% This part of the code constructs the LP problem structures that will be
solved during integration, and solves the LPs at the initial conditions.
[model,INFO] = ModelSetupM(model,Y0,INFO);
if INFO.LPsolver == 0
    [INFO] = LexicographicOpt(model,INFO);
elseif INFO.LPsolver == 1
    [INFO] = LexicographicOptG(model,INFO);
else
    display('Solver not currently supported.');
end

% This is the integration loop.
tint = 0;
% TF and YF will concatenate T and Y that are returned by the DAE numerical
integrator.
TF = [];
YF = [];
while tint<tspan(2)
```

```
% Look at MATLAB documentation if you want to change solver. DFBA systems
tend to be stiff systems and ode15s is more or less accurate for stiff
problems.
    [T,Y] = ode15s(@DRHS,tspan,Y0,options,INFO);
    TF = [TF;T];
    YF = [YF;Y];
    tint = T(end);
    tspan = [tint,tspan(2)];
    Y0 = Y(end,:);
    if tint == tspan(2)
        break;
    end

% Update the right-hand sides of the LPs given the current time and states.
[INFO] = bupdate(tint,Y0,INFO);
% Determine which LPs had a basis change
    value = evts(tint,Y0,INFO);
    ind = find(value<=0);
    fprintf('Basis change at time %d. ',tint);
    k = 0;
    ct = 0;
    while ~isempty(ind)
        k = k + 1;
        ct = ct + size(model{k}.A,1);
        ind2 = find(ind<=ct);
        if ~isempty(ind2)
           INFO.flagbasis = k;
           fprintf('Model %i. \n',k);
           % Perform lexicographic optimization
           if INFO.LPsolver == 0
               [INFO] = LexicographicOpt(model,INFO);
           elseif INFO.LPsolver == 1
               [INFO] = LexicographicOptG(model,INFO);
           else
              display('Solver not currently supported.');
           end
           ind(ind2)=[];
        end
    end
end
% TF contains all the times and YF all the states at these times. Any
plotting options can be included here.
```

d. Inputs for the DRHS.m file

The DRHS function defined in 'DRHS.m' takes time, states and the INFO structure and returns the right-hand side vector of the ODE or DAE system:

```
function dy = DRHS(t, y, INFO).
```

This file is very flexible. A key command that takes place before the right-hand side values are set is:

```
[flux,penalty] = solveModel(t,y,INFO);
```

Here, the flux variable is a matrix with rows corresponding to each model and columns corresponding to each cost vector. Therefore, *flux(i,j)* corresponds to the optimal value of cost vector *j* and model *i* with the order defined in 'main.m'. The penalty vector contains the objective function values of the Phase I LPs. If *penalty(i)*>0, model *i* corresponds to an infeasible LP. Otherwise, model *i* is feasible. We recommend that a penalty state that integrates the sum of all penalty functions is set. If this penalty state is greater than zero at the end of the simulation, then, this DFBA simulation is infeasible.

e. Example inputs for the DRHS.m file

The following MATLAB code is commented in the relevant sections.

```
function dy = DRHS(t, y, INFO)

% Y1 = Volume (L)
% Y2 = Biomass Yeast (gDW/L)
% Y3 = Biomass Algae (gDW/L)
% Y4 = Glucose (mmol/L)
% Y5 = O2 (mmol/L)
% Y6 = Total Carbon (mmol/L)
% Y7 = Ethanol (mmol/L)
% Y8 = Acetate (mmol/L)
% Y9 = Total Nitrogen (mmol/L)
% Y10 = Penalty
% Y11 = NH4+
% Y12 = NH3
% Y13 = CO2
% Y14 = HCO3-
% Y15 = CO3 -2
% Y16 = H +

% Assign values from states
Vol = y(1);
X(1) = y(2);
X(2) = y(3);
for i=1:13
   S(i) = y(3+i);
end
```

```
% Feed rates
Fin = 1;
Fout = 1;

% Biomass Feed concentrations
Xfeed(1) = 0;
Xfeed(2) = 0;

% Mass transfer coefficients
KhO2 = 0.0013;
KhCO2 = 0.035;

% Mass transfer expressions
MT(1) = 0;
MT(2) = 0.6*(KhO2*0.21*1000 - S(2));
MT(3) = 0.58*(KhCO2*0.00035*1000 - S(10));
MT(4) = 0;
MT(5) = 0;
MT(6) = 0;

% Substrate feed concentrations
Sfeed(1) = 15.01;
Sfeed(2) = KhO2*0.21*1000;
Sfeed(3) = KhCO2*0.00035*1000;
Sfeed(4) = 0;
Sfeed(5) = 40;
Sfeed(6) = 0;

% The elements of the flux matrix have the sign given to them by the
% coefficients in the Cost vector in main.
% Example, if:
% C{k}(i).rxns = [144, 832, 931];
% C{k}(i).wts = [3, 1, -1];
% Then the cost vector for this LP will be:
% flux(k,i) = 3*v_144 + v_832 - v_931

%% Update bounds and solve for fluxes
[flux,penalty] = solveModel(t,y,INFO);

% Yeast fluxes
for i=1:4
    v(1,i) = flux(1,i+1);
end
    v(1,5) = 0;
    v(1,6) = 0;

% Algae fluxes
    v(2,1) = 0;
    v(2,2) = flux(2,3);
    v(2,3) = flux(2,4);
    v(2,4) = 0;
    v(2,5) = flux(2,2);

%% Dynamics
```

```
dy(1) = Fin-Fout;    % Volume
dy(2) = flux(1,1)*y(2) + (Xfeed(1)*Fin - y(2)*Fout)/y(1);    % Biomass yeast
dy(3) = flux(2,1)*y(3) + (Xfeed(2)*Fin - y(3)*Fout)/y(1);    % Biomass algae
for i = 1:5
    dy(i+3) = v(1,i)*X(1) + v(2,i)*X(2) + MT(i) + (Sfeed(i)*Fin -
S(i)*Fout)/y(1) ;
end

if (S(2)/1000 > KhO2 && dy(3+2)>0)
   dy(3+2) = 0;
end

if (S(3)/1000 > KhCO2 && dy(3+3)>0)
   dy(3+3) = 0;
end

dy(9) = 0; % Leave total nitrogen constant
dy(10) = penalty(1) + penalty(2); %Penalty function

% Algebraic Equations
Ka = 10^-9.4003;
K1c = 10^-6.3819;
K2c = 10^-10.3767;
Nt = S(6);
Ct = S(3);

x = y(11:16);
F = [-x(1) + Nt*x(6)/(x(6) + Ka) ;
      -x(4) + x(1)/(1 + 2*K2c/x(6));
      -x(3) + x(6)*x(4)/K1c;
      -x(5) + x(4)*K2c/x(6);
      -Nt + x(1) + x(2)
      -Ct + x(3) + x(4) + x(5)];
dy(11:16) = F;

end
```

f. Inputs for the RHS.m file

The RHS function defined in 'RHS.m' takes time, states and the INFO structure and returns two matrices containing the upper and lower bounds for the fluxes specified in the *exID* cell in 'main.m'.

```
function [lb,ub] = RHS( t,y,INFO )
```

Here, *lb* corresponds to the lower bounds and *ub* to the upper bounds. Element *lb(i,j)* contains the lower bound corresponding to flux *j* in *exID{i}*. The same indexing applies for the *ub* matrix containing the upper bounds. The lower and upper bound quantities are functions of time and states. These functions must be continuous functions. In addition, if any lower bound or upper bound is defined as infinity or minus infinity, the value of this bound must remain constant the entire simulation.

g. Example inputs for the RHS.m file

```
function [lb,ub] = RHS( t,y,INFO )

% Y1 = Volume (L)
% Y2 = Biomass Yeast (gDW/L)
% Y3 = Biomass Algae (gDW/L)
% Y4 = Glucose (mmol/L)
% Y5 = O2 (mmol/L)
% Y6 = Total Carbon (mmol/L)
% Y7 = Ethanol (mmol/L)
% Y8 = Acetate (mmol/L)
% Y9 = Total Nitrogen (mmol/L)
% Y10 = Penalty
% Y11 = NH4+
% Y12 = NH3
% Y13 = CO2
% Y14 = HCO3-
% Y15 = CO3 -2
% Y16 = H +

% This subroutine updates the upper and lower bounds for the fluxes in the
% exID arrays in main. The output should be two matrices, lb and ub. The lb
matrix
% contains the lower bounds for exID{i} in the ith row in the same order as
% exID. The same is true for the upper bounds in the ub matrix.
% Infinity can be used for unconstrained variables, however, it should be
% fixed for all time.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Yeast bounds
% Glucose
if (y(4)<0)
    lb(1,1) = 0;
else
    lb(1,1) = -(20*y(4)/(0.5/0.18 + y(4)))*1/(1+y(7)/(10/0.046));
end
ub(1,1) = 0;
% Oxygen
lb(1,2) = -8*y(5)/(0.003/0.016 + y(5));
ub(1,2) = 0;
% CO2
lb(1,3) = 0;
```

```
ub(1,3) = Inf;
% Ethanol
lb(1,4) = 0;
ub(1,4) = Inf;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Algae bounds
% HCO3
lb(2,1) = 0;
ub(2,1) = 0;
% Acetate
lb(2,2) = -14.9*y(8)/(2.2956+y(8)+y(8)^2/0.1557);
ub(2,2) = 0;
% Oxygen
lb(2,3) = -1.41750070911*y(5)/(0.009+y(5));
ub(2,3) = Inf;
% CO2
lb(2,4) = -2.64279793224*y(13)/(0.0009+y(13));
ub(2,4) = Inf;
% Light
Ke1 = 0.32;
Ke2 = 0.03;
L = 0.4; % meters depth of pond
Ke = Ke1 + Ke2*(y(3)+y(2)/2);
Io = 28*(max(sin(2*pi()*t/48)^2,sin(2*pi()*5/48)^2)-sin(2*pi()*5/48)^2)/(1-
sin(2*pi()*5/48)^2);
lb(2,5) = 0;
ub(2,5) = Io*(1-exp(-L*Ke))/(Ke*L);
% Other possible light sources set equal to zero
for i=6:16
   lb(2,i) = 0;
   ub(2,i) = 0;
end
% H+
lb(2,17) = -10;
ub(2,17) = Inf;
% Autotrophic growth
lb(2,18) = 0;
ub(2,18) = 0;
% Mixotrophic growth
lb(2,19) = 0;
ub(2,19) = Inf;
% Heterotrophic growth
lb(2,20) = 0;
ub(2,20) = 0;
% Non-growth associated ATP maintenance
lb(2,21) = 0.183;
ub(2,21) = 0.183;
% Starch
lb(2,22) = 0;
ub(2,22) = 0;
% Photoevolved oxygen
lb(2,23) = 0;
ub(2,23) = 8.28;
% Ethanol
lb(2,24) = 0;
ub(2,24) = 0;
end
```

h. Inputs for the evts.m file

This file is critical for DFBAlab to perform efficient DFBA simulations and will most likely not require any changes. This file needs to be changed only if event detection is needed in addition to the event detection associated with the LPs embedded. In this case, any event detection conditions can be added at the end of the vectors *value*, *isterminal*, and *direction*. The definition of these vectors can be found in the MATLAB documentation.

```
function [value,isterminal,direction] = evts(t,y,INFO)
eps = INFO.tolevt;
lexID = INFO.lexID;
nmodel = INFO.nmodel;
bmodel = INFO.b;
lbct = INFO.lbct;
indlb = INFO.indlb;
indub = INFO.indub;
U = INFO.U;
L = INFO.L;
P = INFO.P;
Q = INFO.Q;
%% Update solutions
[lbx,ubx] = RHS( t,y,INFO );
ct = 0;
total = 0;
for i=1:nmodel
   total = length(bmodel{i}) + total;
end
value = zeros(total,1);
isterminal = ones(total,1); % stop the integration
direction = -1*ones(total,1); % negative direction

for i=1:nmodel
    b = bmodel{i};
    lb = lbx(i,1:lexID(i));
    ub = ubx(i,1:lexID(i));
    lb(indlb{i}) = [];
    ub(indub{i})=[];
    b(1:length(lb)) = lb;
    b(length(lb)+lbct(i)+1:length(lb)+lbct(i)+length(ub)) = ub;
    x = (L{i}\(P{i}*b));
    x = U{i}\x;
    x = Q{i}*x;
%     x = U{i}\(L{i}\(P{i}*b));
%     x = INFO.B{i}\b;
% Detect when a basic variable crosses zero.
    value(1+ct:length(x)+ct) = x + eps;
```

```
    ct = ct + length(x);
end
ADD NEW CONDITIONS HERE
end
```

If the code is not modified, the length of the vectors *value*, *isterminal*, and *direction* is equal to the sum of all m_i where m_i corresponds to the number of rows of $model\{i\}.A$. In addition, the 'main.m' file needs to be modified at the integration loop to distinguish events due to basis changes in the LPs from other types of events. In Example 3 in [9], this file is not modified.

4. Notes

Note 1: In some instances, DFBAlab may fail due to infeasible or unbounded LPs. Although theoretically this shouldn't happen, numerically it does happen some times. When encountering this problem, change INFO.tol a bit and/or change the LP solver and try again.

Note 2: DFBAlab is designed such that only the fluxes defined as cost vectors can be accessed at DRHS.m. This is to ensure uniqueness in the right-hand side of the ODE/DAE system. Future versions of DFBAlab may contain the option of extracting all other fluxes, although these other fluxes must be treated carefully as they can be nonunique.

Bibliography

[1] Varma A, Palsson BØ (1994) Stoichiometric flux balance models quantitatively predict growth and metabolic by-product secretion in wild-type Escherichia coli W3110. Applied and Environmental Microbiology 60(10):3724-3731

[2] Mahadevan R, Edwards J, Doyle FJ III (2002) Dynamic flux balance analysis of diauxic growth in Escherichia coli. Biophysical Journal 83(3):1331-40

[3] Orth JD, Thiele I, Palsson BØ (2010) What is flux balance analysis? Nature Biotechnology 28:245-248

[4] Monk J, Nogales J, Palsson BØ (2014) Optimizing genome-scale network reconstructions. Nature Biotechnology 32:447-452

[5] Chen J, Gomez JA, Höffner K, Phalak P, Barton PI, Henson MA (2016) Spatiotemporal modeling of microbial metabolism. BMC Systems Biology 10(21)

[6] Chen J, Gomez JA, Höffner K, Barton PI, Henson MA (2015) Metabolic modeling of synthesis gas fermentation in bubble column reactors. Biotechnology for Biofuels 8(89)

[7] Höffner K, Harwood SM, Barton PI (2013) A reliable simulator for dynamic flux balance analysis. Biotechnology and Bioengineering 110(3):792-802

[8] Harwood SM, Höffner K, Barton PI (2016) Efficient solution of ordinary differential equations with a parametric lexicographic linear program embedded. Numerische Mathematik 133(4):623-653

[9] Gomez JA, Höffner K, Barton PI (2014) DFBAlab: A fast and reliable MATLAB code for Dynamic Flux Balance Analysis. BMC Bioinformatic 15(409)

[10] Fondi M, Liò P (2015) Genome-scale metabolic network reconstruction. In: Mengoni A, Galardini M, Fondi M (eds) *B*acterial Pangenomics: Methods and Protocols, Springer, New York

[11] Hucka M, Finney A, Sauro H et al (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19(4):524-531

[12] IBM ILOG CPLEX (2009) V12. 1: User's Manual for CPLEX. International Business Machines Corporation 46(53): 157

[13] Gurobi Optimization (2012) Gurobi Optimizer Reference Manual. http://www.gurobi.com/documentation/6.5/refman.pdf. Accessed 8 Nov 2016

[14] Becker SA, Feist AM, Mo ML, Hannum G, Palsson BØ, Herrgard MJ (2007) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. Nature Protocols 2:727-738