Relaxed Flux Balance Analysis

Author: Ronan Fleming, Systems Biochemistry Group, University of Luxembourg.

Reviewer:

Introduction

We consider a biochemical network of m molecular species and n biochemical reactions. The biochemical network is mathematically represented by a stoichiometric matrix $S \in \mathcal{Z}^{m\times n}$. In standard notation, flux balance analysis (FBA) is the linear optimisation problem

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

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. Every FBA solution must satisfy the constraints, independent of any objective chosen to optimise over the set of constraints.

It may occur that the constraints on the FBA problem are not all simultaneously feasible, i.e., the system of inequalities is infeasible. This situation might be caused by an incorrectly specified reaction bound or the absence of a reaction from the stoichiometric matrix, such that a nonzero $b \notin \mathcal{R}(S)$. To resolve the infeasiblility, we consider a cardinality optimisation problem that seeks to minimise the number of bounds to relax, the number of fixed outputs to relax, the number of fixed inputs to relax, or a combination of all three, in order to render the problem feasible. The cardinality optimisation problem, termed *relaxed flux balance analysis,* is

$$\begin{array}{ll} \min\limits_{v,r,p,q} & \lambda\|r\|_0 + \gamma\|p\|_0 + \gamma\|q\|_0 \\ \text{s.t.} & Sv + r = b \\ & l - p \leq v \leq u + q \\ & p, q, r \geq 0 \end{array}$$

where $p, q \in \mathcal{R}^{n}$ denote the relaxations of the lower and upper bounds on reaction rates of the reaction rates vector v, and where $r \in \mathcal{R}^{m}$ denotes a relaxation of the mass balance constraint. Non-negative scalar parameters λ and γ can be used to trade off between relaxation of mass balance or bound constraints. A non-negative vector parameter λ can be used to prioritise relaxation of one mass balance constraint over another, e.g, to avoid relaxation of a mass balance constraint on a metabolite that is not desired to be exchanged across the boundary of the system. A non-negative vector parameter γ may be used to prioritise relaxation of bounds on some reactions rather than others, e.g., relaxation of bounds on exchange reactions rather than internal reactions. The optimal choice of parameters depends heavily on the biochemical context. A relaxation of the minimum number of constraints is desirable because ideally one should be able to justify the choice of bounds or choice of metabolites to be exchanged across the boundary of the system by recourse to experimental literature. This task is magnified by the number of constraints proposed to be relaxed.

PROCEDURE: RelaxFBA applied to Recon 3.0

TIMING: 20 seconds (computation), minutes - days (interpretation)

Recon 3D [brunk_recon_nodate] is the latest, most comprehensive, manually curated, genome-scale reconstruction of human metabolism. Recon3D is a reconstruction which currently encompasses ~3300 open reading frames, ~8000 unique metabolites, as well as ~12000 biochemical and transport reactions distributed over nine cellular compartments: cytoplasm [c], lysosome [l], nucleus [n], mitochondrion [m], mitochondrial intermembrane space [i], peroxisome [x], extracellular space [e], Golgi apparatus [g], and endoplasmic reticulum [r] [thiele_protocol_2010, brunk_recon_nodate]. Recon3.0model is a flux balance analysis model and the largest stoichiometrically and flux consistent subset of Recon3D. That is, no internal reaction in Recon3.0model is mass imbalanced and furthermore, every internal and every external reaction is admits a non-zero steady state flux. In this example, we take Recon3.0model, set the lower bound on the biomass reaction to require the synthesis of biomass yet close all of the external reactions in the model. The resulting model is therefore infeasible, that is, no steady state flux vector satisfies the steady state constraints and the bound constraints for the resulting flux balance analysis problem, irrespective of the objective coefficients, so we use relaxed flux balance analysis to identify the minimial set of external reaction bounds that are required to be relaxed in order to make biomass synthesis feasible.

If necessary, initialise the cobra toolbox

```
global TUTORIAL_INIT_CB;
if ~isempty(TUTORIAL_INIT_CB) && TUTORIAL_INIT_CB==1
    initCobraToolbox
    changeCobraSolver('gurobi','all');
end
```

Load Recon3.0model, unless it is allready loaded into the workspace.

```
clear model relaxOption
if ~exist('modelOrig','var')
    filename='Recon3.0model';
    directory='~/work/sbgCloud/programReconstruction/projects/recon2models/data/reconXComparis
    model = loadIdentifiedModel(filename,directory);
    model.csense(1:size(model.S,1),1)='E';
    modelOrig = model;
else
    model=modelOrig;
end
```

Identify the exchange reactions and biomass reaction(s) heuristically and close (a subset) of them

```
model = findSExRxnInd(model,size(model.S,1),1);
```

```
Found biomass reaction: biomass_reaction
Found biomass reaction: biomass_maintenance
Found biomass reaction: biomass_maintenance_noTrTr
ATP demand reaction is not considered an exchange reaction by default. It should be mass balanced:
DM_atp_c_ h2o[c] + atp[c]  -> h[c] + adp[c] + pi[c]
```

```
if ~any(model.biomassBool)
    error('Could not heuristically identify a biomass reaction')
end
```

Add a linear objective coefficient corresponding to the biomass reaction

```
model.biomassBool=strcmp(model.rxns,'biomass_reaction');
model.c(model.biomassBool)=1;
```

Check that biomass production is feasible

```
FBAsolution = optimizeCbModel(model,'max');
if FBAsolution.stat == 1
    disp('Relaxed model is feasible');
    bioMassProductionRate=FBAsolution.x(model.biomassBool);
    fprintf('%g%s\n', bioMassProductionRate, ' is the biomass production rate')
else
    disp('Relaxed model is infeasible');
end
```

```
Relaxed model is feasible
753.336 is the biomass production rate
```

Remove superflous biomass reactions and display the size of the reduced model

```
model = removeRxns(model,{'biomass_maintenance','biomass_maintenance_noTrTr'});
[m,n] = size(model.S);
fprintf('%6s\t%6s\n','#mets','#rxns'); fprintf('%6u\t%6u\t%s\n',m,n,' totals.')
```

```
 #mets	 #rxns
  5835	 10598	 totals.
```

```

```

First close all exchange reactions, except the biomass reaction

```
model.SIntRxnBool(strcmp(model.rxns,'biomass_reaction'))=0;
model.lb(~model.SIntRxnBool)=0;
model.ub(~model.SIntRxnBool)=0;
```

Now force the biomass reaction to be active

```
model.lb(model.biomassBool) = 1;
model.ub(model.biomassBool) = 10;
```

Check if the model is feasible

```
FBAsolution = optimizeCbModel(model,'max', 0, true);
if FBAsolution.stat == 1
    disp('Model is feasible. Nothing to do.');
    return
else
    disp('Model is infeasible');
end
```

```
Model is infeasible
```

Relaxed flux balance analysis is implemented with the function relaxFBA

```
%    [solution] = relaxFBA(model, relaxOption)
```

The inputs are a COBRA model and an optional parameter vector

```
% INPUTS:
%    model:           COBRA model structure
%    relaxOption:     Structure containing the relaxation options:
%
%                       * internalRelax:
%                         * 0 = do not allow to relax bounds on internal reactions
%                         * 1 = do not allow to relax bounds on internal reactions with finite
%                         * 2 = allow to relax bounds on all internal reactions
%
%                       * exchangeRelax:
%                         * 0 = do not allow to relax bounds on exchange reactions
%                         * 1 = do not allow to relax bounds on exchange reactions of the type
%                         * 2 = allow to relax bounds on all exchange reactions
%
%                       * steadyStateRelax:
%                         * 0 = do not allow to relax the steady state constraint S*v = b
%                         * 1 = allow to relax the steady state constraint S*v = b
%
%                       * toBeUnblockedReactions - n x 1 vector indicating the reactions to be
%                         * toBeUnblockedReactions(i) = 1 : impose v(i) to be positive
%                         * toBeUnblockedReactions(i) = -1 : impose v(i) to be negative
%                         * toBeUnblockedReactions(i) = 0 : do not add any constraint
%
%                       * excludedReactions - n x 1 bool vector indicating the reactions to be
%                         * excludedReactions(i) = false : allow to relax bounds on reaction i
%                         * excludedReactions(i) = true : do not allow to relax bounds on react
%
%                       * excludedMetabolites - m x 1 bool vector indicating the metabolites to
%                         * excludedMetabolites(i) = false : allow to relax steady state constr
%                         * excludedMetabolites(i) = true : do not allow to relax steady state
%
%                       * lamda - trade-off parameter of relaxation on steady state constraint
%                       * gamma - trade-off parameter of relaxation on bounds
%
% Note, excludedReactions and excludedMetabolites override all other relaxation options.
```

Do not allow to relax bounds on any internal reaction

```
relaxOption.internalRelax = 0;
```

Allow to relax bounds on all exchange reactions

```
relaxOption.exchangeRelax = 2;
```

Do not allow to relax the steady state constraint S*v = b

```
relaxOption.steadyStateRelax = 0;
```

Set the tolerance to distinguish between zero and non-zero flux

```
feasTol = getCobraSolverParams('LP', 'feasTol');
relaxOption.epsilon = feasTol/100;%*100;
```

Set the trade-off parameter for relaxation of bounds (advanced user). A larger value of gamma will

```
relaxOption.gamma  = 10;
```

Set the trade-off parameter for relaxation on steady state constraint (advanced user)

```
relaxOption.lambda = 10;
```

Call the relaxFBA function, deal the solution, and set small values to zero

```
tic;
solution = relaxFBA(model,relaxOption);
timeTaken=toc;
[v,r,p,q] = deal(solution.v,solution.r,solution.p,solution.q);
if 0
    p(p<relaxOption.epsilon) = 0;%lower bound relaxation
    q(q<relaxOption.epsilon) = 0;%upper bound relaxation
    r(r<relaxOption.epsilon) = 0;%steady state constraint relaxation
end
```

The output is a solution structure with a 'stat' field reporting the solver status and a set of fields matching the relaxation of constraints given in the mathematical formulation of the relaxed flux balance problem above.

```
% OUTPUT:
%    solution:       Structure containing the following fields:
%                      * stat - status
%                        * 1  = Solution found
%                        * 0  = Infeasible
%                        * -1 = Invalid input
%                      * r - relaxation on steady state constraints S*v = b
%                      * p - relaxation on lower bound of reactions
%                      * q - relaxation on upper bound of reactions
%                      * v - reaction rate
```

Summarise the proposed relaxation solution

```
if solution.stat == 1

   dispCutoff=relaxOption.epsilon;

    fprintf('%s\n',['Relaxed flux balance analysis problem solved in ' num2str(timeTaken) ' se

    fprintf('%u%s\n',nnz(r),' steady state constraints relaxed');

    fprintf('%u%s\n',nnz(abs(p)>dispCutoff & model.SIntRxnBool),' internal lower bounds relaxe
    fprintf('%u%s\n',nnz(abs(q)>dispCutoff & model.SIntRxnBool),' internal upper bounds relaxe
    fprintf('%u%s\n',nnz(abs(p)>dispCutoff & ~model.SIntRxnBool),' external lower bounds relax
    fprintf('%u%s\n',nnz(abs(q)>dispCutoff & ~model.SIntRxnBool),' external upper bounds relax

    maxUB = max(max(model.ub),-min(model.lb));
    minLB = min(-max(model.ub),min(model.lb));
    intRxnFiniteBound = ((model.ub < maxUB) & (model.lb > minLB));
    fprintf('%u%s\n',nnz(abs(p)>dispCutoff & intRxnFiniteBound),' finite lower bounds relaxed'
    fprintf('%u%s\n',nnz(abs(q)>dispCutoff & intRxnFiniteBound),' finite upper bounds relaxed'

    exRxn00 = ((model.ub == 0) & (model.lb == 0));
    fprintf('%u%s\n',nnz(abs(p)>dispCutoff & exRxn00),' lower bounds relaxed on fixed reactior
    fprintf('%u%s\n',nnz(abs(q)>dispCutoff & exRxn00),' upper bounds relaxed on fixed reactior

else
    disp('relaxFBA problem infeasible, check relaxOption fields');
end
```

```
Relaxed flux balance analysis problem solved in 48.7516 seconds.
0 steady state constraints relaxed
0 internal lower bounds relaxed
0 internal upper bounds relaxed
604 external lower bounds relaxed
605 external upper bounds relaxed
604 finite lower bounds relaxed
605 finite upper bounds relaxed
603 lower bounds relaxed on fixed reactions (lb=ub=0)
605 upper bounds relaxed on fixed reactions (lb=ub=0)
```

TROUBLESHOOTING

Given an infeasible problem,

$$\begin{aligned} Sv &= b, \\ l \leq v &\leq u, \end{aligned}$$

the *relaxed flux balance analysis* problem

$$\begin{array}{ll} \min\limits_{v,r,p,q} & \lambda\|r\|_0 + \gamma\|p\|_0 + \gamma\|q\|_0 \\ \text{s.t.} & Sv + r = b \\ & l - p \leq v \leq u + q \\ & p, q, r \geq 0 \end{array}$$

will always find a solution. However, relaxFBA offers the user the option to disallow relaxation of some of the constraints. If too many constraints are not allowed to be relaxed, then relaxFBA will report an infeasible problem. The fields of relaxOption should be reviewed. For example, if relaxation of steady state constraints is not alllowed, yet b is nonzero and not in the range of the stoichiometric matrix, then the relaxFBA problem will be infeasible. To allow the relaxation of the steady state constraint, S*v = b, then use

```
%relaxOption.steadyStateRelax = 1;
```

If relaxFBA does return a solution, but it is not biochemcially realistic, then again review the fields of relaxOption, to allow or disallow relaxation of certain constraints. For example, to specifically disallow relaxation of the bounds on reaction with model.rxns abbreviation 'myReaction', use

```
%relaxOption.excludedReactions=false(n,1);
%relaxOption.excludedReactions(strcmp(model.rxns,'myReaction'))=1;
```

To specifically disallow relaxation of the steady state constraint on a molecualr species with model.mets abbreviation 'myMetabolite', then use:

```
%relaxOption.excludedMetabolite=false(m,1);
%relaxOption.excludedMetabolite(strcmp(model.mets,'myMetabolite'))=1;
```

Even if the set of relaxations are properly set, in a boolean sense, tweaking of the DCA card trade off parameters can help narrow down to a biochemically realistic solution, by iterating between the biochemical literature and the numerical results from relaxFBA after tweaking the parameters. This flexibility is provided for the expert user. See relaxFBA_cappedL1.m. A standard set of advanced parameters are:

```
%relaxOption.nbMaxIteration = 1000; %max number of iterations of the cappedL1 problem
%relaxOption.gamma0  = 0;   %trade-off parameter of l0 part of v
%relaxOption.gamma1  = 0;    %trade-off parameter of l1 part of v
%relaxOption.lambda0 = 10;   %trade-off parameter of l0 part of r
%relaxOption.lambda1 = 0;    %trade-off parameter of l1 part of r
%relaxOption.alpha0  = 10;    %trade-off parameter of l0 part of p and q
%relaxOption.alpha1  = 0;     %trade-off parameter of l1 part of p and q
%relaxOption.theta   = 2;    %parameter of capped l1 approximation
```

ANTICIPATED RESULTS

relaxFBA will return a set of steady state constraints, lower bounds, and upper bounds, that are required to be relaxed to ensure that the FBA problem is feasible. It is necessary to analyse the solution biochemically, to see if it makes sense to relax the suggested constraints. The following code will report a summary of the results.

```
if solution.stat == 1
    printFlag=0;
    lineChangeFlag=0;
    if 0
        dispCutoffLower=relaxOption.epsilon;
        dispCutoffUpper=inf;
    else
        dispCutoffLower=-10;
        dispCutoffUpper=10;
    end
   if any(r)
        fprintf('\n%s\n','Steady state constraints relaxed');
        for i=1:m
            if abs(r(i))>dispCutoffLower && abs(r(i))<dispCutoffUpper
                fprintf('%s\n',model.mets{i});
            end
        end
   else
       fprintf('\n%s\n','No steady state constraints relaxed');
   end
    if any(p)
        fprintf('%s\n','Lower bounds relaxed');
        for j=1:n
            if abs(p(j))>dispCutoffLower && abs(p(j))<dispCutoffUpper && p(j)~=0
                rxnAbbrList=model.rxns(j);
                formulas = printRxnFormula(model, rxnAbbrList, printFlag, lineChangeFlag);
                fprintf('%6g\t%s',p(j),formulas{1});
            end
        end
    else
        fprintf('\n%s\n','No lower bounds relaxed');
    end
    if any(q)
```

```
        fprintf('\n%s\n','Upper bounds relaxed');
        for j=1:n
            if abs(q(j))>dispCutoffLower && abs(q(j))<dispCutoffUpper && q(j)~=0
                rxnAbbrList=model.rxns(j);
                formulas = printRxnFormula(model, rxnAbbrList, printFlag, lineChangeFlag);
                fprintf('%6g\t%s',q(j),formulas{1});
            end
        end
    else
        fprintf('\n%s\n','No upper bounds relaxed');
    end
end
```

```
No steady state constraints relaxed
Lower bounds relaxed
  0.15 eicostet[e]  ->
  0.05 M00017[e]  ->
  0.15 M00117[e]  ->
  0.05 M01207[e]  ->
Upper bounds relaxed
  0.45 adrn[e]  ->
   0.1 arach[e]  ->
   0.5 clpnd[e]  ->
   0.2 dlnlcg[e]  ->
   0.2 hexc[e]  ->
   0.1 nrvnc[e]  ->
  0.25 docosac[e]  ->
   0.2 M02457[e]  ->
  0.05 M03045[e]  ->
```

Generate a relaxed model and test if it is feasible.

```
if solution.stat == 1
    modelRelaxed=model;
    delta=0;%can be used for debugging, in case more relaxation is necessary
    modelRelaxed.lb = model.lb - p - delta;
    modelRelaxed.ub = model.ub + q + delta;
    modelRelaxed.b  = model.b  - r;

    FBAsolution = optimizeCbModel(modelRelaxed,'max', 0, true);
    if FBAsolution.stat == 1
        disp('Relaxed model is feasible');
    else
        disp('Relaxed model is infeasible');
        solutionRelaxed = relaxFBA(modelRelaxed,relaxOption);
    end
end
```

```
Relaxed model is feasible
```

EXPECTED RESULTS

The relaxed model should be feasible. Indicated by 'Relaxed model is feasible'

TROUBLESHOOTING

If the relaxed model is not feasible. If not, there could be a numerical issue due to the numerical tolerance of the linear optimisation solutions or due to the numerical tolerance on the relaxFBA algorithm, both of which are by default set to the feasibility tolerance for the currently installed solver (typically 1e-6 for a double precision solver like Gurobi). If problems persist, examine the numerical properties of the constraints, esp wrt scaling, or try the dqqMinos solver.

```
%changeCobraSolver('dqqMinos','LP')
```