Relaxed Flux Balance Analysis: Toy model

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

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

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

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

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 α 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 α 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: RelaxedFBA applied to a toy model

```
clear;
rxnForms = {' -> A','A -> B','B -> C', 'B -> D','D -> C','C ->'};
rxnNames = {'R1','R2','R3','R4','R5', 'R6'};
model = createModel(rxnNames, rxnNames,rxnForms);
```

```
Metabolite A[c] not in model - added to the model
R1  <=> A[c]
Metabolite B[c] not in model - added to the model
R2 A[c]  <=> B[c]
Metabolite C[c] not in model - added to the model
R3 B[c]  <=> C[c]
Metabolite D[c] not in model - added to the model
R4 B[c]  <=> D[c]
R5 D[c]  <=> C[c]
R6 C[c]  <=>
```

```
model.lb(3) = 2;
model.lb(4) = 2;
model.ub(6) = 3;
```

Print the constraints

```
printConstraints(model, -1001, 1001)
```

```
MinConstraints:
R1 -1000
R2 -1000
R3 2
R4 2
R5 -1000
R6 -1000
maxConstraints:
R1 1000
R2 1000
R3 1000
R4 1000
R5 1000
R6 3
```

Identify the exchange reactions and biomass reaction(s) heuristically

```
model = findSExRxnInd(model,size(model.S,1),0);
%    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 bounds
%                         * 2 = allow to relax bounds on all internal reactions
relaxOption.internalRelax = 2;
%
%                       * exchangeRelax:
%
%                         * 0 = do not allow to relax bounds on exchange reactions
%                         * 1 = do not allow to relax bounds on exchange reactions of the type [0,0]
%                         * 2 = allow to relax bounds on all exchange reactions
relaxOption.exchangeRelax = 0;
%                       * 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
relaxOption.steadyStateRelax = 0;
%                       * toBeUnblockedReactions - n x 1 vector indicating the reactions to be unblocked (optional)
%
%                         * 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 excluded from relaxation (optional)
%
%                         * excludedReactions(i) = false : allow to relax bounds on reaction i
%                         * excludedReactions(i) = true : do not allow to relax bounds on reaction i
%
%                       * excludedMetabolites - m x 1 bool vector indicating the metabolites to be excluded from relaxation (optional)
%
%                         * excludedMetabolites(i) = false : allow to relax steady state constraint on metabolite i
%                         * excludedMetabolites(i) = true : do not allow to relax steady state constraint on metabolite i
%
```

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

```
if 1
    %feasTol = getCobraSolverParams('LP', 'feasTol');
    %relaxOption.epsilon = feasTol/100;%*100;
    relaxOption.epsilon = 10e-6;
else
    relaxOption.nbMaxIteration = 1000;
    relaxOption.epsilon = 10e-6;
    relaxOption.gamma0  = 10;   %trade-off parameter of l0 part of v
    relaxOption.gamma1  = 1;    %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  = 0;    %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
end
```

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
```

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

```
relaxOption
```

```
relaxOption =
       internalRelax: 2
       exchangeRelax: 0
    steadyStateRelax: 0
             epsilon: 1.0000e-05
```

```
tic;
solution = relaxedFBA(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
```

Display the proposed relaxation solution

```
fprintf('%s\n','Relaxation of steady state constraints:')
```

```
Relaxation of steady state constraints:
```

```
disp(r)
```

```
     0
     0
     0
     0
```

```
fprintf('%s\n','Relaxation on lower bound of reactions:')
```

```
Relaxation on lower bound of reactions:
```

```
disp(p)
```

```
           0
           0
        1002
        1002
           0
           0
```

```
fprintf('%s\n','Relaxation on upper bound of reactions:')
```

```
Relaxation on upper bound of reactions:
```

```
disp(q)
```

```
     0
     0
     0
     0
     0
     0
```

Summarise the proposed relaxation solution

```
if solution.stat == 1

   dispCutoff=relaxOption.epsilon;

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

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

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

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

    fprintf('%u%s\n',nnz(abs(p)>dispCutoff | abs(q)>dispCutoff & ~model.SIntRxnBool),' external lower or upper bounds relaxed');

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

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

```
Relaxed flux balance analysis problem solved in 0.048242 seconds.
0 steady state constraints relaxed
2 internal lower bounds relaxed
0 internal upper bounds relaxed
0 internal lower and upper bounds relaxed
0 external lower bounds relaxed
0 external upper bounds relaxed
0 external lower and upper bounds relaxed
2 external lower or upper bounds relaxed
0 finite lower bounds relaxed
0 finite upper bounds relaxed
0 lower bounds relaxed on fixed reactions (lb=ub=0)
0 upper bounds relaxed on fixed reactions (lb=ub=0)
```

```
return
```

Another example

```
rxnForms = {' -> A',...
            'A -> B',...
            'A -> C',...
            'D -> B',...
            'E -> C',...
            'E -> D',...
            'E -> ',...
            'A -> F',...
            'G -> F',...
            'H -> G',...
            'E -> H'};
rxnNames = {'R1','R2','R3','R4','R5','R6','R7','R8','R9','R10','R11'};
model = createModel(rxnNames, rxnNames,rxnForms);
```

Assume all reactions are irreversible

```
model.lb(:) = 0;
model.ub(:) = 10;
```

Reaction R7 with bounds 1 <= v_7 <= 10

```
model.lb(7) = 1;
```

Print the constraints

```
printConstraints(model, -1001, 1001)
```

Identify the exchange reactions and biomass reaction(s) heuristically

```
model = findSExRxnInd(model,size(model.S,1),0);
%    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 bounds
%                       * 2 = allow to relax bounds on all internal reactions
relaxOption.internalRelax = 2;
%
%                     * exchangeRelax:
%
%                       * 0 = do not allow to relax bounds on exchange reactions
%                       * 1 = do not allow to relax bounds on exchange reactions of the type [0,0]
%                       * 2 = allow to relax bounds on all exchange reactions
relaxOption.exchangeRelax = 0;
%                     * 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
relaxOption.steadyStateRelax = 0;
%                     * toBeUnblockedReactions - n x 1 vector indicating the reactions to be unblocked (optional)
%
%                       * 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 excluded from relaxation (optional)
%
%                       * excludedReactions(i) = false : allow to relax bounds on reaction i
%                       * excludedReactions(i) = true : do not allow to relax bounds on reaction i
%
%                     * excludedMetabolites - m x 1 bool vector indicating the metabolites to be excluded from relaxation (optional)
%
%                       * excludedMetabolites(i) = false : allow to relax steady state constraint on metabolite i
%                       * excludedMetabolites(i) = true : do not allow to relax steady state constraint on metabolite i
```

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

```
feasTol = getCobraSolverParams('LP', 'feasTol');
```

```
relaxOption.epsilon = feasTol/100;%*100;
```

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

```
tic;
solution = relaxedFBA(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) ' seconds.'])

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

    fprintf('%u%s\n',nnz(abs(p)>dispCutoff & ~abs(q)>dispCutoff & model.SIntRxnBool),' internal only lower bounds relaxed');
    fprintf('%u%s\n',nnz(abs(q)>dispCutoff & ~abs(p)>dispCutoff & model.SIntRxnBool),' internal only upper bounds relaxed');
    fprintf('%u%s\n',nnz(abs(p)>dispCutoff & abs(q)>dispCutoff & model.SIntRxnBool),' internal lower and upper bounds relaxed');

    fprintf('%u%s\n',nnz(abs(p)>dispCutoff & ~abs(q)>dispCutoff & ~model.SIntRxnBool),' external only lower bounds relaxed');
    fprintf('%u%s\n',nnz(abs(q)>dispCutoff & ~abs(p)>dispCutoff & ~model.SIntRxnBool),' external only upper bounds relaxed');
    fprintf('%u%s\n',nnz(abs(p)>dispCutoff & abs(q)>dispCutoff & ~model.SIntRxnBool),' external lower and upper bounds relaxed');

    fprintf('%u%s\n',nnz(abs(p)>dispCutoff | abs(q)>dispCutoff & ~model.SIntRxnBool),' external lower or upper bounds relaxed');

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

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

REFERENCES

Fleming, R.M.T., et al., Cardinality optimisation in constraint-based modelling: Application to Recon 3D (submitted), 2017.

Brunk, E. et al. Recon 3D: A resource enabling a three-dimensional view of gene variation in human metabolism. (submitted) 2017.