Fit a COBRA model to an experimental flux vector

A given vector of experimentally measured fluxes v_{exp}, may not be consistent with the feasible set of steady state fluxes. Specifically, inconsistent with the set defined by the steady state constraint (Sv=0) as well as the lower and upper bounds on each reactionl, l ≤ v ≤ u. Should this occur, we fit the model to the experimental data, relaxing the constraints on the bounds, that admits a steady state flux v , using the following quadriatic optimisation problem

$$\begin{aligned} \min_{v,p,q} \quad & (v_{exp} - v)^T \text{diag}(w_{exp})(v_{exp} - v) + p^T \text{diag}(w_l) p + q^T \text{diag}(w_u) q \\ \text{s.t.} \quad & Sv = 0, \\ & l - p \leq v \leq u + q, \\ & 0 \leq p \\ & 0 \leq q \end{aligned}$$

where p ≥ 0 and q ≥ 0 are non-negative variables that permit relaxation of the lower and upper bound constraints, respectively. This formulation also allows for different weights to be input as parameters to penalise deviation from experimentally measured mean fluxes, with w_{exp} = weightExp, penalise relaxation of lower bounds, with w_l = weightLower, and penalise relaxation of upper bounds, with w_u = weightUpper. For example, if the experimentally measured flux is actually the mean flux, then one could set w_{exp} to be the inverse of the variance on the experimental flux measurements, thereby increasing the penalty on deviation from an experimentally measured mean flux where the variance is lower. Certain lower or upper bounds might not be realistic to be relaxed, e.g., an essential amino acid can always be taken up but never secreted, therefore the upper bound on the corresponding exchange reaction must be zero. The code below also works if there are additional coupling constraints, $\text{Cv} \leq d$, but these are not relaxed.

Set the solver

```
LPsolverOK = changeCobraSolver('gurobi','QP');
```

```
> changeCobraSolver: Gurobi interface added to MATLAB path.
> gurobi (version 811) is compatible and fully tested with MATLAB R2019a on your operating system.
```

```
%LPsolverOK = changeCobraSolver('mosek','QP');
```

Load an example model

```
filename='~/work/sbgCloud/data/models/unpublished/reconMitochondria/
modelRecon3MitoOpen.mat';
%model.c(strcmp(model.rxns,'EX_o2[c]'))=1;
%model.c(strcmp(model.rxns,'EX_h2o2[c]'))=1;
%filename='~/work/sbgCloud/programReconstruction/projects/exoMetDN/results/
iPSCderivedModels/2018models/model_iHNESC2DN_unfeasible.mat';
try
    model=readCbModel(filename);
catch
    load(filename)
    modelRecon3Mito = model
```

```
end
```

```
Each model.subSystems{x} is a cell array, allowing more than one subSystem per reaction.
modelRecon3Mito = struct with fields:
                            rxns: {1625×1 cell}
                            mets: {1042×1 cell}
                        metNames: {8424×1 cell}
                     metFormulas: {1042×1 cell}
                              lb: [1625×1 double]
                              ub: [1625×1 double]
                      subSystems: {1625×1 cell}
                        rxnNames: {1625×1 cell}
                               S: [1042×1625 double]
                               b: [1042×1 double]
                               c: [1625×1 double]
                             rev: [1625×1 double]
                           genes: {3695×1 cell}
                         grRules: {1625×1 cell}
                       metCharge: [1042×1 double]
                      rxnGeneMat: [1625×3695 double]
                           rules: {1625×1 cell}
                 biomassRxnAbbr: 'biomass'
                    biomassBool: [1625×1 logical]
                    ExchRxnBool: [1625×1 logical]
                      DMRxnBool: [1625×1 logical]
                    SinkRxnBool: [1625×1 logical]
                 SOnlyExMetBool: [1042×1 logical]
                SOnlyIntMetBool: [1042×1 logical]
                     SExMetBool: [1042×1 logical]
                balancedRxnBool: [1625×1 logical]
                balancedMetBool: [1042×1 logical]
                       Elements: {'H'  'C'  'O'  'P'  'S'  'N'  'Mg'  'X'  'Fe'  'Zn'  'Co'  'R'  'Ca
           missingFormulaeBool: [1042×1 logical]
            SConsistentMetBool: [1042×1 logical]
            SConsistentRxnBool: [1625×1 logical]
      unknownSConsistencyMetBool: [1042×1 logical]
      unknownSConsistencyRxnBool: [1625×1 logical]
rxnUnknownInconsistentRemoveBool: [1625×1 logical]
metUnknownInconsistentRemoveBool: [1042×1 logical]
                    SIntRxnBool: [1625×1 logical]
                    SIntMetBool: [1042×1 logical]
          SInConsistentMetBool: [1042×1 logical]
          SInConsistentRxnBool: [1625×1 logical]
```

```
[nMet,nRxn]=size(model.S);
```

Identify the internal and external reactions

```
if ~isfield(model,'SIntRxnBool')  || ~isfield(model,'SIntMetBool')
    %finds the reactions in the model which export/import from the model
    %boundary i.e. mass unbalanced reactions
    %e.g. Exchange reactions
    %     Demand reactions
    %     Sink reactions
    model = findSExRxnInd(model,[],-1);
end
% model.SIntRxnBool         Boolean of reactions heuristically though to be
mass balanced.
% model.SIntMetBool         Boolean of metabolites heuristically though to
be involved in mass balanced reactions.
```

```
nIntRxn=nnz(model.SIntRxnBool);
```

Generate a random experimental flux for the external reactions in a model. NaN means no experimental information is available for a reaction rate. Note that some of the experimental fluxes must be outside the bounds on the model, otherwise no relaxation of the bounds will be necessary.

```
vExp=(rand(nRxn,1)-1)*1000*4;
vExp(model.SIntRxnBool,1)=NaN;

% vExp(model.SIntRxnBool,1)=0;
```

Set the weight on the Euclidean norm of the relaxation to the lower bounds on the predicted steady state flux vector. Only for allow relaxation for external reactions. The weight penalty on relaxation of the lower bound should be greater than that on the experimental flux, and upper bound, if the model lower bound is considered more reliable than the experimental data and the upper bound.

```
weightLower=ones(nRxn,1);
weightLower(model.SIntRxnBool)=inf;
```

Set the weight on the Euclidean norm of the relaxation to the upper bounds on the predicted steady state flux vector. Only for allow relaxation for external reactions. The weight penalty on relaxation of the upper bound should be greater than that on the experimental flux, and lower bound, if the model upper bound is considered more reliable than the experimental data and the lower bound.

```
weightUpper=ones(nRxn,1);
weightUpper(model.SIntRxnBool)=inf;
```

Set the weight weighton the Euclidean distance of the predicted steady state flux from the experimental steady state flux. Uniform at first. The weight penalty on relaxation from the experimental flux vector should be greater than that on the lower and upper bounds, if the experimental data is considered more reliable than the lower and upper bounds of the model. Weights are ignored on the reactions without experimental data, i.e. vExp(n)==NaN.

```
weightExp =ones(nRxn,1);
% weightExp(model.SIntRxnBool) =0;
```

Select the amount of information to be returned by the solver

```
printLevel=0;
%printLevel=1;
```

Compute the steady state flux vector that minimises the weighted Euclidean norm between experimental and predicted steady state flux vector, plus the weighted Euclidean norm relaxation of the model lower bounds, plus the weighted Euclidean norm relaxation of the model upper bounds. Also save the relaxed model that admits such a steady state flux.

```
%   changeCobraSolver('gurobi','all');
%   changeCobraSolver('ibm_cplex','all');
%    changeCobraSolver('glpk','all');
%    changeCobraSolver('pdco','all');
```

```
%    changeCobraSolver('qpng','all')
%
[modelRelaxed,v,p,q,dv] = fitExperimentalFlux(model, vExp, weightLower,
weightUpper, weightExp, printLevel);
```

Estimate the smallest nonzero significant flux

```
feasTol = getCobraSolverParams('LP', 'feasTol');
tol=feasTol*1e5;
disp(tol)
```

```
    0.1000
```

Identify the reactions where the bounds are relaxed

```
boolRelaxedLowerBound=p>tol;
boolRelaxedUpperBound=q>tol;
```

Display the predicted fluxes in relation to the original model bounds and the relaxation of the lower model bounds

```
k=1;
for n=1:nRxn
    if k==100 %only show 100 reactions maximum
        break;
    end
    if n==1

fprintf('%-20s%12s%12s%12s%12s%12s%12s%12s%12s%12s%12s\n','rxns{n}','lb_n
ew','wl','-p','lb','v','vexp','wexp','ub','q','wu','ub_new');
    end
    if boolRelaxedLowerBound(n)
        k=k+1;

fprintf('%-20s%12.4g%12.4g%12.4g%12.4g%12.4g%12.4g%12.4g%12.4g%12.4g%12.4g%12
.4g\n',...
            model.rxns{n},modelRelaxed.lb(n),weightLower(n),-
p(n),model.lb(n),v(n),vExp(n),weightExp(n),model.ub(n),q(n),weightUpper(n),mo
delRelaxed.ub(n));
    end
end
```

```
rxns{n}                 lb_new          wl          -p          lb           v        vexp        wexp
EX_h2o[c]                -1104           1      -103.7       -1000       -1104       -2999           1
EX_akg[c]                -1869           1      -868.9       -1000       -1869       -3348           1
EX_ppcoa[c]              -1566           1        -566       -1000       -1566       -3801           1
```

Display the predicted fluxes in relation to the original model bounds and the relaxation of the upper model bounds

```
k=1;
for n=1:nRxn
```

```
    if k==100 %only show 100 reactions maximum
        break;
    end
    if n==1

fprintf('%-20s%12s%12s%12s%12s%12s%12s%12s%12s%12s%12s\n','rxns{n}','lb_n
ew','wl','-p','lb','v','vexp','wexp','ub','q','wu','ub_new');
    end
    if boolRelaxedUpperBound(n)
        k=k+1;

fprintf('%-20s%12.4g%12.4g%12.4g%12.4g%12.4g%12.4g%12.4g%12.4g%12.4g%12
.4g\n',...
            model.rxns{n},modelRelaxed.lb(n),weightLower(n),-
p(n),model.lb(n),v(n),vExp(n),weightExp(n),model.ub(n),q(n),weightUpper(n),mo
delRelaxed.ub(n));
    end
end
```

rxns{n}	lb_new	wl	-p	lb	v	vexp	wexp
EX_crn[c]	-1000	1	-0	-1000	1407	-490.1	1
EX_dtdp[c]	-1000	1	-0	-1000	1098	-964.1	1