Relaxed Flux Balance Analysis

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

Reviewer: Marouen Ben Guebila.

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

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

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

$$\begin{aligned} \min_{v,r,p,q} \quad & c^T v + \lambda \|r\|_0 + \alpha \|p\|_0 + \alpha \|q\|_0 \\ \text{s.t.} \quad & Sv + r = b \\ & l - p \leq v \leq u + q \\ & p, q, r \geq 0 \end{aligned}$$

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 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 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.

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

```
clear model relaxOption
if ~exist('modelOrig','var')
    %select your own model, or use Recon2.0model instead
    if 0
        filename='Recon3.0model';
        directory='~/work/sbgCloud/programReconstruction/projects/recon2models/data/reconXComp
        model = loadIdentifiedModel(filename,directory);
    else
        filename2='Recon20model';
        if exist('Recon20model.mat','file')==2
            model = readCbModel(filename2);
        end
    end
    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 relaxedFBA

```
%    [solution] = relaxedFBA(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 bounds
%  * 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 - weighting on relaxation of relaxation on steady state constra
%                     * alpha - weighting on relaxation of reaction bounds
%                     * gamma - weighting on zero norm of fluxes
%
% 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 zero norm of flux (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 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) ' se

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

    fprintf('%u%s\n',nnz(abs(p)>dispCutoff & ~abs(q)>dispCutoff & model.SIntRxnBool),' interna
    fprintf('%u%s\n',nnz(abs(q)>dispCutoff & ~abs(p)>dispCutoff & model.SIntRxnBool),' interna
    fprintf('%u%s\n',nnz(abs(p)>dispCutoff & abs(q)>dispCutoff & model.SIntRxnBool),' internal

    fprintf('%u%s\n',nnz(abs(p)>dispCutoff & ~abs(q)>dispCutoff & ~model.SIntRxnBool),' extern
    fprintf('%u%s\n',nnz(abs(q)>dispCutoff & ~abs(p)>dispCutoff & ~model.SIntRxnBool),' extern
    fprintf('%u%s\n',nnz(abs(p)>dispCutoff & abs(q)>dispCutoff & ~model.SIntRxnBool),' externa

    fprintf('%u%s\n',nnz(abs(p)>dispCutoff | abs(q)>dispCutoff & ~model.SIntRxnBool),' externa
```

```
    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 reaction
    fprintf('%u%s\n',nnz(abs(q)>dispCutoff & exRxn00),' upper bounds relaxed on fixed reaction

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

```
Relaxed flux balance analysis problem solved in 47.6492 seconds.
0 steady state constraints relaxed
0 internal only lower bounds relaxed
0 internal only upper bounds relaxed
0 internal lower and upper bounds relaxed
497 external only lower bounds relaxed
498 external only upper bounds relaxed
107 external lower and upper bounds relaxed
1102 external lower or 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,

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

the *relaxed flux balance analysis* problem

$$\begin{aligned} \min_{v,r,p,q} \quad & \lambda \|r\|_0 + \gamma \|p\|_0 + \gamma \|q\|_0 \\ \text{s.t.} \quad & Sv + r = b \\ & l - p \le v \le u + q \\ & p, q, r \ge 0 \end{aligned}$$

will always find a solution. However, relaxedFBA offers the user the option to disallow relaxation of some of the constraints. If too many constraints are not allowed to be relaxed, then relaxedFBA 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 relaxedFBA problem will be infeasible. To allow the relaxation of the steady state constraint, S*v = b, then use

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

If relaxedFBA 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 molecular 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 relaxedFBA 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

relaxedFBA 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 1
        dispCutoffLower=relaxOption.epsilon;
        dispCutoffUpper=inf;
    else
        %useful for numerical debugging
        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
  1000 datp[m]  ->
  1000 datp[n]  ->
  1000 dctp[n]  ->
  1000 dgtp[n]  ->
  1000 dttp[n]  ->
  1000 ethamp[r]  ->
  1000 gpi_sig[r]  ->
  1000 mem2emgacpail_prot_hs[r]  ->
  1000 Ser_Gly_Ala_X_Gly[l]  ->
  1000 10fthf7glu[e]  ->
  1000 4nph[e]  ->
  1000 5adtststerone[e]  ->
  1000 7dhf[e]  ->
  1000 7thf[e]  ->
  1000 adp[e]  ->
  1000 adprbp[e]  ->
  1000 adrnl[e]  ->
  1000 ala_D[e]  ->
  1000 aqcobal[e]  ->
  1000 arachd[e]  ->
  1000 ascb_L[e]  ->
  1000 atp[e]  ->
  1000 bilglcur[e]  ->
  1000 biocyt[e]  ->
  1000 cholate[e]  ->
  1000 chsterol[e]  ->
  1000 chtn[e]  ->
  1000 cmp[e]  ->
```

```
1000 crmp_hs[e]  ->
1000 crn[e]  ->
1000 crtsl[e]  ->
1000 cspg_c[e]  ->
1000 dag_hs[e]  ->
1000 dheas[e]  ->
1000 dlnlcg[e]  ->
0.15 eicostet[e]  ->
1000 estrones[e]  ->
1000 gbside_hs[e]  ->
1000 gchola[e]  ->
1000 gluala[e]  ->
1000 glygn2[e]  ->
1000 glygn4[e]  ->
1000 gthrd[e]  ->
1000 gtp[e]  ->
1000 h2o2[e]  ->
1000 ha[e]  ->
1000 hdcea[e]  ->
1000 i[e]  ->
1000 idp[e]  ->
1000 imp[e]  ->
1000 ksi[e]  ->
1000 Lcystin[e]  ->
1000 leuktrA4[e]  ->
1000 leuktrF4[e]  ->
1000 lnlc[e]  ->
1000 lnlnca[e]  ->
1000 nadp[e]  ->
1000 ncam[e]  ->
1000 o2s[e]  ->
1000 ocdca[e]  ->
1000 ocdcea[e]  ->
1000 octa[e]  ->
1000 pe_hs[e]  ->
1000 peplys[e]  ->
1000 prostgd2[e]  ->
1000 prostge1[e]  ->
1000 prostgf2[e]  ->
1000 ps_hs[e]  ->
1000 ptdca[e]  ->
1000 retinol[e]  ->
1000 s2l2fn2m2masn[e]  ->
1000 spc_hs[e]  ->
1000 sph1p[e]  ->
1000 sphs1p[e]  ->
1000 strch1[e]  ->
1000 strch2[e]  ->
1000 strdnc[e]  ->
1000 tag_hs[e]  ->
1000 tchola[e]  ->
1000 thf[e]  ->
1000 thym[e]  ->
```

```
1000 triodthy[e]  ->
1000 ttdca[e]  ->
1000 utp[e]  ->
1000 vacc[e]  ->
1000 whhdca[e]  ->
1000 xoltri27[e]  ->
1000 xylt[e]  ->
1000 pre_prot[r]  ->
1000 4abutn[e]  ->
1000 ctp[e]  ->
1000 dgmp[e]  ->
1000 dha[e]  ->
1000 dttp[e]  ->
1000 fad[e]  ->
1000 fald[e]  ->
1000 HC00250[e]  ->
1000 HC01361[e]  ->
1000 HC01446[e]  ->
1000 cpppg1[e]  ->
1000 itp[e]  ->
1000 udpg[e]  ->
1000 HC00955[e]  ->
1000 C02470[e]  ->
1000 HC00822[e]  ->
1000 HC02193[e]  ->
1000 HC02195[e]  ->
1000 HC02196[e]  ->
1000 HC02191[e]  ->
1000 HC02194[e]  ->
1000 HC02203[e]  ->
1000 HC02217[e]  ->
1000 malcoa[e]  ->
1000 arachcoa[e]  ->
1000 CE4722[e]  ->
1000 CE4723[e]  ->
1000 CE4724[e]  ->
1000 CE2839[e]  ->
1000 CE2838[e]  ->
1000 23cump[e]  ->
1000 CE5788[e]  ->
1000 CE5798[e]  ->
1000 CE5787[e]  ->
1000 CE5791[e]  ->
1000 CE5867[e]  ->
1000 CE4633[e]  ->
1000 CE5854[e]  ->
1000 udpgal[e]  ->
1000 CE0074[e]  ->
1000 CE5853[e]  ->
1001 20.6508 h2o[c] + 20.7045 atp[c] + 0.38587 glu_L[c] + 0.35261 asp_L[c] + 0.036117 gtp[c] + 0.50563
1000 c101coa[c]  ->
1000 doco13ac[e]  ->
1000 octdececoa[c]  ->
```

```
1000 tetdec2coa[c]  ->
1000 tetdece1coa[c]  ->
1000 5HPET[r]  ->
1000 taur[c]  ->
1000 pe_hs[r]  ->
1000 pmtcoa[r]  ->
1000 alaala[e]  ->
1000 bglc[e]  ->
1000 glygly[e]  ->
1000 gum[e]  ->
1000 leugly[e]  ->
1000 pect[e]  ->
1000 psyl[e]  ->
1000 slfcys[e]  ->
1000 dpcoa[e]  ->
1000 oh1[e]  ->
1000 q10[e]  ->
1000 Lcystin[c]  ->
1000 ncam[c]  ->
1000 pnto_R[c]  ->
1000 34hpp[e]  ->
1000 3mob[e]  ->
1000 3mop[e]  ->
1000 4mop[e]  ->
1000 aicar[e]  ->
1000 cbasp[e]  ->
1000 2pg[e]  ->
1000 5hoxindoa[e]  ->
1000 cholp[e]  ->
1000 cyst_L[e]  ->
1000 dmgly[e]  ->
1000 g3pc[e]  ->
1000 gudac[e]  ->
1000 hcys_L[e]  ->
1000 icit[e]  ->
1000 pep[e]  ->
1000 xtsn[e]  ->
1000 3pg[e]  ->
1000 udpglcur[e]  ->
1000 nicrnt[e]  ->
1000 orot5p[e]  ->
1000 glyc3p[e]  ->
1000 acrn[e]  ->
1000 pcrn[e]  ->
1000 lneldccrn[e]  ->
1000 odecrn[e]  ->
1000 stcrn[e]  ->
1000 pmtcrn[e]  ->
1000 hdcecrn[e]  ->
1000 15HPET[e]  ->
1000 3mhis[e]  ->
1000 5HPET[e]  ->
1000 7dhchsterol[e]  ->
```

```
1000 aclys[e]  ->
1000 adpoh[e]  ->
1000 amet[e]  ->
1000 biliverd[e]  ->
1000 C02356[e]  ->
1000 CE0955[e]  ->
1000 CE1556[e]  ->
1000 CE2176[e]  ->
1000 CE7082[e]  ->
1000 forglu[e]  ->
1000 HC00900[e]  ->
1000 hmcr[e]  ->
1000 lnlccrn[e]  ->
1000 lthstrl[e]  ->
1000 mev_R[e]  ->
1000 pe12_hs[e]  ->
1000 pe13_hs[e]  ->
1000 pe15_hs[e]  ->
1000 pe161_hs[e]  ->
1000 pe224_hs[e]  ->
1000 pe226_hs[e]  ->
1000 pedh203_hs[e]  ->
1000 pelinl_hs[e]  ->
1000 peole_hs[e]  ->
1000 pepalm_hs[e]  ->
1000 peste_hs[e]  ->
1000 saccrp_L[e]  ->
1000 xolest205_hs[e]  ->
1000 3moxtyr[e]  ->
1000 5aop[e]  ->
1000 alltn[e]  ->
1000 CE2510[e]  ->
1000 ddca[e]  ->
1000 glyc_R[e]  ->
1000 Lcyst[e]  ->
1000 oaa[e]  ->
1000 ttdcea[e]  ->
1000 bgly[e]  ->
1000 retinal[e]  ->
1000 maltttr[e]  ->
1000 progly[e]  ->
1000 dhbpt[e]  ->
1000 alaargcys[e]  ->
1000 alaasnleu[e]  ->
1000 alahisala[e]  ->
1000 alalysthr[e]  ->
1000 argalaala[e]  ->
1000 argalaphe[e]  ->
1000 argalathr[e]  ->
1000 argarglys[e]  ->
1000 argargmet[e]  ->
1000 argcysgly[e]  ->
1000 argcysser[e]  ->
```

```
1000 argglupro[e]  ->
1000 argleuphe[e]  ->
1000 argphearg[e]  ->
1000 argpromet[e]  ->
1000 argserser[e]  ->
1000 argtyrval[e]  ->
1000 argvalcys[e]  ->
1000 argvaltrp[e]  ->
1000 asnmetpro[e]  ->
1000 asnpheasp[e]  ->
1000 asnphecys[e]  ->
1000 asntyrgly[e]  ->
1000 asntyrphe[e]  ->
1000 aspalaarg[e]  ->
1000 aspasnglu[e]  ->
1000 aspglupro[e]  ->
1000 aspglutrp[e]  ->
1000 asphiscys[e]  ->
1000 asplysglu[e]  ->
1000 aspmetasp[e]  ->
1000 aspvalasn[e]  ->
1000 cysasnmet[e]  ->
1000 cysaspphe[e]  ->
1000 cysglnmet[e]  ->
1000 cysglutrp[e]  ->
1000 cysleuthr[e]  ->
1000 cystyrasn[e]  ->
1000 glnasngln[e]  ->
1000 glnlyslys[e]  ->
1000 glnproglu[e]  ->
1000 glntrpglu[e]  ->
1000 glntyrleu[e]  ->
1000 gluargleu[e]  ->
1000 gluasnleu[e]  ->
1000 glumethis[e]  ->
1000 gluthr[e]  ->
1000 gluthrlys[e]  ->
1000 glutrpala[e]  ->
1000 glyhisasn[e]  ->
1000 glyhislys[e]  ->
1000 glylysphe[e]  ->
1000 glytyrlys[e]  ->
1000 glyvalhis[e]  ->
1000 hisargcys[e]  ->
1000 hisargser[e]  ->
1000 hiscyscys[e]  ->
1000 hisglnala[e]  ->
1000 hisglugln[e]  ->
1000 hisglylys[e]  ->
1000 hislysala[e]  ->
1000 hislysglu[e]  ->
1000 hislysile[e]  ->
1000 hislysval[e]  ->
```

1000 hismetgln[e] ->
1000 hisphearg[e] ->
1000 histrphis[e] ->
1000 ileargile[e] ->
1000 ileasnhis[e] ->
1000 ileglyarg[e] ->
1000 ileprolys[e] ->
1000 ileserarg[e] ->
1000 iletrptyr[e] ->
1000 leualaarg[e] ->
1000 leuasplys[e] ->
1000 leusertrp[e] ->
1000 lyscyshis[e] ->
1000 lysglnphe[e] ->
1000 lyslyslys[e] ->
1000 lyspheile[e] ->
1000 lystyrile[e] ->
1000 lysvalphe[e] ->
1000 lysvaltrp[e] ->
1000 metargleu[e] ->
1000 metasntyr[e] ->
1000 metglntyr[e] ->
1000 metglyarg[e] ->
1000 metmetile[e] ->
1000 metphearg[e] ->
1000 mettrpphe[e] ->
1000 pheasnmet[e] ->
1000 pheasp[e] ->
1000 pheglnphe[e] ->
1000 pheleu[e] ->
1000 pheleuasp[e] ->
1000 pheleuhis[e] ->
1000 phelysala[e] ->
1000 phelyspro[e] ->
1000 phepheasn[e] ->
1000 phephethr[e] ->
1000 pheproarg[e] ->
1000 phesertrp[e] ->
1000 phethrlys[e] ->
1000 phetrpleu[e] ->
1000 phetyr[e] ->
1000 phetyrgln[e] ->
1000 phetyrlys[e] ->
1000 proargcys[e] ->
1000 proasncys[e] ->
1000 proglulys[e] ->
1000 prophe[e] ->
1000 proproarg[e] ->
1000 propropro[e] ->
1000 provalgln[e] ->
1000 serargala[e] ->
1000 serargtrp[e] ->
1000 sercysarg[e] ->

1000 serlyshis[e] ->
1000 serphelys[e] ->
1000 thrglnglu[e] ->
1000 thrilearg[e] ->
1000 thrmetarg[e] ->
1000 thrphearg[e] ->
1000 thrserarg[e] ->
1000 thrtyrmet[e] ->
1000 trpgluleu[e] ->
1000 trpglupro[e] ->
1000 trpglutyr[e] ->
1000 trpglyphe[e] ->
1000 trpglyval[e] ->
1000 trpilelys[e] ->
1000 trpiletrp[e] ->
1000 trpleuval[e] ->
1000 trpmetval[e] ->
1000 trpphe[e] ->
1000 trpproval[e] ->
1000 trpsertyr[e] ->
1000 trpthrglu[e] ->
1000 trpthrile[e] ->
1000 trpvalasp[e] ->
1000 tyrala[e] ->
1000 tyralaphe[e] ->
1000 tyrargglu[e] ->
1000 tyrargser[e] ->
1000 tyrcysgly[e] ->
1000 tyrcysthr[e] ->
1000 tyrglu[e] ->
1000 tyrphetyr[e] ->
1000 tyrvalmet[e] ->
1000 valarggly[e] ->
1000 valhisasn[e] ->
1000 valleuphe[e] ->
1000 vallystyr[e] ->
1000 valphearg[e] ->
1000 valprotrp[e] ->
1000 valserarg[e] ->
1000 valtrpphe[e] ->
1000 valtrpval[e] ->
1000 trpglyasp[e] ->
1000 hxa[e] ->
1000 Lhcystin[e] ->
1000 pe_hs[c] ->
1000 akg[c] ->
1000 bandmt[c] ->
1000 for[c] ->
1000 mi14p[c] ->
1000 pchol_hs[c] ->
1000 C02712[c] ->
1000 C02528[c] ->
1000 HC02191[c] ->

```
1000 HC02192[c]  ->
1000 HC02197[c]  ->
1000 HC02198[c]  ->
1000 HC02220[c]  ->
1000 Tyr_ggn[c]  ->
1000 c226coa[c]  ->
1000 chol[c]  ->
1000 cholate[c]  ->
1000 coa[c]  ->
1000 crvnc[c]  ->
1000 gchola[c]  ->
1000 glygn2[c]  ->
1000 hdca[c]  ->
1000 lnlccoa[c]  ->
1000 retfa[c]  ->
1000 retinol[c]  ->
1000 tchola[c]  ->
1000 tdechola[c]  ->
1000 thmpp[c]  ->
1000 thmtp[c]  ->
1000 tmndnccoa[c]  ->
1000 vitd3[c]  ->
1000 dhcholestanate[c]  ->
1000 thcholstoic[c]  ->
1000 xol7ah3[c]  ->
1000 xol7aone[c]  ->
1000 7klitchol[c]  ->
1000 dchac[c]  ->
1000 CE1273[c]  ->
1000 2obut[e]  ->
1000 acac[e]  ->
1000 but[e]  ->
1000 cgly[e]  ->
1000 co2[e]  ->
1000 cytd[e]  ->
1000 dgsn[e]  ->
1000 din[e]  ->
1000 duri[e]  ->
1000 fe3[e]  ->
1000 fum[e]  ->
1000 glyleu[e]  ->
1000 glyphe[e]  ->
1000 glypro[e]  ->
1000 h[e]  ->
1000 h2o[e]  ->
1000 ins[e]  ->
1000 lac_L[e]  ->
1000 lys_L[e]  ->
1000 na1[e]  ->
1000 o2[e]  ->
1000 orn[e]  ->
1000 ppi[e]  ->
1000 pro_L[e]  ->
```

```
1000 ser_L[e]  ->
1000 so4[e]  ->
1000 thymd[e]  ->
1000 urea[e]  ->
1000 cys_L[e]  ->
1000 his_L[e]  ->
1000 thr_L[e]  ->
1000 gln_L[e]  ->
1000 phe_L[e]  ->
1000 arg_L[e]  ->
1000 nac[e]  ->
1000 cit[e]  ->
1000 etha[e]  ->
1000 fol[e]  ->
1000 glyc[e]  ->
1000 malt[e]  ->
1000 malttr[e]  ->
1000 rib_D[e]  ->
1000 trp_L[e]  ->
1000 xyl_D[e]  ->
1000 34dhpha[e]  ->
1000 ppa[e]  ->
1000 tre[e]  ->
1000 lcts[e]  ->
1000 ade[e]  ->
1000 etoh[e]  ->
1000 phpyr[e]  ->
1000 2h3mv[e]  ->
1000 2hiv[e]  ->
1000 sucsal[e]  ->
1000 3ityr_L[e]  ->
1000 35diotyr[e]  ->
1000 13_cis_retn[e]  ->
1000 CE1617[e]  ->
1000 34dhoxmand[e]  ->
1000 CE5643[e]  ->
1000 n8aspmd[e]  ->
1000 13dampp[e]  ->
1000 12ppd_R[e]  ->
1000 xylu_L[e]  ->
1000 xylu_D[e]  ->
1000 CE0737[e]  ->
1000 hdd2crn[e]  ->
1000 mlthf[e]  ->
1000 sphgn[e]  ->
1000 coke[e]  ->
1000 hdl_hs[e]  ->
1000 HC00005[e]  ->
1000 CE2172[e]  ->
1000 CE5629[e]  ->
1000 gd3_hs[e]  ->
1000 gluside_hs[e]  ->
1000 gm3_hs[e]  ->
```

1000 cmpacna[e] ->
1000 34dhpac[c] ->
1000 ts3[c] ->
1000 gd3_hs[l] ->
1000 k[g] ->
1000 na1[r] ->
1000 pail_hs[e] ->
1000 CE1243[e] ->
1000 CE5026[e] ->
1000 CE1261[e] ->
1000 gd1b_hs[e] ->
1000 nadh[e] ->
1000 sbt_D[e] ->
1000 12dhchol[c] ->
1000 3dhcdchol[c] ->
1000 3dhchol[c] ->
1000 3dhdchol[c] ->
1000 3dhlchol[c] ->
1000 7dhcdchol[c] ->
1000 7dhchol[c] ->
1000 ca3s[c] ->
1000 coprost[c] ->
1000 dca3s[c] ->
1000 gca3s[c] ->
1000 gcdca3s[c] ->
1000 gdca3s[c] ->
1000 gudca3s[c] ->
1000 hyochol[c] ->
1000 icdchol[c] ->
1000 isochol[c] ->
1000 lca3s[c] ->
1000 tca3s[c] ->
1000 tcdca3s[c] ->
1000 tdca3s[c] ->
1000 thyochol[c] ->
1000 tudca3s[c] ->
1000 uchol[c] ->
1000 udca3s[c] ->
1000 hyochol[e] ->
1000 am1ccs[e] ->
1000 am1csa[e] ->
1000 am9csa[e] ->
1000 csa[e] ->
1000 fvs[e] ->
1000 glz[e] ->
1000 lvst[e] ->
1000 mhglz[e] ->
1000 nfd[e] ->
1000 nfdoh[e] ->
1000 ptvstlac[e] ->
1000 pvs[e] ->
1000 tlacfvs[e] ->
1000 tmdm1[e] ->

```
  1000 tripvs[e]  ->
  1000 C13856[e]  ->
  1000 M02956[e]  ->
  1000 M00241[e]  ->
  1000 M00008[e]  ->
  0.05 M00017[e]  ->
  1000 M00019[e]  ->
  0.15 M00117[e]  ->
  1000 M01197[e]  ->
  0.05 M01207[e]  ->
  1000 M01235[e]  ->
  1000 M01238[e]  ->
  1000 M02560[e]  ->
  1000 h2co3[e]  ->
  1000 M02837[e]  ->
  1000 his_L[c]  ->
  1000 ile_L[c]  ->
  1000 leu_L[c]  ->
  1000 lys_L[c]  ->
  1000 met_L[c]  ->
  1000 phe_L[c]  ->
  1000 thr_L[c]  ->
  1000 trp_L[c]  ->
  1000 val_L[c]  ->
  1000 ala_L[c]  ->
  1000 arg_L[c]  ->
  1000 asn_L[c]  ->
  1000 asp_L[c]  ->
  1000 cys_L[c]  ->
  1000 gln_L[c]  ->
  1000 glu_L[c]  ->
  1000 pro_L[c]  ->
  1000 ser_L[c]  ->
  1000 tyr_L[c]  ->
  1000 gly[c]  ->
  1000 4abut[l]  ->
  1000 CE5026[c]  ->
  1000 4glu56dihdind[c]  ->
  1000 dopa[c]  ->
  1000 srtn[c]  ->
  1000 adrnl[c]  ->
  1000 ach[c]  ->
  1000 hista[c]  ->
  1000 nrpphr[c]  ->
  1000 Lkynr[c]  ->
  1000 btn[m]  ->
Upper bounds relaxed
  1000 13_cis_retn[n]  ->
  1000 datp[m]  ->
  1000 dgpi_prot_hs[r]  ->
  1000 dgtp[m]  ->
  1000 dttp[m]  ->
  1000 melanin[c]  ->
```

```
1000 mem2emgacpail_prot_hs[r]  ->
1000 10fthf[e]  ->
1000 10fthf5glu[e]  ->
1000 10fthf6glu[e]  ->
1000 13_cis_retnglc[e]  ->
1000 2hb[e]  ->
1000 34dhoxpeg[e]  ->
1000 34dhphe[e]  ->
1000 5adtststerones[e]  ->
1000 5dhf[e]  ->
1000 5mthf[e]  ->
1000 5thf[e]  ->
1000 6dhf[e]  ->
1000 6thf[e]  ->
1000 abt[e]  ->
1000 acetone[e]  ->
1000 ach[e]  ->
0.45 adrn[e]  ->
1000 amp[e]  ->
 0.1 arach[e]  ->
1000 arachd[e]  ->
1000 bhb[e]  ->
1000 bilirub[e]  ->
 0.5 clpnd[e]  ->
1000 crtstrn[e]  ->
1000 crvnc[e]  ->
1000 dag_hs[e]  ->
1000 dhdascb[e]  ->
1000 dhf[e]  ->
 0.2 dlnlcg[e]  ->
1000 dopa[e]  ->
1000 elaid[e]  ->
1000 estradiol[e]  ->
1000 fuc_L[e]  ->
1000 glyc_S[e]  ->
1000 glygn5[e]  ->
1000 gmp[e]  ->
1000 gsn[e]  ->
1000 hco3[e]  ->
1000 hdca[e]  ->
1000 hdcea[e]  ->
 0.2 hexc[e]  ->
1000 hista[e]  ->
1000 hpdca[e]  ->
1000 inost[e]  ->
1000 ksi_deg1[e]  ->
1000 lac_D[e]  ->
1000 leuktrC4[e]  ->
1000 leuktrD4[e]  ->
1000 leuktrE4[e]  ->
1000 lgnc[e]  ->
1000 lneldc[e]  ->
1000 lnlncg[e]  ->
```

1000 lpchol_hs[e] ->
1000 mag_hs[e] ->
1000 meoh[e] ->
1000 mercplaccys[e] ->
1000 mthgxl[e] ->
1000 nad[e] ->
1000 nrpphr[e] ->
 0.1 nrvnc[e] ->
1000 oagd3_hs[e] ->
1000 ocdca[e] ->
1000 pchol_hs[e] ->
1000 pglyc_hs[e] ->
1000 prostge2[e] ->
1000 rbt[e] ->
1000 retfa[e] ->
1000 retn[e] ->
1000 Rtotal[e] ->
1000 Rtotal2[e] ->
1000 Rtotal3[e] ->
1000 s2l2n2m2masn[e] ->
1000 sl_L[e] ->
1000 tchola[e] ->
1000 thmtp[e] ->
1000 thyox_L[e] ->
1000 tmndnc[e] ->
1000 tststerone[e] ->
1000 tsul[e] ->
1000 udp[e] ->
1000 ump[e] ->
1000 urate[e] ->
1000 vitd3[e] ->
1000 whtststerone[e] ->
1000 xolest_hs[e] ->
1000 xolest2_hs[e] ->
1000 acmana[e] ->
1000 ahdt[e] ->
1000 ctp[e] ->
1000 dgtp[e] ->
1000 dtmp[e] ->
1000 g1p[e] ->
1000 isomal[e] ->
1000 HC01104[e] ->
1000 HC01444[e] ->
1000 HC01577[e] ->
1000 HC01609[e] ->
1000 HC01700[e] ->
1000 orot[e] ->
1000 prpp[e] ->
1000 so3[e] ->
1000 prostgh2[e] ->
1000 prostgi2[e] ->
1000 HC00004[e] ->
1000 HC00822[e] ->

1000 HC02192[e] ->
1000 HC02193[e] ->
1000 HC02220[e] ->
1000 HC02197[e] ->
1000 HC02198[e] ->
1000 HC02187[e] ->
1000 HC02202[e] ->
1000 HC02204[e] ->
1000 coa[e] ->
1000 CE2250[e] ->
1000 CE1943[e] ->
1000 CE2915[e] ->
1000 CE2916[e] ->
1000 CE2917[e] ->
1000 malthp[e] ->
1000 CE2839[e] ->
1000 3ump[e] ->
1000 CE5786[e] ->
1000 CE5789[e] ->
1000 CE5797[e] ->
1000 CE5868[e] ->
1000 CE5869[e] ->
1000 CE4881[e] ->
1000 CE1926[e] ->
1000 crm_hs[e] ->
1000 galside_hs[e] ->
1000 CE1925[e] ->
1000 3bcrn[e] ->
1000 3hdececrn[e] ->
1000 3octdeccrn[e] ->
1000 3octdece1crn[e] ->
1000 c101crn[e] ->
1000 c10crn[e] ->
1000 c4dc[e] ->
1000 c51crn[e] ->
1000 c8crn[e] ->
0.25 docosac[e] ->
1000 tetdec2crn[e] ->
1000 tetdece1crn[e] ->
1000 4abut[n] ->
1000 dchac[e] ->
1000 glgchlo[e] ->
1000 gltcho[e] ->
1000 gumgchol[e] ->
1000 gumtchol[e] ->
1000 pectintchol[e] ->
1000 psylchol[e] ->
1000 psyltchol[e] ->
1000 tdechola[e] ->
1000 fmn[e] ->
1000 pan4p[e] ->
1000 q10h2[e] ->
1000 5HPET[c] ->

1000 Lcystin[c] ->
1000 fol[c] ->
1000 ncam[c] ->
1000 5oxpro[e] ->
1000 ahcys[e] ->
1000 cholp[e] ->
1000 cyst_L[e] ->
1000 dcmp[e] ->
1000 ethamp[e] ->
1000 glyald[e] ->
1000 icit[e] ->
1000 L2aadp[e] ->
1000 Lkynr[e] ->
1000 xmp[e] ->
1000 hLkynr[e] ->
1000 nicrnt[e] ->
1000 argsuc[e] ->
1000 pcrn[e] ->
1000 lneldccrn[e] ->
1000 stcrn[e] ->
1000 3mtp[e] ->
1000 15kprostgf2[e] ->
1000 2oxoadp[e] ->
1000 34hpl[e] ->
1000 3hpp[e] ->
1000 3uib[e] ->
1000 56dura[e] ->
1000 acgly[e] ->
1000 acthr_L[e] ->
1000 C02712[e] ->
1000 C05957[e] ->
1000 C06314[e] ->
1000 C06315[e] ->
1000 C11695[e] ->
1000 CE1273[e] ->
1000 CE1556[e] ->
1000 CE2176[e] ->
1000 CE5304[e] ->
1000 CE6031[e] ->
1000 didecaeth[e] ->
1000 diholineth[e] ->
1000 docohxeth[e] ->
1000 docteteth[e] ->
1000 elaidcrn[e] ->
1000 hepdeceth[e] ->
1000 hexdeceeth[e] ->
1000 hexdiac[e] ->
1000 hxcoa[e] ->
1000 lineth[e] ->
1000 lnlccrn[e] ->
1000 mi1p_D[e] ->
1000 Nacasp[e] ->
1000 oleth[e] ->

```
1000 pailste_hs[e]  ->
1000 pchol2palm_hs[e]  ->
1000 pcholn261_hs[e]  ->
1000 pcholn28_hs[e]  ->
1000 pcholn281_hs[e]  ->
1000 pmeth[e]  ->
1000 sphmyln180241_hs[e]  ->
1000 sphmyln18114_hs[e]  ->
1000 sphmyln18115_hs[e]  ->
1000 sphmyln18116_hs[e]  ->
1000 sphmyln181161_hs[e]  ->
1000 sphmyln18117_hs[e]  ->
1000 sphmyln18118_hs[e]  ->
1000 sphmyln181181_hs[e]  ->
1000 sphmyln18120_hs[e]  ->
1000 sphmyln181201_hs[e]  ->
1000 sphmyln18121_hs[e]  ->
1000 sphmyln18122_hs[e]  ->
1000 sphmyln181221_hs[e]  ->
1000 sphmyln18123_hs[e]  ->
1000 sphmyln1824_hs[e]  ->
1000 sphmyln1825_hs[e]  ->
1000 steeth[e]  ->
1000 tmlys[e]  ->
1000 trideceth[e]  ->
1000 xolest183_hs[e]  ->
1000 abt_D[e]  ->
1000 glyc2p[e]  ->
1000 glyclt[e]  ->
1000 phlac[e]  ->
1000 pser_L[e]  ->
1000 bz[e]  ->
1000 mepi[e]  ->
1000 1mncam[e]  ->
1000 progly[e]  ->
1000 thbpt[e]  ->
1000 itp[n]  ->
1000 alaargcys[e]  ->
1000 alaarggly[e]  ->
1000 alaglylys[e]  ->
1000 alahisala[e]  ->
1000 argalaphe[e]  ->
1000 argarg[e]  ->
1000 argarglys[e]  ->
1000 argcysgly[e]  ->
1000 argcysser[e]  ->
1000 arggluglu[e]  ->
1000 argglygly[e]  ->
1000 arghisthr[e]  ->
1000 arglysasp[e]  ->
1000 argprothr[e]  ->
1000 argserser[e]  ->
1000 argvalcys[e]  ->
```

1000 asnasnarg[e] ->
1000 asncyscys[e] ->
1000 asnphecys[e] ->
1000 asntyrthr[e] ->
1000 aspasnglu[e] ->
1000 aspglu[e] ->
1000 asphispro[e] ->
1000 asplysglu[e] ->
1000 asplyshis[e] ->
1000 aspprolys[e] ->
1000 cysasnmet[e] ->
1000 cysaspphe[e] ->
1000 cyscys[e] ->
1000 cysglnmet[e] ->
1000 cysgluhis[e] ->
1000 cysglutrp[e] ->
1000 cyssermet[e] ->
1000 glnasngln[e] ->
1000 glnhishis[e] ->
1000 glnhislys[e] ->
1000 glnlyslys[e] ->
1000 glnlystrp[e] ->
1000 gluglu[e] ->
1000 gluilelys[e] ->
1000 gluleu[e] ->
1000 glumet[e] ->
1000 glumethis[e] ->
1000 glutrpala[e] ->
1000 glylyscys[e] ->
1000 glylysphe[e] ->
1000 glyvalhis[e] ->
1000 hisasp[e] ->
1000 hiscyscys[e] ->
1000 hisglu[e] ->
1000 hishislys[e] ->
1000 hislysthr[e] ->
1000 hismet[e] ->
1000 hisprolys[e] ->
1000 histrphis[e] ->
1000 ileargile[e] ->
1000 ileasp[e] ->
1000 ileglnglu[e] ->
1000 ileglyarg[e] ->
1000 ileserarg[e] ->
1000 leuasnasp[e] ->
1000 leuleutrp[e] ->
1000 leupro[e] ->
1000 leuproarg[e] ->
1000 leutrp[e] ->
1000 leutrparg[e] ->
1000 leutyrtyr[e] ->
1000 leuval[e] ->
1000 lysargleu[e] ->

1000 lysgluglu[e] ->
1000 lyslyslys[e] ->
1000 lyspheile[e] ->
1000 lystrparg[e] ->
1000 lysvalphe[e] ->
1000 methislys[e] ->
1000 metmetile[e] ->
1000 pheasnmet[e] ->
1000 pheglnphe[e] ->
1000 phelysala[e] ->
1000 phephe[e] ->
1000 phepheasn[e] ->
1000 phephethr[e] ->
1000 phesertrp[e] ->
1000 proargasp[e] ->
1000 procys[e] ->
1000 proglnpro[e] ->
1000 prohis[e] ->
1000 prohistyr[e] ->
1000 proleuarg[e] ->
1000 prolyspro[e] ->
1000 proproarg[e] ->
1000 propropro[e] ->
1000 protrplys[e] ->
1000 protrpthr[e] ->
1000 serargala[e] ->
1000 sercysarg[e] ->
1000 serglyglu[e] ->
1000 serphelys[e] ->
1000 sertrphis[e] ->
1000 thrargtyr[e] ->
1000 thrasntyr[e] ->
1000 thrglntyr[e] ->
1000 thrhishis[e] ->
1000 thrthrarg[e] ->
1000 trpalapro[e] ->
1000 trpargala[e] ->
1000 trpaspasp[e] ->
1000 trpglngln[e] ->
1000 trpglugly[e] ->
1000 trpglyleu[e] ->
1000 trpglyval[e] ->
1000 trphismet[e] ->
1000 trpiletrp[e] ->
1000 trplys[e] ->
1000 trpmetarg[e] ->
1000 trpprogly[e] ->
1000 trpproleu[e] ->
1000 trpthrtyr[e] ->
1000 trptyrgln[e] ->
1000 trptyrtyr[e] ->
1000 tyralaphe[e] ->
1000 tyrasparg[e] ->

```
1000 tyrcysgly[e]  ->
1000 tyrleuarg[e]  ->
1000 tyrthr[e]  ->
1000 tyrtrpphe[e]  ->
1000 tyrtyr[e]  ->
1000 tyrvalmet[e]  ->
1000 valarggly[e]  ->
1000 valleuphe[e]  ->
1000 valphearg[e]  ->
1000 valserarg[e]  ->
1000 valtrpphe[e]  ->
1000 valtrpval[e]  ->
1000 valval[e]  ->
1000 homoval[e]  ->
1000 pe_hs[c]  ->
1000 adprbp[c]  ->
1000 mi145p[c]  ->
1000 band[c]  ->
1000 acgal[e]  ->
1000 acnam[e]  ->
1000 HC00342[e]  ->
1000 pa_hs[e]  ->
1000 CE2934[e]  ->
1000 HC02191[c]  ->
1000 HC02193[c]  ->
1000 HC02194[c]  ->
1000 HC02195[c]  ->
1000 HC02196[c]  ->
1000 Tyr_ggn[c]  ->
1000 btn[c]  ->
1000 coa[c]  ->
1000 fad[c]  ->
1000 lnlc[c]  ->
1000 lnlccoa[c]  ->
1000 nad[c]  ->
1000 odecoa[c]  ->
1000 pmtcoa[c]  ->
1000 retinol[c]  ->
1000 stcoa[c]  ->
1000 tag_hs[c]  ->
1000 thf[c]  ->
1000 tmndnc[c]  ->
1000 dxtrn[e]  ->
1000 dhcholestanate[e]  ->
1000 thcholstoic[e]  ->
1000 xol7ah3[e]  ->
1000 xol7ah3[c]  ->
1000 xol7aone[e]  ->
1000 7klitchol[e]  ->
1000 glcn[e]  ->
1000 acac[e]  ->
1000 adn[e]  ->
1000 akg[e]  ->
```

```
1000 asn_L[e]  ->
1000 asp_L[e]  ->
1000 co2[e]  ->
1000 dad_2[e]  ->
1000 dcyt[e]  ->
1000 fe2[e]  ->
1000 gal[e]  ->
1000 glu_L[e]  ->
1000 glyb[e]  ->
1000 glypro[e]  ->
1000 ile_L[e]  ->
1000 ins[e]  ->
1000 k[e]  ->
1000 lac_L[e]  ->
1000 leu_L[e]  ->
1000 mal_L[e]  ->
1000 met_L[e]  ->
1000 no2[e]  ->
1000 pi[e]  ->
1000 pro_L[e]  ->
1000 ser_L[e]  ->
1000 succ[e]  ->
1000 urea[e]  ->
1000 uri[e]  ->
1000 val_L[e]  ->
1000 pnto_R[e]  ->
1000 gly[e]  ->
1000 cys_L[e]  ->
1000 ala_L[e]  ->
1000 thr_L[e]  ->
1000 gln_L[e]  ->
1000 phe_L[e]  ->
1000 tyr_L[e]  ->
1000 for[e]  ->
1000 nh4[e]  ->
1000 ac[e]  ->
1000 acgam[e]  ->
1000 cit[e]  ->
1000 drib[e]  ->
1000 fru[e]  ->
1000 galt[e]  ->
1000 glcr[e]  ->
1000 glcur[e]  ->
1000 glyc[e]  ->
1000 hxan[e]  ->
1000 malt[e]  ->
1000 ptrc[e]  ->
1000 spmd[e]  ->
1000 trp_L[e]  ->
1000 ura[e]  ->
1000 xan[e]  ->
1000 ppa[e]  ->
1000 pyr[e]  ->
```

```
1000 btn[e]  ->
1000 ade[e]  ->
1000 acald[e]  ->
1000 gua[e]  ->
1000 4abut[e]  ->
1000 taur[e]  ->
1000 phpyr[e]  ->
1000 CE4970[e]  ->
1000 CE4968[e]  ->
1000 vanillac[e]  ->
1000 2m3hvac[e]  ->
1000 3h3mglt[e]  ->
1000 3mglutac[e]  ->
1000 3mglutr[e]  ->
1000 mvlac[e]  ->
1000 ethmalac[e]  ->
1000 methsucc[e]  ->
1000 4ohbut[e]  ->
1000 agm[e]  ->
1000 T4hcinnm[e]  ->
1000 egme[e]  ->
1000 HC02020[e]  ->
1000 chsterols[e]  ->
1000 CE1401[e]  ->
1000 melatn[e]  ->
1000 CE4890[e]  ->
1000 C09642[e]  ->
1000 mhista[e]  ->
1000 ppbng[e]  ->
1000 sphings[e]  ->
1000 CE1918[e]  ->
1000 aact[e]  ->
1000 N1aspmd[e]  ->
1000 fdp[e]  ->
1000 ldl_hs[e]  ->
1000 HC00006[e]  ->
1000 HC00007[e]  ->
1000 HC00008[e]  ->
1000 HC00009[e]  ->
1000 gluside_hs[e]  ->
1000 34dhpe[e]  ->
1000 sph1p[n]  ->
1000 sphs1p[n]  ->
1000 gd3_hs[g]  ->
1000 na1[g]  ->
1000 na1[c]  ->
1000 retn[n]  ->
1000 Ser_Gly_Ala_X_Gly[r]  ->
1000 5cysgly34dhphe[e]  ->
1000 galgluside_hs[e]  ->
1000 mem2emgacpail_prot_hs[e]  ->
1000 glc_D[e]  ->
1000 cdca24g[c]  ->
```

1000 cdca3g[c] ->
1000 dca24g[c] ->
1000 hca24g[c] ->
1000 hca6g[c] ->
1000 hyochol[c] ->
1000 lca24g[c] ->
1000 lca3g[c] ->
1000 12dhchol[e] ->
1000 3dhcdchol[e] ->
1000 3dhchol[e] ->
1000 3dhdchol[e] ->
1000 3dhlchol[e] ->
1000 7dhcdchol[e] ->
1000 7dhchol[e] ->
1000 ca3s[e] ->
1000 coprost[e] ->
1000 dca3s[e] ->
1000 gca3s[e] ->
1000 gcdca3s[e] ->
1000 gdca3s[e] ->
1000 gudca3s[e] ->
1000 icdchol[e] ->
1000 isochol[e] ->
1000 lca3s[e] ->
1000 tca3s[e] ->
1000 tcdca3s[e] ->
1000 tdca3s[e] ->
1000 thyochol[e] ->
1000 tudca3s[e] ->
1000 uchol[e] ->
1000 udca3s[e] ->
1000 3ispvs[e] ->
1000 56dhpvs[e] ->
1000 6epvs[e] ->
1000 6melvst[e] ->
1000 am1accs[e] ->
1000 am1alcs[e] ->
1000 am4n9cs[e] ->
1000 am4ncs[e] ->
1000 crglz[e] ->
1000 deoxfvs[e] ->
1000 dhglz[e] ->
1000 dspvs[e] ->
1000 nfdlac[e] ->
1000 nfdnpy[e] ->
1000 ptvst[e] ->
1000 thrfvs[e] ->
1000 tmdm5[e] ->
1000 M01807[e] ->
1000 M00503[e] ->
1000 M01820[e] ->
1000 M00510[e] ->
1000 M00003[e] ->

```
  1000 M00010[e]  ->
   0.2 M02457[e]  ->
  0.05 M03045[e]  ->
  1000 M02561[e]  ->
  1000 M01111[e]  ->
  1000 M01966[e]  ->
  1000 M01989[e]  ->
  1000 gpi_sig[e]  ->
  1000 glu_L[c]  ->
  1000 tyr_L[c]  ->
  1000 CE4888[c]  ->
  1000 4abut[c]  ->
  1000 kynate[c]  ->
  1000 tym[c]  ->
  1000 cbl2[m]  ->
  1000 protein[c]  ->
```

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 = relaxedFBA(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 relaxedFBA 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')
```

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.