Nutrition Algorithm Walkthrough

Authors: Bronson R. Weston and Bram Nap, University of Galway.

```
%initCobraToolbox()
changeCobraSolver('ibm_cplex') % change solver if necessary
```

```
CPXPARAM_Output_WriteLevel                       3
CPXPARAM_Output_CloneLog                         -1
Found incumbent of value 0.000000 after 0.00 sec. (0.00 ticks)

Root node processing (before b&c):
  Real time             =    0.00 sec. (0.00 ticks)
Parallel b&c, 16 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.00 sec. (0.00 ticks)
Could not find installation of tomlab_cplex, so it cannot be tested
Original LP has 1 row, 2 columns, 1 non-zero
Objective value = 0
OPTIMAL SOLUTION FOUND BY LP PRESOLVER

 > [glpk] Primal optimality condition in solveCobraLP satisfied.Could not find installation of mosek, so i

   ------------------------------------------------------------
   pdco.m                     Version pdco5 of 15 Jun 2018
   Primal-dual barrier method to minimize a convex function
   subject to linear constraints Ax + r = b,  bl <= x <= bu

   Michael Saunders       SOL and ICME, Stanford University
   Contributors:     Byunggyoo Kim (SOL), Chris Maes (ICME)
                     Santiago Akle (ICME), Matt Zahr (ICME)
                     Aekaansh Verma (ME)
   ------------------------------------------------------------

The objective is linear
The matrix A is an explicit sparse matrix

m        =        1     n        =        2      nnz(A)  =        1
max |b | =        0     max |x0| =  1.0e+00      xsize   =  1.0e+00
max |y0| =        1     max |z0| =  1.0e+00      zsize   =  1.0e+00

x0min    =        1     featol   =  1.0e-06      d1max   =  1.0e-04
z0min    =        1     opttol   =  1.0e-06      d2max   =  5.0e-04
mu0      =  1.0e-01     steptol  =     0.99     bigcenter=     1000

LSMR/MINRES:
atol1    =  1.0e-10     atol2    =  1.0e-15      btol    =  0.0e+00
conlim   =  1.0e+12     itnlim   =       10      show    =        0

Method   =        2     (1 or 11=chol  2 or 12=QR  3 or 13=LSMR  4 or 14=MINRES 21=SQD(LU)  22=SQD(MA57))
Eliminating dy before dx


Bounds:
  [0,inf]  [-inf,0]  Finite bl  Finite bu  Two bnds   Fixed    Free
        0         0          0          0         0       2       0
  [0, bu]  [bl,  0]  excluding fixed variables
        0         0

Itn   mu stepx stepz  Pinf  Dinf  Cinf   Objective    nf  center       QR
```

```
  0                  -6.6 -99.0  -Inf  1.2500000e-07        1.0
  1 -1.0 1.000 1.000 -99.0 -99.0  -Inf  0.0000000e+00  1     1.0        1
  2 -3.0 1.000 1.000 -99.0 -99.0  -Inf  0.0000000e+00  1     1.0
  3 -5.0 1.000 1.000 -99.0 -99.0  -Inf  0.0000000e+00  1     1.0
  4 -7.0 1.000 1.000 -99.0 -99.0  -Inf  0.0000000e+00  1     1.0
   Converged

max |x| =     0.000    max |y| =     0.000    max |z| =     0.000   scaled
max |x| =     0.000    max |y| =     0.000    max |z| =     0.000 unscaled
max |x| and max |z| exclude fixed variables
PDitns  =         4     QRitns =         0    cputime =       0.3

Distribution of vector     x         z
[      1,     10 )         0         2
[    0.1,      1 )         0         0
[   0.01,    0.1 )         0         0
[  0.001,   0.01 )         0         0
[ 0.0001,  0.001 )         0         0
[  1e-05, 0.0001 )         0         0
[  1e-06,  1e-05 )         0         0
[  1e-07,  1e-06 )         0         0
[  1e-08,  1e-07 )         0         0
[      0,  1e-08 )         2         0
Elapsed time is 0.272940 seconds.

 > [pdco] Primal optimality condition in solveCobraLP satisfied.
 > [pdco] Dual optimality condition in solveCobraLP satisfied.
Could not find installation of quadMinos, so it cannot be tested
Could not find installation of dqqMinos, so it cannot be tested
Could not find installation of cplex_direct, so it cannot be tested

 > [cplexlp] Primal optimality condition in solveCobraLP satisfied.
 > [cplexlp] Dual optimality condition in solveCobraLP satisfied.
Could not find installation of tomlab_snopt, so it cannot be tested
removing: C:\Users\brons\BronsonFork\fork-cobratoolbox\src\analysis\thermo\componentContribution\new
removing: C:\Users\brons\BronsonFork\fork-cobratoolbox\src\analysis\thermo\groupContribution\new
removing: C:\Users\brons\BronsonFork\fork-cobratoolbox\src\analysis\thermo\inchi\new
removing: C:\Users\brons\BronsonFork\fork-cobratoolbox\src\analysis\thermo\molFiles\new
removing: C:\Users\brons\BronsonFork\fork-cobratoolbox\src\analysis\thermo\protons\new
removing: C:\Users\brons\BronsonFork\fork-cobratoolbox\src\analysis\thermo\trainingModel\new

 > changeCobraSolver: IBM ILOG CPLEX interface added to MATLAB path.
 > isCompatible: The solver compatibility cannot be verified with MATLAB R2019b.
ans = logical
   1
```

```
clear options
```

Background:

The nutrition algorithm seeks to identify the minimal changes to a diet/feed/medium to achieve the largest shift in flux for one or more reactions of interest (ROIs). The algorithm produces points when adding or removing nutrients from the diet/feed/medium. It also produces or consumes points when minimizing or maximizing ROI fluxes, respectively. Overall, the goal of the algorithm is to minimize the total points accumulated (*p*), such that:

$$p = w^{a} v^{a} + w^{r} v^{r} + (m \odot w^{roi}) v^{roi}.$$

w^a and *v*^a denote the weight and the flux of the “Food Added” dietary reactions, respectively. *w*^r and *v*^r denote to the weight and the flux of the “Food Removal” reactions, respectively. w^roi and v^roi are the weight

and flux for the ROIs. By default, *w* and *v* are vectors of ones, unless manually weighted via the optional struct inputs, as discussed later. Finally, *m* is a vector defining if an ROI is to be minimized or maximized. Thus, for a given ROI, the corresponding element in *m* will be equal to -1 or 1 depending on if the goal is to maximize or minimize the ROI, respectively. For the full technical details pertaining to the nutrition algorithm formulation, see Weston and Thiele, 2022 [1].

The nutrition algorithm is implemented in the nutritionAlgorithm.m function in the COBRA Toolbox [2]. The function is highly versatile, allowing the user to customize their solution specifications as they feel fit. In this walkthrough, the different strategies for utilizing the nutrition algorithm will be demonstrated on a genome-scale metabolic reconstruction (GEM) of Atlantic salmon (*Salmo solar),* called SALARECON [3].

Setting up GEM for use:

A typical GEM includes a compartment representing extracellular space and a series of exchange reactions that permit flux, both, into and out of the extracellular space. For the nutrition algorithm to run properly two modifications are required to this "typical GEM". First, all reversable exchange reactions must be broken apart into two seperate reactions, each modelling a different direction of flux. Second, all reactions permitting flux of nutrients into the model must be labeled with 'Diet_EX_' at the beginning of the reaction name to be recognized as dietary reactions (i.e., targeted reactions) by the algorithm. For instance, an exchange reaction for glutamate labelled 'EX_glu[e]' should be labeled 'Diet_EX_glu[e]'. It is also recommended that every dietary reaction has a corresponding "Exit" reaction (i.e., a secretion reaction that allows flux of the nutrient out of the model) and that diet reaction fluxes have a fixed flux (i.e., lb=ub). This option is favored over flexible exchange bounds as it permits the algorithm to analyze and optimize an initial fixed "diet" while maintianing flexible exchange flux by allowing the organism to choose not to uptake certain nutrieints through the exit reactions.

To convert a typical GEM, as described earleir, into a suitable format for the nutrition algorithm, the user can utilize the function, convert_EX_to_diet, as follows:

```
load('NAW_salarecon.mat','model'); %initial salmon model has typical EX_
reactions
mainSalmon= convert_EX_to_diet(model); % convert salmon to model with
dietary reactions
```

To briefly demonstrate the equivalence of the two models, let's simulate both models optimizing for biomass :

```
sln1=optimizeCbModel(model); %Optimize "typical" GEM with exchange reactions
sln1.f
```

```
ans = 1.6135
```

```
sln2=optimizeCbModel(mainSalmon); %Optimize diet-based model
sln2.f
```

```
ans = 1.6135
```

nutritionAlgorithm.m Inputs and Outputs:

The nutritionAlgorithm.m function takes four inputs. (1) the GEM (model), (2) a cell array containing the ROIS (or a string is acceptible if using only one ROI), (3) a cell array containing the minimization and maximization goals for each reaction of interest (a string is acceptible if using only one ROI) and (4) a struct containing special augmentation commands to the algorithm (options) which will be discussed in more detail in the following sections. Note that the options struct is only necessary for custimizing the algorithm performance, and the algorithm runs sufficiently with only the first three inputs as well. Shown below is an example where the nutrition algorithm is asked to maximize the Biomass reaction in the mainSalmon model:

```
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,'Biomass','max');
```

```
_____________________________________________________
Solution points =-1.7993
0.30286 come from diet
-2.1021 come from rois
menuChanges = 1×2 table
```

	Food Rxn	Flux
1	'Food_Added_EX_pi_e'	-0.3029

```
Points Simulation Solution:
   Biomass flux = 2.1021
Detailed Analysis:
 Biomass
   Original Diet RoI range = 0:1.6135
   New Diet RoI range = 0:2.1021
_____________________________________________________
```

The nutrition algorithm outputs six objects and prints a display containing details of the solution. First, let's cover the display contents. The first three lines details the point value produced by the solution and the number of points produced (positive value) or consumed (negative) value by the diet and the ROIs. A table is also displayed regarding the recommended dietary changes and a description of the reaction of interest fluxes is provided. In this case, the algorithm recommends adding 0.3 mmol of phosphate to the diet to acheive a Biomass flux of 2.1.

Finally, a more in depth analysis is provided that examines the potential range in fluxes for the ROIs on the original diet compared to the new diet. This describes that a flux variablity of 0 to 1.61 grams of Biomass flux was achievable on the original diet in comparision to 0 to 2.1 grams on the new diet.

The outputs of the model include (1) an updated version of the original inputted model that includes changes to the bounds of the dietary reactions to reflect suggestions of the algorithm (newDietModel), (2) a version of the model (pointsModel) that is augmented to include Food_Added, Food_Removed and points relevant reactions (i.e., the model that is actually used to find the optimal diet), (3) the flux of the reactions of interest in the points solution (roiFlux), (4) the overall points model solution (pointsModelSln), (5) a summary of the optimal dietary changes (menuChanges) and (6) a struct containing the solution results for each simulation behind the detailed analysis. Each of outputs can be helpful to explore the model solution. For instance, pointsModel and pointsModelSln can be used together to explore the points model solution:

```
surfNet(pointsModel,'point[P]',[],pointsModelSln.v)
```

```
Met #535  point[P], point[P], , metCharges: NaN
```

```
Consuming reactions with non-zero fluxes :
  #895  Point_EX_roiPoints[roiP]_[P] (-2.102142), Bd: -1000000 / 1000000, Point_EX_roiPoints[roiP]_[P], gr
roiPoint[roiP]   <-   point[P]
Producing reactions with non-zero fluxes :
  #870  Point_EX_unitOfFoodChange[dP]_[P] (0.30286), Bd: -1000000 / 1000000, Point_EX_unitOfFoodChange[dP]
unitOfFoodChange[dP]   ->   point[P]
  #871  Point_EX_Point[P] (-1.799282), Bd: -1000000 / 1000000, Point_EX_Point[P], grRules:
point[P]   <-

Show previous steps...
```

Manipulating Nutrition Algorithm Outputs:

There are two ways that the outputs of the algorithm can be manipulated. The first is that the command window outputs can be turned off using the options struct as follows:

```
options.display='off';
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,'Biomass','max',options);
```

Another option is to turn off the detailed analysis. This is useful to enhance the speed of the function if the user is not interested in investigating the solution flux variability:

```
options.slnType='Quick';
tic
nutritionAlgorithm(mainSalmon,'Biomass','max',options);
qSln=toc
```

```
qSln = 0.2834
```

```
options.slnType='Detailed';
tic
nutritionAlgorithm(mainSalmon,'Biomass','max',options);
dSln=toc
```

```
dSln = 0.4146
```

Weighting Reaction of Interest:

As the nutrition algorithm aims to strike a balance between shifting diet and driving metabolic change, it may be of the interest for the user to manipulate how the algorithm finds that balance. To do so, an easy method is to adjust the weighting of the reaction of interest through the options struct variable. While the default weighting for each ROI is equal to one, in this example we show how to increase the weight of ‘Biomass’ to five.

```
clear options
options.roiWeights=5;
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,'Biomass','max',options);
```

```
Solution points =-12.8829
9.5966 come from diet
-22.4795 come from rois
menuChanges = 4×2 table
```

	Food Rxn	Flux
1	'Food_Added_EX_FM_e'	-6.6410
2	'Food_Added_EX_chol_e'	-1.1622
3	'Food_Added_EX_pi_e'	-1.7865
4	'Food_Added_EX_thymd_e'	-0.0069

```
Points Simulation Solution:
   Biomass flux = 4.4959
Detailed Analysis:
 Biomass
   Original Diet RoI range = 0:1.6135
   New Diet RoI range = 0:4.4959
```

Under these new weighting conditions (Fig 2B) the algorithm recommends adding fish meal, choline, phosphate and thymidine to achieve a new biomass flux of 4.5 grams a day. The initial default weighting of one resulting in the recomendation of only adding phosphate to achieve biomass flux of 2.1 grams a day. In short, increasing the weight of a reaction of interest encourages the algorithm to find more extreme shifts in a ROI at the cost of permitting larger changes to the initial diet.

Targeting Multiple reactions of Interest:

Perhaps the most powerful feature of the nutrition algorithm is its ability to optimize feed to target multiple ROIs simultaneously. In the case of salmon, a farmer might like to identify a feed that maximizes growth rate and minimizes nitrogen waste at the same time. This scenario can be specified as follows:

```
clear options
mainSalmon=addElementTracker(mainSalmon,'N','Exit'); % Add nitrogen waste
reaction to track nitrogen excretion rate
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,{'Biomass','N_Track'},{'max','min'});
```

```
Solution points =0.12019
1.4418 come from diet
-1.3216 come from rois
menuChanges = 2×2 table
```

	Food Rxn	Flux
1	'Food_Added_EX_met__L_e'	-1.2322

	Food Rxn	Flux
2	'Food_Added_EX_pi_e'	-0.2096

```
Points Simulation Solution:
   Biomass flux = 1.9517
   N_Track flux = 0.63013
Detailed Analysis:
 Biomass
   Original Diet RoI range = 0:1.6135
   New Diet RoI range = 0:1.9517
 N_Track
   Original Diet RoI range = 3.7975:9.8893
   New Diet RoI range = 0.63013:9.1396
```

As you can see, the algorithm simultaneously optimized the diet such that the new diet's corresponding nitrogen waste flux is reduced the new biomass is greater than the original.

Note that, in general this analysis assumes that an organism's objective function is maximized and the variability analyis provides the flux solutions feasible at that maximal objective function flux. Thus, the nitrogen waste reported for the original diet and the new diet corresponded to a biomass flux of 1.6135 and 1.9517, respectively. However, if the ROI is the objective function, such as in this case, then the analysis is not restricted to a maximal objective function flux. This is why the biomass is reported as a range of values rather than just the maximal.

Limiting Solution Space:

The nutrition algorithm tracks the points produced from adding and removing food. The user can leverage this to to impose specific limits to the solution space to restrict the flux of total points produced from either adding food or removing it:

```
clear options
options.foodAddedLimit=1;
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,{'Biomass','N_Track'},{'max','min'},options);
```

```
Solution points =0.2006
1.3975 come from diet
-1.1969 come from rois
menuChanges = 3×2 table
```

	Food Rxn	Flux
1	'Food_Added_EX_met__L_e'	-0.8714
2	'Food_Added_EX_pi_e'	-0.1286

	Food Rxn	Flux
3	'Food_Removed_EX_FM_e'	-0.3975

```
Points Simulation Solution:
   Biomass flux = 1.8209
   N_Track flux = 0.62399
Detailed Analysis:
 Biomass
   Original Diet RoI range = 0:1.6135
   New Diet RoI range = 0:1.8209
 N_Track
   Original Diet RoI range = 3.7975:9.8893
   New Diet RoI range = 0.62399:6.7058
```

If you take the summation of flux, you will find that it adds up to one for the foodAdded reactions, but it also removes food as well. If you would like to further restrict the solution to not remove any food you can do so as follows:

```
options.foodAddedLimit=1;
options.foodRemovedLimit=0;
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,{'Biomass','N_Track'},{'max','min'},options);
```

```
Solution points =0.49619
1 come from diet
-0.50381 come from rois
menuChanges = 2×2 table
```

	Food Rxn	Flux
1	'Food_Added_EX_met__L_e'	-0.7743
2	'Food_Added_EX_pi_e'	-0.2257

```
Points Simulation Solution:
   Biomass flux = 1.9777
   N_Track flux = 1.4738
Detailed Analysis:
 Biomass
   Original Diet RoI range = 0:1.6135
   New Diet RoI range = 0:1.9777
 N_Track
   Original Diet RoI range = 3.7975:9.8893
   New Diet RoI range = 1.4738:8.5298
```

Customizing Weighting Scheme of Targeted Food Items:

In many scenarios, it may be desirable to weight individual food items or to restrict the solution to only adding specific food items of interest. These types of modifications can be made using the optional variable, targetedFoodItems, which is a cell array where the specific dietary reactions are listed in the first column and the corresponding weights are listed in the second. Let's restrict our solution space to only consider

manipulating the feed rate of fish meal (FM), soybean meal (SBM) or black soldier fly (BSF) meal and assign each variable a weight of one:

```
clear options
options.targetedDietRxns={'Diet_EX_FM_e',1;'Diet_EX_SBM_e',1;'Diet_EX_BSF_e',
1} ;
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,{'Biomass','N_Track'},{'max','min'},options);
```

```
Solution points =0.70635
1.0803 come from diet
-0.37393 come from rois
menuChanges = 2×2 table
```

	Food Rxn	Flux
1	'Food_Removed_EX_BSF_e'	-0.8355
2	'Food_Removed_EX_FM_e'	-0.2448

```
Points Simulation Solution:
   Biomass flux = 1.6135
   N_Track flux = 1.2396
Detailed Analysis:
 Biomass
   Original Diet RoI range = 0:1.6135
   New Diet RoI range = 0:1.6135
 N_Track
   Original Diet RoI range = 3.7975:9.8893
   New Diet RoI range = 1.2396:2.8432
```

In this case algorithm suggests optimal amounts of black soldier fly and fish meal to remove from the diet to maintain growth rate and simultaneously reduce nitrogen waste. Note that the model initial state specifies a feed of 1 gram of each type of meal (i.e., fish meal, soybean meal and black soldier fly meal).

It may also be of interest to specify the weights of a few dietary reactions but assign all other dietary reactions to a specific weight. In this case, let's set all of the feed meals a weight of 1 and all other dietary reactions a weight of 0.1:

```
options.targetedDietRxns={'Diet_EX_FM_e',1;'Diet_EX_SBM_e',1;'Diet_EX_BSF_e',
1; 'All', 0.1} ;
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,{'Biomass','N_Track'},{'max','min'},options);
```

```
Solution points =-6.335
28.3224 come from diet
-34.6574 come from rois
menuChanges = 17×2 table
```

	Food Rxn	Flux
1	'Food_Added_EX_arg__L_e'	-5.3581
2	'Food_Added_EX_chol_e'	-19.0223

	Food Rxn	Flux
3	'Food_Added_EX_cysi__L_e'	-0.0611
4	'Food_Added_EX_glc__D_e'	-146.9912
5	'Food_Added_EX_his__L_e'	-2.1419
6	'Food_Added_EX_ile__L_e'	-6.5778
7	'Food_Added_EX_leu__L_e'	-9.9967
8	'Food_Added_EX_lys__L_e'	-9.8247
9	'Food_Added_EX_met__L_e'	-1.6330
10	'Food_Added_EX_phe__L_e'	-4.5296
11	'Food_Added_EX_pi_e'	-20.4799
12	'Food_Added_EX_thr__L_e'	-6.5001
13	'Food_Added_EX_thymd_e'	-24.4230
14	'Food_Added_EX_trp__L_e'	-0.9257

:
:

```
Points Simulation Solution:
   Biomass flux = 34.6574
   N_Track flux = 0
Detailed Analysis:
 Biomass
   Original Diet RoI range = 0:1.6135
   New Diet RoI range = 0:34.6574
 N_Track
   Original Diet RoI range = 3.7975:9.8893
   New Diet RoI range = 0:9.8048e-12
```

Customizing Weighting of Food Removal Reactions:

nutritionAlgorithm.m also allows a user to explicitely modify the weighting of food removal reactions. By default, food removal reactions are created for all dietary reactions with a lb < 0 and each food removal reaction is assigned a weight of one. Before demonstrating the methods to customize food removal weighting, let's make things interesting by assigning a weighting scheme that results in a solution that identifies flux for food removal and food adding reactions:

```
clear options
options.roiWeights=[50 10];
options.targetedDietRxns={'Diet_EX_FM_e',0.1;'Diet_EX_SBM_e',0.1;'Diet_EX_BSF
_e',0.1; 'All' 15};
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,{'Biomass','N_Track'},{'max','min'},options);
```

```
Solution points =-73.5666
10.0336 come from diet
-83.6002 come from rois
menuChanges = 6×2 table
```

	Food Rxn	Flux
1	'Food_Added_EX_BSF_e'	-0.3373
2	'Food_Added_EX_met__L_e'	-0.4859
3	'Food_Added_EX_pi_e'	-0.0467
4	'Food_Removed_EX_FM_e'	-0.5118
5	'Food_Removed_EX_SBM_e'	-1.0000
6	'Food_Removed_EX_chol_e'	-0.5000

```
Points Simulation Solution:
   Biomass flux = 1.6888
   N_Track flux = 0.083825
Detailed Analysis:
 Biomass
   Original Diet RoI range = 0:1.6135
   New Diet RoI range = 0:1.6888
 N_Track
   Original Diet RoI range = 3.7975:9.8893
   New Diet RoI range = 0.083825:3.5572
```

The solution specifies removal of choline, soybean meal and fish meal. But what if an aquaculture farmer is only interested in removing fishmeal from the diet? This can be specified by utilizing the foodRemovalWeighting struct field on the options variable. Just like the targetedDietRxns struct field, the foodRemovalWeighting will limit the solution space to only the items entered into the cell array.

```
options.foodRemovalWeighting={'Diet_EX_SBM_e',1};
```

```
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,{'Biomass','N_Track'},{'max','min'},options);
```

```
Solution points =-71.902
50.2777 come from diet
-122.1797 come from rois
menuChanges = 4×2 table
```

	Food Rxn	Flux
1	'Food_Added_EX_BSF_e'	-2.3743
2	'Food_Added_EX_met__L_e'	-2.6994
3	'Food_Added_EX_pi_e'	-0.5700
4	'Food_Removed_EX_SBM_e'	-1.0000

```
Points Simulation Solution:
   Biomass flux = 2.5332
   N_Track flux = 0.4478
Detailed Analysis:
 Biomass
   Original Diet RoI range = 0:1.6135
   New Diet RoI range = 0:2.5332
 N_Track
   Original Diet RoI range = 3.7975:9.8893
   New Diet RoI range = 0.4478:17.8949
```

It is also an option to tie the foodRemovalWeighting scheme directly to the targetedDietRxns array. By entering the string 'ditto' the weighting becomes equivalent between the two cell arrays:

```
options.foodRemovalWeighting='ditto';
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,{'Biomass','N_Track'},{'max','min'},options);
```

```
Solution points =-74.0568
48.9333 come from diet
-122.9901 come from rois
menuChanges = 5×2 table
```

	Food Rxn	Flux
1	'Food_Added_EX_BSF_e'	-3.3904
2	'Food_Added_EX_met__L_e'	-2.6563
3	'Food_Added_EX_pi_e'	-0.5700
4	'Food_Removed_EX_FM_e'	-1.0000

	Food Rxn	Flux
5	'Food_Removed_EX_SBM_e'	-1.0000

```
Points Simulation Solution:
   Biomass flux = 2.5332
   N_Track flux = 0.36676
Detailed Analysis:
 Biomass
   Original Diet RoI range = 0:1.6135
   New Diet RoI range = 0:2.5332
 N_Track
   Original Diet RoI range = 3.7975:9.8893
   New Diet RoI range = 0.36676:17.1517
```

Other string inputs for foodRemovalWeighting include 'inverse' and 'ones'. Inverse simply applies the inverse weights as those in targetedDietRxns and 'ones' keeps the same dietary reactionsn targeted by targetedDietRxns but assigns weights of ones to the removal reactions.

Note that in the last two examples, the new diet resulted in a nitrogen waste flux that has a much larger possible flux range than the original diet. Indeed, the lower limit of flux was reduced compared to the original diet, but the upper limit has increased. This is a consequence of the fact that the nutrition algorithm only seeks to optimize one bound. In the case of minimization, the nutrition algorithm optimizes the lower limit of flux and for maximization, it optimizes the upper limit. For more details, please see Weston and Thiele, 2022 [1].

Optimizing For Profit:

Importantly, weighting can either be abstract or tied to real attributes. The default weighting scheme is abstract, as the weights do not correspond to anything palpable. However, it may be of interest to tie weights to quantitatively precise measurements, such as cost. In fact, if both the food item weights and the ROI weights are assigned the same quantitative units, the nutrition algorithm then becomes a method for optimizing towards that quantitative unit. In the case of salmon farming, the goal is likely to optimize profit. Three apsects to profit are the cost of feed, the costs associated with nitrogen waste, and the net salmon meat produce/ sale price. As salmon is sold per mass, the biomass function can be assigned a realistic value to represent the selling price of the fish per gram. In this case, we choose €10 per kg. We can then weight the nitrogen waste reaction with an estimated cost of €0.2 per mol. Finally, the cost of each type of feed can be set to the cost per a gram. We use rough estimates of €1.3 per kg of fishmeal, €0.6 per kg of soybean meal and €3 per kg of black soldier fly meal. As a result, each “point” consumed by the model corresponds to a euro of profit and the algorithm identifies the solution that maximizes the net profit.

```
clear options
mainSalmonTemp=changeRxnBounds(mainSalmon,{'Diet_EX_FM_e'},0,'b');
mainSalmonTemp=changeRxnBounds(mainSalmonTemp,{'Diet_EX_SBM_e'},0,'b');
mainSalmonTemp=changeRxnBounds(mainSalmonTemp,{'Diet_EX_BSF_e'},0,'b');
options.roiWeights=[10/1000 0.2/1000]; % divided by 1000 to convert from kg
to g (for biomass ROI) and mol to mmol (for nitrogen excretion ROI)
```

```
options.targetedDietRxns={'Diet_EX_FM_e',1.3/1000;'Diet_EX_SBM_e',0.6/1000;'D
iet_EX_BSF_e',3/1000}; %Weights devided by 1000 to convert from kg to grams
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmonTemp,{'Biomass','N_Track'},
{'max','min'},options);
```

```
Solution points =-0.014127
0.001711 come from diet
-0.015838 come from rois
menuChanges = 2×2 table
```

	Food Rxn	Flux
1	'Food_Added_EX_FM_e'	-0.4428
2	'Food_Added_EX_SBM_e'	-1.8922

```
Points Simulation Solution:
   Biomass flux = 1.6135
   N_Track flux = 1.4833
Detailed Analysis:
 Biomass
   Original Diet RoI range = 0:0
   New Diet RoI range = 0:1.6135
 N_Track
   Original Diet RoI range = 1.5:1.5
   New Diet RoI range = 1.4833:3.458
```

Here, the algorithm recommends a combination of fish meal and soybean meal at roughly a 1:5 ratio to achieve an estimate profit of €0.015 per day per a fish. Note that the model is set to reflect the growth rate of a 100 gram fish, so as the fish grows larger the feed demand, nitrogen excretion rate and profit margin will all increase linearly.

Also note here that the dietary flux of all feed components was intentionally set to zero before simulation. This ensures that all feed consumed by the model produces points (i.e., is correlated to a price) via the foodAdded reactions.

Targetting a Metabolite:

One may also be interested in optimizing for the synthesis or consumption rate of a particular metabolite. As such, nutritionAlgorithm.m takes any metabolite inputs and creates a sink or demand reaction depending on if the goal is to minimize (i.e., optimize for consumption) or maximize (i.e., optimize for synthesis) the metabolite, respectively. This feature is demonstrated below utilizing the metabolite, 'Lipids[c]':

```
clear options
options.roiWeights=100; %weight increased here from default to ensure
solution is interesting
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,{'Lipids[c]'},{'max'},options);
```

```
Solution points =-4452.5621
```

```
720.4411 come from diet
-5173.0032 come from rois
menuChanges = 6×2 table
```

	Food Rxn	Flux
1	'Food_Added_EX_FM_e'	-4.6328
2	'Food_Added_EX_chol_e'	-69.9286
3	'Food_Added_EX_fru_e'	-528.2909
4	'Food_Added_EX_pi_e'	-70.4732
5	'Food_Added_EX_thymd_e'	-47.0914
6	'Food_Added_EX_uri_e'	-0.0242

```
Points Simulation Solution:
   DM_Lipids[c] flux = 51.73
Detailed Analysis:
 Original objective max flux =1.6135 & New objective max flux =7.8699
 DM_Lipids[c]
   Original Diet RoI range = 0:9.1038e-14
   New Diet RoI range = 0:47.8015
```

Note that the ROI displayed is 'DM_Lipids[c]' which indicates that it is a demand reaction for the 'Lipid[c]' metabolite.

Objective Flux Scalar:

Sometimes a reaction that the algorithm is asked to optimize can conflict with the flux of an organism's objective function. One way to ensure that the organism's objective function is simultaneously optimized with another ROI is by specifying the objective function as another ROI within the algorithm, as demonstrated in earlier examples. However, another method is through the objective flux scalar (OFS). The OFS is a variable that confines the algorithm to identify a solution in the context of a specific flux for an organism's objective function. An OFS of one (the default value) means that the nutrition algorithm will confine its solution to have a lower bound for the organism's objective function that is equal to the maximum flux before a dietary change. This ensures that the nutrition algorithm is constrained to make predictions that align with the current objective function performance, and to not identify solutions that conflict with the objective function flux. In some cases, however, it may be permissible to allow a reduced flux for the organism's objective function. In such cases, one may manipulate the OFS. By reducing the OFS to 0.5, the nutrition algorithm constrains its solution such that a flux of half the initial objective function flux is permissable. By reducing the OFS to zero, the nutrition algorithm no longer considers the organism's objective function in it's solution (unless it is specified as an ROI). This concept is demonstrated in the code below. For more details regarding OFS, see the Methods section of Weston and Thiele [1].

```
clear options
options.roiWeights=10;
```

```
options.targetedDietRxns={'Diet_EX_FM_e',1;'Diet_EX_SBM_e',1;'Diet_EX_BSF_e',
1} ;
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,{'N_Track'},{'min'},options);
```

```
Solution points =7.8976
1.9912 come from diet
5.9064 come from rois
menuChanges = 4×2 table
```

	Food Rxn	Flux
1	'Food_Removed_EX_BSF_e'	-0.3336
2	'Food_Removed_EX_FM_e'	-0.1131
3	'Food_Removed_EX_SBM_e'	-1.0000
4	'Food_Removed_EX_chol_e'	-0.5446

```
Points Simulation Solution:
   N_Track flux = 0.59064
Detailed Analysis:
 Original objective max flux =1.6135 & New objective max flux =1.6135
 N_Track
   Original Diet RoI range = 3.7975:9.8893
   New Diet RoI range = 0.59064:2.0625
```

```
options.OFS=0.5;
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,{'N_Track'},{'min'},options);
```

```
Solution points =4.6459
3.2914 come from diet
1.3545 come from rois
menuChanges = 4×2 table
```

	Food Rxn	Flux
1	'Food_Removed_EX_BSF_e'	-0.8135
2	'Food_Removed_EX_FM_e'	-0.4556
3	'Food_Removed_EX_SBM_e'	-1.0000
4	'Food_Removed_EX_chol_e'	-1.0223

```
Points Simulation Solution:
   N_Track flux = 0.13545
Detailed Analysis:
 Original objective max flux =1.6135 & New objective max flux =0.80674
 N_Track
   Original Diet RoI range = 3.7975:9.8893
   New Diet RoI range = 0.13545:0.82202
```

```
options.OFS=0;
[newDietModel,pointsModel,roiFlux,pointsModelSln,menuChanges,detailedAnalysis
]=nutritionAlgorithm(mainSalmon,{'N_Track'},{'min'},options);
```

```
Solution points =4.5
4.5 come from diet
0 come from rois
menuChanges = 4×2 table
```

	Food Rxn	Flux
1	'Food_Removed_EX_BSF_e'	-1.0000
2	'Food_Removed_EX_FM_e'	-1.0000
3	'Food_Removed_EX_SBM_e'	-1.0000
4	'Food_Removed_EX_chol_e'	-1.5000

```
Points Simulation Solution:
   N_Track flux = 0
Detailed Analysis:
 Original objective max flux =1.6135 & New objective max flux =0
 N_Track
   Original Diet RoI range = 3.7975:9.8893
   New Diet RoI range = 0:0
```

```
clear options
```

As you can see, by setting OFS to zero the new diet provides a solution that corresponds to a maximal biomass flux (i.e, the objective flux) of zero.

Working with More Sophisticated Objective Functions:

The nutrition algorithm was originally developed with simple objective functions in mind (i.e., only one non-zero value in the c vector). To impliment more sophisticated objective functions, on can integrate the objective function into the S matrix with the function "integrate_cVector_into_model". This functon creates a new metabolite and reaction representing the objective function and updates the c-vector such that it is compatible with the nutrition algorithm. To demonstrate the equivalence of the models, let's try simulating a model with and without the modification and evaluate the solutions.

```
newmodel=integrate_cVector_into_model(mainSalmon);
clear options
options.roiWeights=10;
options.display='off';
options.targetedDietRxns={'Diet_EX_FM_e',1;'Diet_EX_SBM_e',1;'Diet_EX_BSF_e',
1};
[~,~,~,salmonModelSln,~,~]=nutritionAlgorithm(mainSalmon,{'N_Track'},
{'min'},options);
[~,~,~,newModelSln,~,~]=nutritionAlgorithm(newmodel,{'N_Track'},
{'min'},options);
abs(salmonModelSln.f-newModelSln.f)<1e-8 %is the solution difference less
than expected numerical error? The answer is yes
```

```
ans = logical
   1
```

As you can see, the difference in points produced between the two solution is less than Matlab's numerical error, indicating that the two solutions are equivalent. Now lets try a more sophisticated objective functon... we may be interested in minimizing respiration rate (reaction 546) while maximizing growth rate to achieve a more conservative metabolism. After simulation, we can see that the resperation rate in our solution is significantly different from before modifying the objective function.

```
testModel=mainSalmon;
testModel.c(546)=-1;
testModel=integrate_cVector_into_model(testModel);
[~,~,~,pointsModelSln,~,~]=nutritionAlgorithm(testModel,{'N_Track'},
{'min'},options);
abs(pointsModelSln.v(546)-newModelSln.v(546))<1e-8 %is the solution
difference less than expected numerical error? The answer is no
```

```
ans = logical
   0
```

Citations:

[1] Weston & Thiele 2022. "A nutrition algorithm to optimize feed and medium composition using genome-scale metabolic models." *Metabolic Engineering.*

[2] Heirendt, Laurent, et al. "Creation and analysis of biochemical constraint-based models using the COBRA Toolbox v. 3.0." *Nature protocols* 14.3 (2019): 639-702.

[3] Zakhartsev, Maksim, et al. "SALARECON connects the Atlantic salmon genome to growth and feed efficiency." *PLOS Computational Biology* 18.6 (2022): e1010194.