Tutorial 2 Whole Body Metabolic Model Personalisation

Authors: Anna Sheehy 08-2024

This tutorial takes you through the optional step of personalizing the WBM models independently of microbiome models or microbiome data. The personalization demonstrated in this tutorial includes incorporation of clinical metadata e.g., age, weight, height, hematocrit, cardiac output, blood metabolomics, urine metabolomics or cerebrospinal fluid metabolomics.

We will also demonstate how to add, silence or remove reactions from the models here. You may want to personalise your WBMs in the following scenarios:

1. You have clinical/metadata for your subjects that you would like to incorporate into the model
2. You want to investigate the effects of changing physiological parameters of the model
3. You want to add or remove reactions to reflect a particular disease state (i.e. gene knockout)
4. You want to add or remove reactions as you have discovered an update in literature that you need to incorporate into the models

Organising your directories

For ease of access we will again define the folders used to store and read results from. If tutorial 1 and 1.5 are used, please use the same inputs used in those tutorials. If you downloaded the pre-made tutorial files set the path for the downloaded files as the resultDirectory variable. We will also define the metadata path again as we will need it later in this tutorial. Note that the explanation and steps are the exact same for the setup as in the other tutorials.

resultDir - The path where all the results of this tutorial will be stored. This ensure all your results will be in one place and are thus easily accesible. Make sure the the folder is accesible to you.

metadataPath - The location of the metadata file. This is used in both the generation of human-microbiome models as well as the analysis of results towards the end of the tutorial.

readsTablePath - The path to the the taxonomic reads file containing the amount of reads per taxonomic assignment. Here we will set to

We set up the variables in the paths variable, which is a structure. Each field in the structure is dedicated to store the relevant variables used in the different steps of Persephone. Normally we would define each variable at the start in the configuration file. However here we will only specifiy the output directories used in Persephone. The other variables we will define as we go along for clarity.

Please copy and paste the paths to the required files in the code below. Remember to add file extensions where relevant.

```
%Define the location of the required files, make sure you dont forget the
file extensions where relevant!
resultPath = '';
paths.General.metadataPath = '';
paths.Mars.readsTablePath = ''
```

Now we will define some paths as to where we want to store all of our results. We do not have to make these directories ourselves, the code will do it for us.

```
paths.seqC.outputPathSeqC = [resultPath, filesep, 'resultSeqC'];
paths.Mars.outputPathMars = [resultPath, filesep, 'resultMars'];
paths.mgPipe.outputPathMgPipe = [resultPath, filesep, 'resultMgPipe'];
paths.persWBM.outputPathPersonalisation = [resultPath, filesep,
'personalisedWBMs'];
paths.mWBM.outputPathMWBM = [resultPath, filesep, 'mWBMmodels'];
paths.fba.outputPathFluxResult = [resultPath, filesep, 'resultFlux'];
paths.fba.outputPathFluxAnalysis = [paths.fba.outputPathFluxResult, filesep,
'fluxAnalysis'];
paths.stats.outputPathStatistics = [resultPath, filesep, 'resultStatistics'];
```

The function initPersephone.m performs the initCobraToolbox.m function, sets the COBRA solver and checks if the required toolboxes are installed (line 1-7). Additionaly it generates the folder structure for the results. IMPORTANT, the output structure used in all 3 tutorials is generated with initPersephone.m. The metadata and the reads table are put through sanity checks. initPersphpne.m ensures that in the metdata the columns with the sample IDs and sample sex have the correct headers and readable data types. It always assumed the first column contains sample IDs. It will create a new updated file and will also update the metadata path accordingly. If you use the test data files you can see that the column sample ID has been changed to ID and the column gender has been altered to Sex. Additionally the data in the column Sex has been changed from M/F to male/female. Important if your own metadata does not have alternative headers accounted for in the function it will raise ahnerror. That is easily fixed by changing your column headers to ID or Sex depending on which data type is causing the issue. Finally we test if the sample IDs in the microbiome data match that of the metadata.

```
% Create the file structure for the results of all the parts of the
% tutorial (not just this tutorial)
[initialised, statToolboxInstalled, updatedMetadataPath] =
initPersephone(resultPath, paths)
```

The following ouputs are returned from initPersephone

initialised - Boolean, indicates if Persephone was succesfully initialised.

statToolboxInstalled - Boolean, indicates if the statistics toolbox is installed. Parts of Persephone are skipped if false

updatedMetdataPath - The path to the updated metadata file.

If you look at the directory given in the resultPath variable, you will see the following folder structure.

-resultDir

-HMmodels

-personalisedWBMs

-resultFlux

-fluxAnalysis

-resultMars

-resultMgPipe

-resultSeqC

-resultStatistics

The content of each of the folders will be discussed in the appropriate sections in the three tutorials.

We have to also update the metadata path to reflect the updated metadata file.

```
paths.General.metadataPath = updatedMetadataPath;
```

Now that we have checked that all our software is installed and available, our results directory has been set up and our metadata and read files has been processed and ready for use we can start with processing the metagenomic reads data through MARS.

Section 1: Personalize WBMs with clinical metadata

Example One: Personalizing a model with Physiological parameters

To personalise the WBM, we will use the function 'personalizeWBM.m'. The function is developed to apply specific changes to the WBM such as updated reaction bounds on blood transport reactions or modified blood volume based on physiological parameters. The function takes the following input

- metadata (REQUIRED): this can be a 2 column cell array with inputs for personalizing one model(see example below), or it can be a path to an excel file with any number of patients' metadata. In either case sex must be provided
- persPhysiology (OPTIONAL): This should be a cell array listing all of the physiological parameters (which must exactly match column headers in your metadata) that you would like to use to personalise your model(s).
- resPath (OPTIONAL): A character array of the path where you would like the personalized models and personalization overview stored
- Diet (OPTIONAL): A diet in the form of a cell array to constrain the models with. By default, EUAverageDietNew will be used from the COBRAtoolbox
- persMetabolites (OPTIONAL): A cell array containing the list of metabolites that you would like to be used to personalise your model(s). This should contain two columns, the first with the vmh ID of the metabolite which can be obtained from vmh.life, and the second with the compartment in which you would like the adjustment to be applied to. This can be ’blood’, ’urine’ or cerebrospinal fluid (’csf’). It is recommended that you choose the compartment based on the sample type e.g., serum metabolomics should be used to update blood metabolite constraints.

The following physiological parameters can be used as to personalise the WBMs:

- **Sex**: sex determines which WBM will be loaded and personalised: Harvey or Harvetta- each model is personalised to the average physiological parameters of a 40 year old male/female. All metabolite bounds are based on sex specific data from the Human Metabolic Database (HMDB) and the female model has ovaries and a uterus while the male model has testis. For more detail see: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7285886/pdf/MSB-16-e8982.pdf
- **Age**: used in the calculation of cardiac output when option 6 or 7 are selected (default is option 1, CO is caluculated based on heart rate and stroke volume)
- **Weight**: used to calculate blood volume and in the case of cardiac output option 0, 4 or 5, it is used to calculate CO
- **Hematocrit**: used in the calculation of renal flow rate, blood flow rate and plasma flow rate.
- **Creatinine**: used to calculate max and min possible urinary secretion rates
- **Heartrate**: used by default to calculate CO and in option 1, 6 and 7
- **Vo2**: used to calculate CO in option 3 and option 5
- **Height**: used to calculate blood volume and used in CO calculation for option 4
- **Stroke volume**: used to calculate CO by default and in option 1, 2, 6 and 7

personalizeWBM will always use the default calculation of CO

Formating physiological input data

The metadata should be a two column cell array with the parameter listed in the first column (with units in brackets) and the value in the second column. Sex must be included

```
% Input metadata to personalise
persPhysiology = {'ID';
                  'Age';
                  'Hematocrit';
                  'Heartrate';
                  'Vo2';
                  'Stroke volume';
                  'Height';
                  'Weight'};
```

Load a diet (optional)

The model will be constrained by a diet before being personalised so that none of the personalisation applied is overwritten by applying the diet afterwards. You do not need to provide a diet for this function to work- it will automatically use EUAverageDietNew. If you wish to use a different diet, you can include it as an input.

```
% Load diet
Diet = 'EUAverageDiet'
% Ensure Diet is valid
Diet = validateDietPath(Diet, resultPath)
```

Setup

Here we show which steps are required to set up your device to support the various functions used to create and analyse human-microbiome models for all three tutorials. First we need to have a copy of COBRA toolbox. COBRA Toolbox download instructions can be found at https://github.com/opencobra/cobratoolbox

To see if the COBRA toolbox is installed correctly we run initCobraToolbox

```
global CBTDIR
if isempty(CBTDIR)
    initCobraToolbox
end
```

As the function gives no errors or warnings we know that the COBRA toolbox is correctly set-up and ready to use.

To decrease simulation times of the models we need an industrial solver. The different solvers supported are

- ibm_cplex
- tomlab_cplex
- gurobi
- mosek

To see MATLAB version and solver version compatibility see https://opencobra.github.io/cobratoolbox/stable/installation.html#solver-installation. Various solvers can be obtained through an free academic license. For modelling purposes here, we recommomend using ibm_cplex if possible.

To set our solver we use changeCobraSolver. If another solver than ibm_cplex is used, replace the 'ibm_cplex' in the the code line below with the solver name you use as found in the bullet list above.

```
solver = 'ibm_cplex';
% solver = 'gurobi';

changeCobraSolver(solver);
```

Example One: Apply physiological changes

```
% Run the function to personalise one model
[iWBM, iWBMcontrol_female, iWBMcontrol_male, persParameters] = persWBM(...
    paths.General.metadataPath, ...
    'persPhysiology', persPhysiology, ...
    'resPath', paths.persWBM.outputPathPersonalisation, 'solver', solver);
```

Check changes

Your newly personalized WBM will be saved to your results directory along with an excel file which gives you an overview of the changes that were applied to the model.

To test that your changes have in fact been applied to the model you can run a number of different analyses to observe the changes in the model. For testing the changes applied by metabolomic inputs you can test the uptake/secretion rate of the metabolite you have changed to any organ to/from the blood, as follows

```
persParameters
```

How we see changes in the model with both some bounds that are changed and FBA results that are changed. I would pick the same comparisons for all 3 examples

Now you have a personalised WBM model and summary of the paramters which the function succesfully personalised. **Talk where these can be found and what that tells us.**

Example 2: Metabolomic changes

Blood, Urine and cerebral spine fluid metabolites can also be personalised if clinical measurements of a patient have been collected from one or more of the three biofluids. The function will than update the transport bounds of the metabolite in that biolfuid to different organs to more closely represent the indivual metabolite level of a patient. It is important that you denote the compartment in which you wish to adjust a metabolite. There are three metabolite pools in which you can adjust the bounds using this function:

[bc] = blood compartment

[u] = urinary compartment

[csf] = cerebrospinal fluid

The metabolites that can be personalised from metabolomics can be found in the following files in the COBRAtoolbox. If your metabolite is not in this file, it cannot be personalised.

- 'NormalBloodConcExtractedHMDB.txt'
- 'NormalCSFConcExtractedHMDB.txt'
- 'NormalUrineConcExtractedHMDB.txt'

The format for the input data can be seen in the supplementary excel file which can be downloaded from: LINK. ***Best practice is to have inputdata be the metadata file. That would mean that all information you might want to use for statistical analysis and personalisation are in the same file which is convenient.***

Formatting metabolomic metadata

The metadata variable has to be formatted in a specfic way. It is important that in the file with these physiological values the same column name as mentioned here is used to ensure the function is able to find the column in the file.

Give a list of the parameters we can change and their corresponding column header. also give the units that can be used

Please take care on how the units are formatted as that will deterime the conversion factors to the units used within to script to personalise the WBMs. The format of the column name for any metabolomic data you might use is important as well. The format is as follows:

VMH ID[compartment] (unit)

For example, if we have blood measurements for D-glucose in mg/dL the correct format for the column header would be:

glc_D[bc] (mg/dL)

To obtain the VMH ID you can use the function 'getVMHID'. The function will return the ID if it finds an exact match, but if it doesn't it will give you a list of suggested metabolites that contain the string provided.

```
% obtain VMH ID
mets = {'Glucose'; 'Fructose'};
suggest = true;
[metID, suggestedMets] = getVMHID(mets, suggest);
```

You'll notice here that no metIDs were found. That is because the in the VMH the metabolites are written as:

'D-glucose' and **'D-Fructose'**

Alternatively, we can use vmh.life to search the compound name and retrieve the VMH ID from there.

Now we can select some metabolite names from the suggested list and get the IDs

```
% Re-run the function using suggested metabolite names
mets = {'D-glucose'; 'D-Fructose'};
suggest = true;
[metID, suggestedMets] = getVMHID(mets, suggest)
```

We now have to incorporate our changes in our metadata file! You can see two different metadatafile attached here. One with 'incorrect' formatting and one with 'correct' formatting.

```
% Define metabolomic parameters
persMetabolites = {'lac_D', 'blood';
                   'na1', 'urine';
                   'cys_L', 'csf'};
Diet = 'EUAverageDiet';
Diet = validateDietPath(Diet, resultPath);
```

```
% Run the function to personalise one model
[iWBM, iWBMcontrol_female, iWBMcontrol_male, persParameters] = persWBM(...
    paths.General.metadataPath, ...
    'persMetabolites', persMetabolites, ...
    'resPath', paths.persWBM.outputPathPersonalisation, 'solver', solver);
```

Example Three: Apply metabolomic and physiological changes to a model

```
% Define physiological parameters
```

```
perPhysiology = {'ID';
                 'Age';
                 'Hematocrit';
                 'Heartrate';
                 'Vo2';
                 'Stroke volume';
                 'Height';
                 'Weight'};

% Define metabolomic parameters
persMetabolites = {'lac_D', 'blood';
                   'na1', 'urine';
                   'cys_L', 'csf'};

% Load your diet
Diet = 'EUAverageDiet'
Diet = validateDietPath(Diet, resultPath);
```

```
% Run the function to personalise one model

[iWBM, iWBMcontrol_female, iWBMcontrol_male, persParameters] = persWBM(...
    paths.General.metadataPath, ...
    'persPhysiology', persPhysiology, ...
    'persMetabolites', persMetabolites, ...
    'resPath', paths.persWBM.outputPathPersonalisation, 'solver', solver);
```

Check changes

```
persParameters
```

Example Five: Creating personalised mWMBs

5.1 - One to one, personalised model to personalised microbiome

This combination can be used when you have personal metadata for each patient and you want each WBM model to be individual to the patient

```
% Set your sovlver
solver = 'ibm_cplex';
% numWorkers
numWorkers = 2;
```

```
% Combine microbiome models with WBMs
createBatchMWBM(paths.mgPipe.outputPathMgPipe, ...
            paths.mWBM.outputPathMWBM, ...
            paths.General.metadataPath,...
            'Diet', 'EUAverageDiet',...
            'solver', solver,...
            'numWorkersCreation', numWorkers);
```

5.2 - One personalised model for multiple microbiomes

This might be used if you have some general clincal information about a group e.g., disease status, mean age, mean glucose levels and you want to make some general adjustements to the male and female models and use one (or two male and female) personalised WBMs to combine with multiple microbiome models.

Set path to personalised iWBMs

```
% Folder where you want your combined mWBMs saved
saveDir = '';
% Path to persoanlised WBM for combining with each microbiome model
wbmDirectory = '';
% Set your sovlver
solver = 'ibm_cplex';
% Choose number of cores to be used
numWorkers = '';
```

Combine microbiome models with WBMs

We will now use the createBatchMWBM.m function to create the mWBM models.

```
% Combine microbiome models with WBMs
createBatchMWBM(paths.mgPipe.outputPathMgPipe, ...
            saveDir, ...
            paths.General.metadataPath,...
            'Diet', 'EUAverageDiet',...
            'solver', solver,...
            'numWorkersCreation', numWorkers, ...
            'wbmDirectory', wbmDirectory);
```

Section 2: Adding or Removing Reactions to/from a Model

This tutorial will take you step by step through the process of adding reactions and removing specfic reactions from metabolic models. We will use the model HM_CSM5MCXD.mat created in tutorial 1. If you did not run tutorial 1, HM_CSM5MCXD.mat can be found in the supplementary materials as well.

General Principles:

Why add or remove/silence reactions?

Adding reactions:

To investigate a particular capability of a model. For example, the examine the maximum production of a particular metabolite in a compartment, you can add a 'demand' or 'sink' reaction for that metabolite in the relevant compartment and then set it as the models objective (maximising or minimising depending of the intrinsic reaction direction and the direction you are interested in investigating). Alternatively you can add a reaction to add new metabolic functionality

Removing or silencing reactions:

To mimic biological conditions or gene knock-out in cases where a patient is missing a protein or gene, for example In-born Errors in Metabolism (IEMs). Single reactions can be blocked/silenced or deleted from the model.

Alternatively, all reactions linked to a specific gene can be silenced- **NOTE reactions linked to mulitple genes...still occur (CHECK DETAILS)** (what does this mean)

Common Applications

Adding demand reaction: usually and used as objective

Adding sink reactions: often added to ensure a specific metabolite is not constraining the limits of the solution with a different objective (**To remove?)**

Adding transport reactions: usually added to ensure that a metabolite moves to all compartments in which it belongs

Adding novel reactions: Usually used to ensure a new capability of the model.

Adding Exchange reactions: Usually used to ensure metabolite can be exchanged with the environment to satisfy steady-state constraints

Silencing a single reaction: might be used to examine if a reaction is contributing to the flux of the objective (you can also analyse reduced cost) (**This last statement reads like it requires more indepth explanation?)**

Silencing multiple reactions linked to one gene (aka gene knockout): may be used when examining metabolism in individuals who have IEMs or to investigate disease states which feature protein malfunction etc.

Removing a reaction that is incorrect: Done in order to remove infeasibilities from the model or to make the model more accurate. NOTE: it is good practice to thouroughly test the model by silencing the reaction first before actually removing it.

Example One: Add a demand reaction

Demand reactions are formulated as exchange reactions, but operate from within the model instead of on the compartments that interact with the environment as normal exchange reactions do. The demand reaction is formulated as

trp_L[bc] -> Ø.

Demand reactions can give you interesting insights into the metabolic capabilities of your models and how they differ between your samples whether that is down to microbiome composition or personalisation. Demand reactions can be added to compartments of the model which also allows for a more targeted investigation whereby the user can solely examine e.g., blood, uring or cerebrospinal fluid. Note that this example is the same as example 5 in Tutorial 2: Solve and analyse WBMs

Lets add a demand reaction for L-tryptophan in the blood. Find the VMH id of tryptophan on vmh.life

```
% Define metabolite of interest
metabolite = 'trp_L[bc]';
```

Not all metabolites in the model are present in the blood compartment. We can check if a metabolite ID is present in a compartment by running surfNet as we done in the previous examples

```
% Load a model to investigate
modelHM = load(fullfile(paths.mWBM.outputPathMWBM,
'mWBM_CSM5MCXD_male.mat'));

% Check if metabolite exists in the model
surfNet(modelHM, metabolite)
```

We see the metabolite number is 1189 and has many producing reactions. This indicates the metabolite exists and is connected to the rest of the model.

To investigate the demand of L-tryptophan in the blood, we need to add a demand reaction. We add on a demand reaction with the function addDemandReaction which takes the following inputs.

- model - a metabolic model, in our case a WBM or mWBM
- metaboliteNameList - a cell array of metabolites that a demand reactions has to be created for. Usually we only add one demand reaction.

Now we add the demand reaction on to the model. We now rename the updated model to modelHMdemand which we have not done before when we made changes. This is because adding a demand reactions changes the model and its behavious, it is good practice to have the original model still in case we want to make other changes. It is also good practice to only have one user added demand reaction active at a time. We could silence each newly made demand reaction after we optimised it, but by using the original model again for a new demand reaction addition we are sure we do not carry over the changes from the previous addition.

```
% Add a demand reaction for the metabolite of interest
modelHMdemand = addDemandReaction(modelHM,metabolite);
```

We can see that the demand reaction has been added by using surfNet. A new demand reaction will always have the reaction ID DM_metabolite ID[comparment]. So for our case it would be DM_trp_L[bc]

```
surfNet(modelHMdemand, 'DM_trp_L[bc]')
```

As you can see we have succesfully added on the demand reaction. The only issue is that the upper bound is set to 1000, which is a normal unconstrained value to use for smaller models. However in WBMs we use the value 1000000, to ensure we do not hit the lower unconstrained bound set by the function we will manually change the upper bound.

```
% Unconstrain the demand reaction's upper bound for analysis
modelHMdemand = changeRxnBounds(modelHMdemand,['DM_' metabolite],100000,'u');
% Check the bounds again for the reaction
surfNet(modelHMdemand, 'DM_trp_L[bc]')
```

Now the demand reaction has been added on to the model, it can be used as a modelling option. For more information on solving demand reactions please look at tutorial 2.

Example two: Add a transport reaction

When might you want to add an exchange reaction? Perhaps you have found in literature that a particular metabolite is found in an organ that it is not currently taken up by in your model, e.g., you discover that vitamin D3 is taken up by Natural Killer cells and you want your model to reflect that (**This is an example and does not necessarily reflect true metabolic capacities**). We will use the function addReactionsHH that takes the following inputs:

- model - A WBM that has to be altered, in our case modelHM we defined earlier.
- rxnAbbrs - List of abbreviations for the new reaction(s)
- rxnNames - List of full descriptive names for the new reaction(s)
- reactions - List of the reaction formula(s). The required format is {'A -> B + 2 C'} or {'A <=> B + 2 C'} for one way or reversible reactions, respectively.
- gprs - List of gene-protein-reaction rules. Use booleans and / or if multiple genes are linked to the reaction.
- subSystems - List of subsystem(s) for each reaction added.
- couplingFactor - Coupling factor to ensure proper intergration with the WBM, defaults to 20000. It is advised not to change this value.

We will now assign values to the inputs. Good practice for creating reaction abbreviations/IDs for transport reactions is to take the organ abbreviation (if applicable), in this case Nkcells_, the VMH ID of the metabolite that is transported in all capitals, VITD3, and than lower case t to indicate transport and either and i or an r to indicate irreversible or reversible, repsectively. Lastly we indicate the compartment on the left side of the reaction where the compound and the the compartment on the right side of the reaction with a 2 inbetween to indicate which compartments are involved in the transport and which compartment is on which side of the reaction, in this case e2c

General reaction abbreviation: Organ Abrreviation + VMH metabolite ID full caps + t + i/r + e2c

Nkcells_VITD3tre2c

In case of blood transport reactions the formula for transport reactions is the organ abbreviation that is involved in transport, Nkcells_, followed by EX_, the VMH metabolite ID as found on the VMH, no need for full caps here and finally (e)_[bc].

General reactiona abbreviation blood transport: Organ Abbreviation + EX_ + VMH metabolite ID (normal capitalisation as found on vmh.life) + (e)_[bc]

Nkcells_EX_vitd3(e)_[bc]

Whenever you are in doubt for proper abbriviation structure look in the WBMs for similar reactions and follow those naming conventions. Because there is no transport for vitamin D3 from the blood to the natural killer cell extracellular space or from the natural killer cell extracellular space to the cytosol we will need to add both transport reactions in order to get vitamin D3 from the blood to the cytosol of the natural killer cells. If you find that the metabolite is metabolised in a different compartment within the cell, you need to add on an additional transport reaction that will go from the cytosol to the desired compartment.

Naming transport reactions is generally quite easy, but it is important to keep them descriptive. Good practive is to say what is transported, which organs is involved and which compartments.

General reaction name: Transport of + full name of metabolite + organ + compartments

Transport of vitamin D3, Nkcells, blood to [e]

Here we will now define the reaction abbreviation and names for both reactions.

```
rxnAbbrs = {'Nkcells_EX_vitd3(e)_[bc]'
    'Nkcells_VITD3tre2c'};

rxnNames = {'Transport of vitamin D3, Nkcells, blood to [e]'
    'Transport of vitamin D3, Nkcells, [e] to [c]'}
```

It is important to double check that the reaction abbreviations/IDs do not already exist in the model! We check this by running findRxnIDs which takes a metabolic model and a list of reaction abbreviations/IDs that have to be searched

```
findRxnIDs(modelHM, rxnAbbrs)
```

We can see that the output we get is a vector of two 0s. This means no index was found for these reaction abbreviations/IDs and we can safely use them.

Now we will define the reaction formula. You need to know the VMH IDs of your metabolites, the organs (if applicable) and compartments the reaction takes place in. We you -> to indicate irriversible reactions. Normally <- is not used as it is better practice to switch the right side and left side so the reaction is read correctly from left to right. For reversible reactions we use <=>. To indicate more metabolites we use + between metabolite IDs. If another stoichiometry than 1 is used for participants of the reaction indicate it with the appropriate number in front of the metabolite. It is important to add spaces between arrows, + signs and stoichiometries as the function will otherwise not recognize the formula.

General example: met1[compartment] + 2 met2[compartment] -> met3[compartment]

NOTE! This is not desired: met3[compartment] <- met1[compartment] + 2 met2[compartment]. The general example should thus be used

We will define our two formulas here. For transport reactions be aware that sometimes protons or other metabolites might be synport or antiport and you should consult the literature for the correct metabolites to use.

```
reactions = {'vitd3[bc] <=> Nkcells_vitd3[e]'
    'Nkcells_vitd3[e] <=> Nkcells_vitd3[c]'};
```

Finally we have to indicate the gene-protein-reaction rules (gprs) and the subsystem associated with the reaction. Usually for generic transporters it is difficult to know which genes are responsible for the enzymes transporting the metabolites, if there are any. In this example we leave the gprs as an empty string (") as we do

not have information on the gprs. To find out which subsytems our reactions best belong to, we can look at the current existing subsystem in the model.

```
% Obtain all the subsystems in the model
subystemsModel = modelHM.subSystems;

% Convert the datatypes and get all unique values
uniqueSubsystemsModel =
unique(cellfun(@cell2mat,subystemsModel,'UniformOutput',false))
```

We can see here a long list of available subystems. The transport from the blood compartments is usually set as Transport, biofluid. Transport from the [e] to [c] is usually set to Transport, extracellular. We do not enter the coupling factors as we take the default.

```
% Set the gprs
gprs = {''
    ''};

% Set the subsystems
subSystems = {'Transport, biofluid'
    'Transport, extracellular'};
```

Before we add the new reactions to the model, we need to introduce the novel metabolites first as these do not yet exist yet in the model. We can do this with the addMultipleMetabolites.m function which takes

- model: The model to add the Metabolite batch to. In our case modelHM
- metIDs: The metabolite IDs to add. No duplicate IDs may be provided and no ID may be present in the supplied model.

The two novel metabolites are vitamin D3 in the [e] and [c] compartments of the natural killer cells. We use the metabolite IDs we generated for the reaction formula. Good practice is to save the output model in a different variable name so you will also have the original model available.

```
% Set the metabolite IDs and add them to the model
metIDs = {'Nkcells_vitd3[e]'
    'Nkcells_vitd3[c]'};
modelHMets = addMultipleMetabolites(modelHM, metIDs);
```

Now we call the function to add the reactions to de HM WBM. Good practice is to save the output model in a different variable name so you will also have the original model available.

```
modelHMAddTransport = addReactionsHH(modelHMets, rxnAbbrs, rxnNames,
reactions, gprs, subSystems);
```

Now we check if the reactions are in the model as we expect

```
printRxnFormula(modelHMAddTransport, 'rxnAbbrList', rxnAbbrs)
printFluxBounds(modelHMAddTransport, rxnAbbrs, true)
```

Now we see that the our new reactions have indeed been incorporated into the model.

When we solve the models, these two reactions will not be able to carry flux because vitamin D3 is not metabolised anywhere in the natural killer cells. If we want to test if the reactions are correctly connected to the system we can either introduce a demand reaction (example 1) for vitamin D3 in the cytosol of natural killer cells or incorporate the vitamin D3 to the rest of the metabolism of the natural killer cell and the WBM. Here we will test the transport reactions with by introducing a demand reaction.

```
% Introduce a demand reaction in the model
modelHMAddTransportDemand = addDemandReaction(modelHMAddTransport,
'Nkcells_vitd3[c]');

% Set the objective function
modelHMAddTransportDemand = changeObjective(modelHMAddTransportDemand,
'DM_Nkcells_vitd3[c]');

% Solve model and print solution
solution = optimizeWBModel(modelHMAddTransportDemand);
solution.f
```

We see that the maximum uptake of vitamin D3 uptake in the cytosol of natural killer cells is 63.6207 mmol/day. This indicates that both transport reactions are able to carry flux as otherwise vitamin D3 would not be able to reach the cytosol of natural killer cells. For more indepth explanation in the use of optimizeWBModel and solving models please look at tutorial 2.

Example 3: Adding a metabolic reaction

Keeping with the example of vitamin D3 in natural killer cells, we now found out that vitamin D3 is not only transported but also metabolised in natural killer cells, in this case into 25-hydroxy vitamin D3. You can find appropriate gprs from the same reactions in different organs or from literature. A comprehenive method of finding genes, and in which compartments they are work can be found in [2]. We base our reaction off the pre-existing 25-hydroxylase in the VMH: RE1303C. It is important to check if your reaction is mass and charge balanced before you add it.

We will use the same function addReactionsHH.m here. There are no set rules to create reaction abbreviations/IDs for metabolic transformation reactions but an easy rule is the organ + the vmh ID (in capitals) + a shortened version to indicate the type of reaction. Here we would get Nkcells_VITD3HYD, the HYD stemming from hydroxylase. The same goes for the reaction name.

```
% Set the reaction abbreviation/ID and name
rxnAbbrs = {'Nkcells_VIT3HYD'};
rxnNames = {'25-hydroxylation of vitamin D3, Nkcells'};
```

Now we will define the reaction formula. As we copy an existing reaction from the VMH we can use those metabolites in the correct organ. If the reaction is not a copy from the VMH you have to consult literature and if the enzyme is known, consults other reactions in the VMH performed by the enzyme to come to a proper reaction formula.

```
% Set reaction formula
reactions = {['Nkcells_o2[c] + Nkcells_h[c] + Nkcells_nadph[c] +
Nkcells_vitd3[c] ...' ...
    '-> Nkcells_h2o[c] + Nkcells_25hvitd3[c] + Nkcells_nadp[c]']};
```

We do not know the gprs here so again we leave them empty. In case you do know the gprs you should use the Entrez Gene ID as used in the VMH to indicate genes. In the subystems in example 2 we can see that there is a subsystem called Vitamin D metabolism and is what we will use here.

```
% Set the gprs
gprs = {''};

% Set the subsystems
subSystems = {'Vitamin D metabolism'};
```

Again we need to add in the novel 25-hydroxy vitamin D3 metabolite to the model. We use the metabolite IDs we generated for the reaction formula. Good practice is to save the output model in a different variable name so you will also have the original model available.

```
% Set the metabolite IDs and add them to the model
metIDs = {'Nkcells_25hvitd3[c]'};
modelHMVitDMets = addMultipleMetabolites(modelHM, metIDs);
```

Now we can add the reaction to the model that already has the transporters in them!

```
modelHMVitD = addReactionsHH(modelHMVitDMets, rxnAbbrs, rxnNames, reactions,
gprs, subSystems);
```

Now we check if the reactions are in the model as we expect

```
printRxnFormula(modelHMVitD, 'rxnAbbrList', rxnAbbrs)
printFluxBounds(modelHMVitD, rxnAbbrs, true)
```

Again we will not get flux through the new reaction as the produced metabolite cannot go anywhere. Here we will again introduce a demand reaction to allow flux through the reaction.

```
% Introduce a demand reaction in the model
modelHMVitDemand = addDemandReaction(modelHMVitD, 'Nkcells_25hvitd3[c]');

% Set the objective function
```

```
modelHMVitDemand = changeObjective(modelHMVitDemand, 'Nkcells_VIT3HYD');

% Solve model and print solution
solution = optimizeWBModel(modelHMVitDemand);
solution.f
```

We see that the maximum uptake of 25-hydroxy vitamin D3 uptake in the cytosol of natural killer cells is 63.6207 mmol/day. This indicates that also the novel added reaction is able to carry flux. For more indepth explanation in the use of optimizeWBModel and solving models please look at tutorial 2.

Example 4: Adding a excretion reaction

A excretion reaction is added to the model if there is a new compound created or an existing compound that has to be excreted. These can only be done in the urine, sweat, air, or feces compartments. Important is that the metabolite must first reach the desired compartment via transport reactions and example 1 can be used as template to achieve this. In the modelHM now, Thymidine 5'-triphosphate cannot be excreted in urine. Let's say we found information that says it can. We need to add transport reactions from the blood to urine and then add the exchange reaction. We already showed how to add transport reactions, here we will only show how to add an exchange reaction. The function we will use is addExchangeRxn.m and takes

- model: Cobra model structure, in our case modelHM
- metList: List of metabolites, in our case Thymidine 5'-triphosphate
- lb: Array of lower bounds, as we excrete from the body this will be 0
- ub: Array of upper bounds, as we excrete from the body this will be 1000000

Let us set the variables

```
% Set variables
metList = {'dttp[u]'};
lb = 0;
ub = 1000000;
```

Now we will add the exchange reaction. Note that we save the updated model in a new variable as it is good practice to keep the original model.

```
% Add exchange reaction
modelHMExcretion = addExchangeRxn(modelHM, metList, lb, ub);
```

We get the warning that the metabolite has been added to the model as it was not present before. This is what we expected. We would have to add the metabolite also in the urinary compartment with the same function we used in example 2 and 3. However for ease we have not shown that here. Now we will check if our changes are done as we expect

```
% Reaction ID will always follow EX_metabolite ID
rxnID = {'EX_dttp[u]'};
% Print the reaction formula and the flux bounds
printRxnFormula(modelHMExcretion, 'rxnAbbrList', rxnID)
```

```
printFluxBounds(modelHMExcretion, rxnID, true)
```

Now we can indeed see that the bounds and the reaction exist. We can test if the exchange reaction can carry flux by linking it to the rest of the model with transport reactions as shown in example 2.

Example 5: silencing a reaction

In case you want to know if a reaction contributes to metabolism or if you want to know what the effect is if a particular reaction is not present we can silence a reaction. Silencing a reaction is putting both bounds on 0, forcing no flux through the reaction. We can easily do this with the changeRxnBounds function which takes the following inputs:

- model - A metabolic model, in our case a WBM or mWBM
- rxnNameList - A cell array of reactions that have to have their bounds changed.A cell array of reaction IDs or a string for a single reaction ID that species which reaction(s) need their bounds altered.
- value - A numeric value that say to which value the bound has to be changed to. Can be one value for all reactions, or one value for each reaction to be changed. In our case the value 0.
- boundType - A string, indicating which bound has to be changed, u= upper, l= lower, b = both. In our case 'b' as we want both bounds to have the same value.

Let us use here the conversion of vitamin D3 to 25-hydroxy vitamin D3 in the liver. First we need to know the reaction name, which we can find in the VMH, 'RE1303C'.

```
% Set the reaction ID that has to be silenced
rxnSilence = 'Liver_RE1303C';

% The the value of the bounds
value = 0;

% Indicate which bounds should be altered
boundType = 'b';

% Silence the reaction
modelHMSilenced = changeRxnBounds(modelHM, rxnSilence, value, boundType);
```

We can now look at the if our changes worked.

```
% The original model
printFluxBounds(modelHM, {'Liver_RE1303C'}, true)
```

```
% The silenced model
printFluxBounds(modelHMSilenced, {'Liver_RE1303C'}, true)
```

As we can see, the reaction has been succesfully silenced. To understand how to use the altered model for optimisation purposes please look at tutorial 2.

Example 6: silencing a gene

We can also silence all reactions associated with a gene to simulate a gene knock-out. We first need to have our gene of interest, let us take hexokinase 1. We first need to find the Entrez gene ID in the VMH, 3098.3. Now we can find all associated reactions with the function findRxnsFromGenes which takes.

- model: COBRA model structure, in our case modelHM
- genes: string of single gene or cell array of multiple genes for which `rxns` are desired.
- numericFlag: 1 if using Human Recon (Default = 0)
- ListResultsFlag: 1 if you want to output `ListResults` (Default = 0)

The most important thing to mention is that gpr rules cannot be found in the HM WMBs because they were removed to save space in the models. We have to thus load either Harvey or Harvetta, based on the sex of our sample and then find the associated reactions. We can then silence the reactions found in the HM WBM. Here we will demonstrate with Harvetta as HM_CSM5MCXD.m is from a female sample.

```
% Load the base-WBM
baseWBM = loadPSCMfile('Harvetta');

% Set the gene we want to find reactions for
geneList = {'3098.3'};

% Indicate if we are using recon, this 0 as use WBMs
numericFlag = 0;

% We want the ListResults as it makes processing easier
listResultsFlag = 1;
```

To find to reactions we will only look at the second output of the function as that is easiest for us to work with.

```
% Find the reactions
[~, listResults] = findRxnsFromGenes(baseWBM, geneList, numericFlag,
listResultsFlag);
```

The listResults give the following information:

- Column 1: Reaction IDs
- Column 2: Reaction formula
- Column 3: The subsystem associated with the reaction
- Column 4: Reaction Name
- Column 5: The GPR rules
- Column 6: The original gene used to find the reactions with

Now we can easily extract the reactions associated with the gene we want to silence by only taking the first column of the listResults variable

```
% Obtain the reaction IDs
```

```
rxnsToSilence = listResults(:,1);
```

Now we set set the constraints of all reactions we found to 0 for both upper and lower bounds in the HM WBM

```
% The the value of the bounds
value = 0;

% Indicate which bounds should be altered
boundType = 'b';

% Silence the reaction
modelHMSilenced = changeRxnBounds(modelHM, rxnsToSilence, value, boundType);
```

Now we will take the first reaction of the rxnsToSilence variable to check if the changes were succesful.

```
% The original model
printFluxBounds(modelHM, listResults(1), true)

% The silenced model
printFluxBounds(modelHMSilenced, listResults(1), true)
```

And indeed we see the reaction has been succesfully silenced.

Example 7: removing a reaction

In special circumstances we can remove a reaction, e.g., when it interferes with the model function, if it is proven to be incorrect or if if does not connect with the rest of metabolism. Good practice, however, is to silence to reaction and througouhly test that the removed reactions does not interfere with other results and does not cause any other problems. To remove a reaction we can use removeRxns.m which takes:

- model: COBRA model structure, here modelHM
- rxnRemoveList: Cell array of reaction abbreviations to be removed

Let us try and remove the excretion reaction of D-glucose in the urine

```
% Set the reaction ID we want to remove
rxnToRemove = {'EX_glc_D[u]'};
% Remove the reaction
modelRemoved = removeRxns(modelHM, rxnToRemove);
```

Now we will see if the removal worked

```
% For the original model
findRxnIDs(modelHM, rxnToRemove)

% For the model with a removed reaction
findRxnIDs(modelRemoved, rxnToRemove)
```

And indeed we see that the reaction cannot be found anymore in the model and it has been succesfully removed.

Example Two: Add a sink reaction

Now lets define our reactions and run addReactionsHH. When adding a demand reaction for ATP, or in fact any reaction to the model there are a few formatting rules that need to be followed for consistnecy in the models.

The format of reaction name should be as follows:

"Organ_TYPE_metabolite_compartment abbreviation"

The reactions formula should then indicate the direction of the substrates and products and each substrate and product must start with organ and also denote the compartment:

"**Organ_met[compartment] <-> Organ_met[compartment]" This is a transport reaction?**

To find the abbreviations for the metabolites you can visit:

- Virtual Metabolic Human website: https://www.vmh.life/

The organs found in Harvey and Harvetta are:

add

The compartments in the model are:

add

To find out which metabolites are present in which organs and compartments are present in the model you can explore the model using:

- surfNet, a function that allows you to explore the model. Or simply use the 'find' and 'contains' funciton in matlab to search the model

As an example we will add a sink reaction for pyridoxal in the small intestinal epithelium

```
% Add reactions, defining all relevant fields
modelNsink = addReactionsHH(modelHM, {'sIEC_sink_pydx[c]'}, {'Sink for
pyridoxal, sIEC'},...
                 {'glc_D[c]->'}, {''}, {'Exchange/demand reaction'})
```

Solve new model to ensure new reaction is present and model is still feasible

```
% Check for reaction and bounds on reaction
rxn = 'sIEC_sink_pydx[c]'
idxN = find(contains(modelNsink.rxnNames, 'Sink for pyridoxal, sIEC'));
idxR = find(contains(modelNsink.rxns, rxn));
```

Print the reaction formula and bounds, rxnbounds can be printed with printRxnBounds function

```
% check reaction formula
wbmReaction = printRxnFormula(modelNsink, rxn)

lb = modelNsink.lb(idxN);
ub = modelNsink.ub(idxN);
```

Solve model using any objective to check feasibility

```
% Set objective
modelAlt = changeObjective(modelNsink, 'Whole_body_objective_rxn');
% Set model sense
modelAlt.osenseStr = 'max';
% Solve model
FBA1 = optimizeWBModel(modelAlt);
FBA1.f
```

Solve for you new reaction to gain insights into the model in relation to your metabolite of interest

```
% Set objective
modelAlt = changeObjective(modelNsink, rxn);
% Set model sense
modelAlt.osenseStr = 'max';
% Solve model
FBA2 = optimizeWBModel(modelAlt);
FBA2.f
```