## Parameter database

The model structure and parameters values can be obtained from an sqlite3 database.

First, one must create a database schema, populate its tables with parameters, then load the database for simulations or parameter fitting. One may also load specific parameter values into a model database file.

These operations are easily performed with the scripts explained below.

### Creating a database schema (database file)

To generate a database file with the correct model schema, run the script: "/scripts/db/makeSchema.py" [here](https://github.com/danielriosgarza/hungerGamesModel/blob/main/scripts/db/makeSchema.py). First specify the name of the database file in the top of the script. By default, the database file will be stored in: ".../files/dbs/".

For example, creating a database to simulate grow of the three species (Bh, Bt, and Ri):

* open the script (".../scripts/db/makeSchema.py") in a text editor add the desired database name:

import os  
from pathlib import Path  
import sqlite3  
  
#Replace this name for the desired name.  
databaseName = 'modelDB\_bhbtri.sqlite3'

* Run the script.
* Check that "modelDB\_bhbtri.sqlite3" file was created in the folder ".../files/dbs/"

To edit and manipulate the database, it is useful to use a graphical interface database browser. I recommend using [DB Browser for SQlite](https://sqlitebrowser.org/)

Here is how it looks:

![fig:](data:text/html; charset=utf-8;base64,)

### Populating the database

Once the schema is created, one can load it into the browser and fill the parameters manually. Alternatively, run the script (".../scripts/db/inserParameters.py") [here](https://github.com/danielriosgarza/hungerGamesModel/blob/main/scripts/db/insertParameters.py). This script populates the database with default parameters tables (located in ".../files/strainSummaries/dbsTemplateTables").

Set the "databaseName" variable to the desired database file.

To modify, either edit the corresponding "tsv" table or edit tabs of the excel file "dbTemplate.xlsx" and save them as a ".tsv" file with their corresponding table name. The database tables are:

#### elements

[template example](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/strainSummaries/dbsTemplateTables/elements.tsv)

stores information about chemical elements. The fields are:

* id (atomic symbol)
* name
* MolecularWeight

#### metabolites

[template example](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/strainSummaries/dbsTemplateTables/metabolites.tsv)

* id
* color (used in the kinetic plots)
* MolecularWeight

#### metabolite2elements

[template example](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/strainSummaries/dbsTemplateTables/metabolites2elements.tsv)

stores the relation between metabolites and chemical elements

* id (unique integer)
* metabolite
* element
* atoms

#### wc

[template example](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/strainSummaries/dbsTemplateTables/wc.tsv)

stores the concentration of components in the media

* metabolite
* concentration (in mM)

#### species

[template example](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/strainSummaries/dbsTemplateTables/species.tsv)

stores metadata for each species

* id (bt, bh, ri)
* name
* genomeSize
* geneNumber
* patricID
* ncbiID

#### feedingTerms

[template example](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/strainSummaries/dbsTemplateTables/feedingTerms.tsv)

stores the feeding terms for each species

* id
* name
* species

#### feedingTerms2metabolites

[template example](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/strainSummaries/dbsTemplateTables/feedingTerms2metabolites.tsv)

* id
* feedingTerm (the feeding term id)
* metabolite
* yield (positive for consumed, negative for produced metabolites)
* monodK (Monod's constant, zero for produced metabolites)

#### subpopulations

[template example](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/strainSummaries/dbsTemplateTables/subpopulations.tsv)

stores information about the intraspecific subpopulations

* id
* species
* mumax (maximum growth rate)
* pHoptimal
* pHalpha
* count
* color
* state (active, inactive, dead)

#### subpopulations2subpopulations

[template example](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/strainSummaries/dbsTemplateTables/subpopulations2subpopulations.tsv)

stores the transitions between intra-species subpopulations

* id
* subpopulation\_A
* subpopulation\_B
* hillFunc
* rate

#### subpopulations2feedingTerms

[template example](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/strainSummaries/dbsTemplateTables/subpopulations2feedingTerms.tsv)

stores the relation between subpopulations and feeding terms

* id
* subpopulation
* feedingTerm

## Loading parameters from a database and performing your own simulations

* First load the necessary packages

from pathlib import Path  
import os  
import sys  
  
#assuming the current path id /scripts/core import the main model classes  
from mainClasses import \*  
  
#add the database scripts to the sys.path and import them  
sys.path.append(os.path.join(Path(os.getcwd()).parents[0], 'db'))  
from readModelDB import \*  
  
#import plotly to make plots that are rendered in the browser  
import plotly.io as pio  
pio.renderers.default='browser'

* load the parameter database and file with pH and acid concentration to be used as an input to the linear model that predicts the pH

#pH file  
ipH\_path = os.path.join(Path(os.getcwd()).parents[1], 'files', 'strainSummaries', 'bhbtri\_ipH4.tsv')   
#databse file with parameters  
databaseName = 'modelDB\_bhbtri.sqlite3'  
databaseFolder = os.path.join(Path(os.getcwd()).parents[1], 'files', 'dbs')  
  
#Load the database  
db = get\_database(os.path.join(databaseFolder, databaseName))

* get the starting pH from the default WC metabolome

#getStarting pH  
wc = createMetabolome(db, 'wc')  
predictpH = getpH(wc.metabolites, ipH\_path) #this function will be used to predict the pH  
pH = predictpH(wc.get\_concentration()) #is the starting pH

* load the metabolomes and microbiome that will be used in the reactor and in the feed. Since we are not simulating migration, make them steril

wc\_f = createMetabolome(db, 'wc', pH, pHFunc=predictpH)  
wc\_r = createMetabolome(db, 'wc', pH, pHFunc=predictpH)  
  
#get the feed obj. Make it sterile  
bhbtri\_f = Microbiome({'bh':createBacteria(db, 'bh', 'wc'), 'bt':createBacteria(db, 'bt', 'wc'), 'ri':createBacteria(db, 'ri', 'wc')})  
bhbtri\_f.subpopD['xa'].count = 0  
bhbtri\_f.subpopD['xe'].count = 0  
bhbtri\_f.subpopD['xi'].count = 0  
  
#create the reactor obj, with starting populations  
bhbtri\_r = Microbiome({'bh':createBacteria(db, 'bh', 'wc'), 'bt':createBacteria(db, 'bt', 'wc'), 'ri':createBacteria(db, 'ri', 'wc')})

* create the culture regime (here, batch with 120 h).
* The regime is implemented by "pulses". The pulse class requires the following inputs:
  + metabolome
  + microbiome
  + t\_start
  + t\_end
  + n\_steps
  + vin (initial inflow)
  + vout (initial outflow)
  + qin (continuous inflow, volume in mL added per hour)
  + qout (continuous outflow, volume in mL removed per hour).

Different regimes and perturbations can be created by a list of "Pulse" objects.

batchA = Pulse(wc\_f, bhbtri\_f, 0, 120, 10000, 0, 0, 0,0)

* run a simulation and plot the results

r\_bhbtri = Reactor(bhbtri\_r, wc\_r,[batchA], 60)  
r\_bhbtri.simulate()  
r\_bhbtri.makePlots()

## Loading specific model parameters into a desired database file

It is not necessary to create a new database or to change the template file to use different parameter settings. This can be done automatically from a parameter file.

For example,

* consider the model parameters described [here](https://github.com/danielriosgarza/hungerGamesModel/wiki/State-equations)
* with "tsv" files structured as:
  + [bh](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/params/bh.tsv)
  + [bt](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/params/bt.tsv)
  + [ri](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/params/ri.tsv)
* we can load parameters into a database of choice and simulate the model.

from pathlib import Path  
import os  
import sys  
  
sys.path.append(os.path.join(Path(os.getcwd()).parents[0], 'db'))  
  
from readModelDB import \*  
from mainClasses import \*  
from loadParameters import \*  
  
import plotly.io as pio  
pio.renderers.default='browser'  
  
ipH\_path = os.path.join(Path(os.getcwd()).parents[1], 'files', 'strainSummaries', 'bhbtri\_ipH4.tsv')   
  
databaseName = 'modelDB\_bhbtri.sqlite3'  
  
databaseFolder = os.path.join(Path(os.getcwd()).parents[1], 'files', 'dbs')  
  
#update database with parameters from a file  
##########################################  
  
#create a database connection  
conn = create\_connection(os.path.join(databaseFolder, databaseName))  
  
#load the parameter file (parameter files are located at "/files/params" )  
bh\_params = getPramsFromFile('bh', os.path.join(Path(os.getcwd()).parents[1], 'files', 'params', 'bh.tsv'))  
  
bt\_params = getPramsFromFile('bt', os.path.join(Path(os.getcwd()).parents[1], 'files', 'params', 'bt.tsv'))  
  
ri\_params = getPramsFromFile('ri', os.path.join(Path(os.getcwd()).parents[1], 'files', 'params', 'ri.tsv'))  
  
  
#assign these parameters (depending on the strain, use the specific function)  
assignBhParams(bh\_params, conn)  
  
assignBtParams(bt\_params, conn)  
  
assignRiParams(ri\_params, conn)  
  
#Load database  
db = get\_database(os.path.join(databaseFolder, databaseName))

This script loaded the parameter file and altered the database values. Now the simulation may proceed as described above with the new parameter values. This quick parameter assigment allows us to fit the parameters to experimental data. ([see](https://github.com/danielriosgarza/hungerGamesModel/wiki/Fitting-models-to-experimental-data))

## Exploring model structure with simple Apps

## Jupyter notebooks