Hands-on 2: Environmental evaluation of feeding strategies with agent-based modeling and life cycle assessment: from theory to practice

Florence Garcia-Launay

florence.garcia-launay@inrae.fr

PEGASE, INRAE, Institut-Agro, Saint-Gilles, France

2024, July, 21st – ASAS-NANP Symposia, Calgary, Canada

General statement and License for further use

This tutorial and the associated files have been developed for Hands-on 2 at the 2024 ASAS Pre-Conference on Modeling in Animal Nutrition, Calgary, 2024, July 21st. It is based on the principles of the Mogador model, that is registered at European organization for the protection of authors and publishers of digital creations (see QR code), and has been developed in a common project between INRAE UMR PEGASE and Ifip.

- https://github.com/National-Animal-Nutrition-Program/2024ASAS_Garcia-Launay

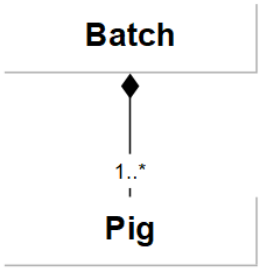
- https://forgemia.inra.fr/florence.garcia-launay/Hands-on_ASAS2024/-/tree/main

1. Step 0. Individual-based model of a pig-fattening batch, without nutritional consideration

In this step 0, the objective is to learn how it is possible to produce a model in Python code, while using both individual-based and discrete-events paradigm. The model developed in Step 0 simulates the growth of each individual in a batch of pigs (n=400) and simulates the delivery of the pigs to the slaughterhouse.

Classes_Step0.py

To implement an agent-based or individual-based model, it is easy to use object-oriented programming, available in Python. Therefore, this is possible to produce a simple class diagram for this first model (Figure 1).




```
class Batch:

    def __init__(self, nb_pigs, ID=''):

        self.dict_pigs = {} # dictionnary of pigs objects in the batch
        self.nbpigs = nb_pigs # number of pigs in the batch

        for i in range(self.nbpigs): # to create pigs objects in the batch
            self.dict_pigs[temp] = Pig(ADG_pot, Agei, BWi, BGompertz, temp)
                #arguments necessary to create pig objects


class Pig:

    def __init__(self, ADG_pot, Agei, BWi, BGompertz, ID=''):
        ...
        ...
```

Figure 1. Simple class diagram used in Step 0 to initiate an individual-based model of a batch of pigs, and extraction of Python code (from Classes_Step0.py) which shows how creating these classes and their link.

When creating pig objects, it is necessary to generate their own parameter values to consider the variability in growth potential among them. For this purpose in Step 0, for each pig, we draw at random (normal law) values for growth potential (ADG_pot), initial age (Agei) and initial body weight (BWi), as well as for a Gompertz factor (BGompertz) that will make it possible to calculate each day the growth rate according to a Gompertz equation.

The possibilities of the object-oriented programing are used to apply a simple growth function that calculates each day the body gain according to the current body weight and the growth potential. The batch level makes it possible to iterate easily on all the pig objects to apply the growth function (Figure 2).

```
class Batch:

    def __init__(self, nb_pigs, ID=''):

    def batch_growth(self, param_duration, file_day, i):
        listpigs = deepcopy(self.IDpigs)
        self.AverageLW = sum([self.dict_pigs[i].LW for i in self.IDpigs])/len(listpigs)
        for name in listpigs:
            self.dict_pigs[name].function_growth(param_duration)

class Pig:

    def __init__(self, ADG_pot, Agei, BWi, BGompertz, ID=''):

    def function_growth(self, param_duration):
```

Figure 2. Extraction of Python code (from Classes_Step0.py) which shows how applying the growth function to each pig within a batch.

Then, it is necessary to apply also the Discrete-Events paradigm to simulate the occurrence of the deliveries to slaughterhouse. For this purpose, it is useful to create an agenda of events (Figure 3). In this very simple case, all the events are scheduled at the start of the simulation. However, it is possible to add some events in the agenda into a function, for instance to schedule the entrance of a new batch in the room when the previous batch has been fully delivered to slaughterhouse. The agenda is created as a dictionary in Python, with the keys corresponding to various times of simulation. For each key in the agenda of events, there is a list of events, which is provided as a string of characters. Usually, an event calls a function, in our case it is the function “delivery” of class Batch. Events are executed with the function exec of Python (built-in function) that makes it possible to run code dynamically.

```
param_duration = 120 # maximum fattening duration is 120 days

AGENDA = {(param_duration): ['My_Batch.delivery(param_duration, f_perf, i)'],
          (param_duration-7): ['My_Batch.delivery(param_duration, f_perf, i)'],
          (param_duration-14): ['My_Batch.delivery(param_duration, f_perf, i)'],
          (param_duration-21): ['My_Batch.delivery(param_duration, f_perf, i)'],
          (param_duration-28): ['My_Batch.delivery(param_duration, f_perf, i)']}

for i in range(param_duration + 1):
    My_Batch.batch_growth(param_duration, f_daily, i)
    if i in AGENDA.keys():
        for evs in AGENDA[i]:
            exec(evs)
```

Figure 3. Extraction of Python code (from FGL_NANP2024_Step0.ipynb) which shows how creating an agenda of events and how executing them.

FGL_NANP2024_Step0.ipynb

Simulation for this first step does not need any input file. However, it produces two output text files, the first one provides the performance of each pig (one row per pig) in the batch and is named by default Performances_Step0.txt, the second provides daily output for each pig (one row per pig and per day) and is named by default Daily_Output_Step0.txt. Running the code is made in Jupyter lab by running successively each cell in FGL_NANP2024_Step0.ipynb.

FGL_NANP2024_Step0.ipynb comprises several cells:

- one cell that should be run when the module Classes_Step0.py is modified

- one cell that is run to import the modules

- one cell which initiates the simulation by creating the objects

- one cell which creates the output files (without filling them with output data)

- one cell that runs the simulation (and will fill the output files)

- one cell that provides a very simple display of the outputs.

After simulation, it is possible to get these output files and to analyze the results obtained, but this is a very simple model in which the variability of performance among pigs only results from the normal distribution of potential average daily gain and precocity (BGompterz parameter) provided as inputs, without consideration of any possible covariance between parameters.

2. Step 1. Individual-based model of a pig-fattening batch, which simulates the effects of feeding strategies on animal performance

In Step 1, the objective is to build and test an individual-based model of a batch of pigs, which accounts for the effects of the feeding strategy applied, and which applies the principles described in Vautier et al. (2013) DOI to generate consistently the variability of potential for feed intake and protein deposition among pigs. In this Step 1, the same principles as in Step 0 are used to apply both individual-based and discrete-events paradigms.

For Step 1, the model is made of three different files: Functions_Step1.py, Classes_Step1.py, and FGL_NANP2024_Step1.ipynb.

Functions_Step1.py

This module contains various functions that will be used in the model. By externalizing them to a single module, it makes (hopefully) the other modules easier to read and understand:

- function read_param is a function which reads the general input file that is chosen to initiate the simulation (in Step1, it can be either General_Input_file_Step1_twophase.csv or General_input_file_Step1_multiphase.csv). It provides a dictionary called param.

- function lect_profile is a function which reads the files that provide pig profiles for simulation. In Step 1, it can be either profile.csv, profiles_barrows.csv or profiles_females.csv. It returns a dictionary, which contains all the characteristics of the pig profiles, with the names of the profiles being the keys of the dictionary.

- function lect_feed is a function which reads the file that provides the nutrients contents of the feeds. It returns a dictionary which contains all the characteristics of the feeds, with the names of the feeds being the keys of the dictionary.

- function_INRAPorc is a function which applies a simplified version of the InraPorc model (van Milgen et al. 2008). This function is applied to each pig object each day. It could have been included as a method of the class Pig, but it has been externalized to make it easier to read. It applies the principles of InraPorc to calculate the potential for protein deposition according to the pig profile. It also calculates the protein deposition allowed by net energy supply and by digestible lysine supply (but not the other amino acids). Then it calculates also the excretion of fecal and urinary N. It also calculates the excretion of organic matter and of the residue (for further calculation of enteric methane and emissions of methane from manure).

Classes_Step1.py :

The class diagram is updated to consider the characteristics of the feeds that are used for the simulations (Figure 4).

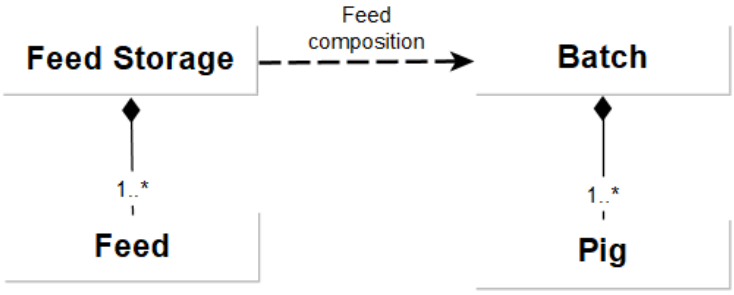


Figure 4. Class diagram of the model provided for Step 1 (Classes_Step1.py)

In the __init__ function of the class Batch, pig objects are generated according to the principles of Vautier et al. (2013). They are defined by their profile in InraPorc model which comprises 5 parameters describing feed intake and protein deposition potentials : a and b that are parameters of the feed intake equation, PDm wich is mean potential protein deposition, BGompertz which corresponds to precocity and initial body weight. In Step 1, pig objects can be generated by applying variability in pig profiles (case when Variability ==1) among pigs or by providing to each pig the same profile (case When Variability ==0).

The class Batch has two functions included, which are batch_growth and delivery. The function batch_growth iterates each day on each pig of the batch to apply the growth function called function_INRAPorc. The delivery function simulates the delivery of pigs to the slaughterhouse. This function is applied as an event. Each time the delivery function is applied, the model first makes the list of pigs that are ready to be delivered to slaughterhouse (i.e. pigs that have reached the target slaughter weight), checks whether the fattening room should be emptied (in this case all pigs will be considered ready to be delivered to slaughterhouse) or not, changes the status of the pigs sent to slaughterhouse, and remove the pigs delivered from the dictionary of pig objects of the batch.

FGL_NANP2024_Step1.ipynb

This is the main file which is used to run the simulation. It comprises several cells:

- one cell that should be run when the models Calsses_Step1.py or Functions_Step1.py are modified

- one cell that is run to import the modules

- one cell that creates a dictionary of InraPorc parameters that are used, whatever the pig profiles applied.

- one cell that reads the input files to 1) provide dictionaries for pig profiles and feeds, and 2) to create the list called seq_reul which describes the feed sequence applied.

- one cell which initiates the simulation by creating the objects

- one cell which creates the output files (without filling them with output data)

- one cell that runs the simulation (and will fill the output files)

- one cell that provides a very simple display of the outputs, that can further developed by the users.

Input files

There are several input files needed to run the code in Step 1. All these input files can be modified by the user in order to test some other pig profiles and feeding strategies:

- profiles_barrows.csv contains a dataset of INRAPorc pig profiles for barrows according to the average profile used by Brossard et al. (2014 DOI) and the variance-covariance matrix defined by Vautier et al. (2013).

- profiles_females.csv contains a dataset of INRAPorc pig profiles for females according to the average profile used by Brossard et al. (2014 DOI) and the variance-covariance matrix defined by Vautier et al. (2013).

- profile.csv provides a single pig profile that can be applied when the user wants to test the model without applying variability of profile among pigs.

- Feeds.csv provides a description of each Feed that will be created in the Feed Storage. In this file, feeds are described for their contents of various nutrients as in the InraPorc® software, with one row per feed. By default, the file contains 4 feeds provided:

- GrowingStd which is a standard growing feed,
- FinishingStd which is a standard finishing feed,
- FeedAStd which is a feed designed to be applied at the start of the fattening period and which has a high level of digestible amino acids contents,
- FeedBStd that is a feed designed to be applied at the end of the fattening period and which has a low level of digestible amino acids contents.

- General_Input_file_Step1_twophase.csv and General_input_file_Step1_multiphase.csv provide the general input file for the simulation. It indicates which are the files chosen for the pig profiles and the feeds. They also describe the feeding sequence which will be applied. General_Input_file_Step1_twophase.csv describes a two-phase feeding sequence which uses feeds GrowingStd and FinishingStd. General_input_file_Step1_multiphase.csv describes a ten-phase feeding sequence wich uses FeedAStd and FeedBStd.

Table 1. Extraction of General_input_file_Step1_multiphase.csv that shows how to describe a multiphase feeding sequence, in 10 phases with a blend of FeedAStd and FeedBStd for each phase according to Percentage_feed1.

Name	nbrows	nbcols	Value	Type

Number_Phase	1	1	10	int
Mode	1	10	2 2 2 2 2 2 2 2 2 2	int
Point_diet_change	1	10	0 40 50 60 70 80 90 100 110 120	float
Feed1	1	10	FeedAStd FeedAStd FeedAStd FeedAStd FeedAStd FeedAStd FeedAStd FeedAStd FeedAStd FeedAStd	str
Feed2	1	10	FeedBStd FeedBStd FeedBStd FeedBStd FeedBStd FeedBStd FeedBStd FeedBStd FeedBStd FeedBStd	str
Percentage_feed1	1	10	100 83 69 59 50 42 33 23 12 0	float

Table 1 illustrates how a multiphase feeding sequence can be described in these input files. Number_Phase provides the number of feeding phases. Mode is by default 2 (means that the end of each sequence is defined by average live weight of the batch). Point_diet_change provides the average live weights that define the end of each phase. Feed1 and Feed2 provide the names of the feeds that are used together in each phase. Percentage_feed1 defines the percentage of Feed1 in the blend for each feeding phase.

Output files

The model provides two output files, which are initiated in FGL_NANP2024_Step1.ipynb, and further filled during the simulation. In both files, to make it easily understandable, all the variables are named with their usual labels in animal nutrition, and their units are provided in the labels as well :

- Performances_ Step1_twophases.txt or Performances_ Step1_ multiphase.txt depending on the general input file used to run the code. It provides the performance of each pig in the batch (one row per pig)..

- Daily_output_ Step1_twophases.txt or Daily_output_ Step1_ multiphase.txt depending on the general input file used to run the code. It provides the daily output for each pig (one row per pig and per day). -

3. Step 2. Individual-based model of a pig-fattening batch, which simulates the effects of feeding strategies on animal performance, and calculates the environmental impacts through Life Cycle Assessment (LCA)

In Step 2, the objective is to associate the individual-based model of a batch of pigs built in Step 1 with life cycle assessment (LCA), in order to provide the environmental evaluation of feeding strategies. Therefore, the following description will only focus on the additional files and functions added relatively to Step 1. In order to perform this environmental evaluation, the model provided In Step 2 applies the different steps of a LCA as follows:

- Goal and scope, system boundaries: the environmental evaluation targets only the resources and emissions associated to the pig fattening unit, including the upstream processes for feed production (Figure 5). In this model, the inventories for feed production, including upstream processes, are not included, and environmental impacts of feeds are provided in the input file. Since the environmental assessment focuses only on the pig fattening unit, the functional unit to express the environmental impacts is the kg of weight gain during fattening.

- Inventory analysis: To calculate the emissions in the building (from pigs and from manure) and from manure storage, the principles of the Modagor model (Cadéro et al. 2018) have been applied. N Emissions (N_NH3, N_N2O, N_NOx) are calculated successively in the building and during manure storage, applying the emission factors provided in IPCC (2006) and EMEP/EAA (2019). Enteric methane is calculated according to Rigolot et al. (2010) as proposed in LEAP guidelines. Methane emissions from manure are calculated according to IPCC (2006).

- Life Cycle Impact assessment: the Environmental Footprint 3.0 method is applied by default, through characterization factors and by providing impacts of feeds calculated with this method.

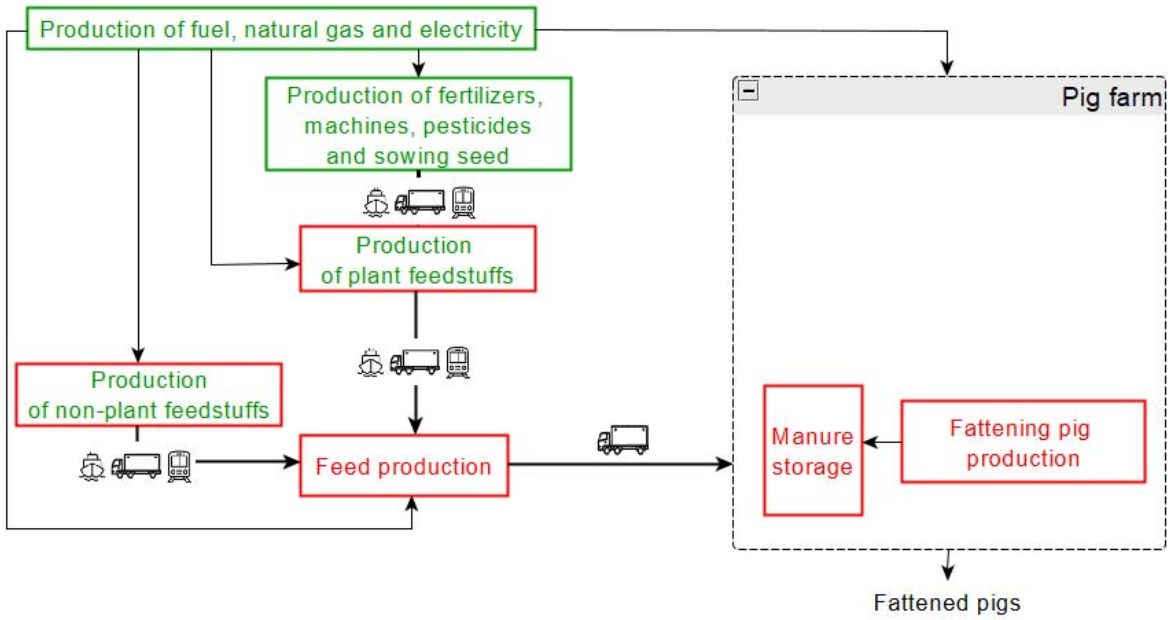


Figure 5. Simplified system boundaries defined for the model provided in Step 2.

Functions_Step2.py

In addition to the functions already available in Step 1, the module Functions_Step2.py contains a function called function_LCA_pig that performs the environmental assessment for each individual pig.

This function calculates emissions in pig building and at manure storage, according to the above - mentioned principles. It calculates the climate change and acidification impacts of the total feed consumed (Figure 6).

The climate change (CC) impact per kg of weight gain is calculated by this function by adding the CC impact of the total feed intake, the total methane emissions multiplied by their characterization factor, the total N2O emissions multiplied by their characterization factor, the climate change impact of the feed transport from the feed factory to the farm, and the CC impact associated to electricity consumption on farm.

The Acidification (AC) impact per kg of weight gain is calculated by this function by adding AC impact of the total feed intake, total ammonia emissions multiplied by their characterization factor, total NOx emissions multiplied by their characterization factor, acidification impact of the feed transport from the feed factory to the farm, and AC impact associated to electricity consumption on farm.

```
    OnePig.CC_feed = sum([OnePig.dict_stockFeed[f] * Feeds.dict_feeds[f].CC for f in OnePig.dict_stockFeed.keys()])
    OnePig.AC_feed = sum([OnePig.dict_stockFeed[f] * Feeds.dict_feeds[f].AC for f in OnePig.dict_stockFeed.keys()])

    OnePig.CC_gain = ((OnePig.CC_feed +
                       (OnePig.CH4_St + OnePig.CH4_enteric) * CF['CC_EF30']['CH4'] +
                       (OnePig.N_N2OEmiH + OnePig.N_N2OEmiSt) * (44/28) * CF['CC_EF30']['N2O'] +
                       (OnePig.Cumulated_FM_intake/1000) * input['Feed_Road'] * background['LorryTransport']['CC_EF30'] +
                        input['ElectricityHousing'] * background['ElectricitykWh']['CC_EF30'])/ (OnePig.LW - OnePig.LWi))

    OnePig.AC_gain = ((OnePig.AC_feed +
                       (OnePig.N_NH3EmiH + OnePig.N_NH3EmiSt) * (17/14) * CF['Acidification_EF30']['NH3'] +
                       (OnePig.N_NOxEmiH + OnePig.N_NOxEmiSt) * (46/14) * CF['Acidification_EF30']['NOx'] +
                      OnePig.Cumulated_FM_intake * (input['Feed_Road']/1000) * background['LorryTransport']['Acidification_EF30'] +
                      input['ElectricityHousing'] * background['ElectricitykWh']['Acidification_EF30'])/ (OnePig.LW - OnePig.LWi))
```

Figure 6. Extraction of Python code (from Functions_Step2.py) which shows how calculating the environmental impacts of each individual pig (per kg of body weight gain during fattening).

Input files

In addition to the input files already needed in Step 1, the calculation of the impacts of the production on climate change and acidification needs the following input files:

- Background_Data.csv provides environmental impacts for the production of 1kWh electricity, and for 1t.km of lorry transport.

- InputLCAFR.csv provides input parameters used for environmental evaluation (emission factors….)

- CF.csv provides characterization factors for climate change and acidification according to the EF 3.0 method/

The file Feeds.csv needs also to contain the climate change and acidification impacts of the feeds and the general input file has to specify which is the input file used to provide parameters to calculate the emissions.

Output files

In addition to the output files already produced in Step 1, model in Step 2 provides an output file called Performances_LCA_Step2_twophases.txt or Performances_LCA_Step2_multiphase.txt depending on the input file used. It provides the environmental assessment for each pig for climate change and for acidification per kg of body weight gain during fattening. It also provides the total climate change and acidification impacts for the total feed consumption for each pig.

4. Comparison of various feeding strategies with and without variability of feed intake and growth potential among pigs

In both Step 1 and Step 2, there are two different feeding strategies available and described in the input files:

- a two-phase strategy with a growing and a finishing feed

- a ten-phase strategy providing a blend of various proportions of FeedAStd and FeedBStd.

At each step, it is also possible to compare the model outputs with and without variability of pig profiles. In the general input file used, when parameter Variability equals 1, variability among pigs is applied. When parameter Variability equals 0, one single pig profile is used for all pigs.