FitMultiCell

Release 0.0.7

The FitMultiCell developers

May 23, 2022

USER'S GUIDE

1 Install **3**
- 1.1 Requirements . . . 3
- 1.2 Dependencies . . . 3
- 1.3 Install from PIP . . . 3
- 1.4 Install from GIT . . . 3
- 1.5 Install as user into your home directory (recommended) . . . 4
- 1.6 Install on AWS . . . 4
- 1.7 Upgrade . . . 4

2 Running FitMultiCell pipeline in an HPC infrastructure **5**
- 2.1 Install fitmulticell . . . 5
- 2.2 Install pyABC . . . 5
- 2.3 Install Morpheus . . . 5
- 2.4 Install Redis . . . 6
- 2.5 Running fitmulticell . . . 6

3 Examples **7**
- 3.1 Getting started . . . 7
  - 3.1.1 Basic usage . . . 7
  - 3.1.2 Fit SBML model . . . 12
- 3.2 PEtab . . . 15
  - 3.2.1 PEtab extention for FitMultiCell . . . 15
  - 3.2.2 Fitting multiple conditions using PEtab format . . . 25
- 3.3 Application examples . . . 39
  - 3.3.1 Single cell motility . . . 39
  - 3.3.2 Classification . . . 45
- 3.4 Summary statistics . . . 47
  - 3.4.1 Cluster Count . . . 47
  - 3.4.2 Cluster size . . . 51
  - 3.4.3 Count Active infected Cell . . . 54
  - 3.4.4 Count Cell types . . . 58

4 Containerization **63**
- 4.1 Docker . . . 63
- 4.2 Singularity . . . 63
- 4.3 Charlie-cloud . . . 63

5 Youtube tutorial series **65**
- 5.1 Part 1 . . . 65
- 5.2 Part 2 . . . 65

5.3 Part 3 . . . 65
5.4 Part 4 . . . 65

6 Release Notes **67**
6.1 0.0 series . . . 67
6.1.1 0.0.7 (2021-11-28) . . . 67
6.1.2 0.0.6 (2021-11-28) . . . 67
6.1.3 0.0.5 (2021-11-10) . . . 67
6.1.4 0.0.4 (2021-11-03) . . . 68
6.1.5 0.0.3 (2019-11-25) . . . 68
6.1.6 0.0.2 (2019-10-10) . . . 68
6.1.7 0.0.1 (2019-09-24) . . . 68

7 Contact **69**

8 License **71**

9 Logo **73**

10 API reference **75**
10.1 FitMultiCell . . . 75
10.2 Model . . . 75

11 Contribute **79**
11.1 Workflow . . . 79
11.2 Environment . . . 79
11.2.1 Pre-commit hooks . . . 79
11.2.2 Tox . . . 80
11.3 GitLab CI/CD . . . 80
11.4 Documentation . . . 80
11.5 Unit tests . . . 80
11.6 PEP8 . . . 81

12 Deploy **83**
12.1 Versions . . . 83
12.2 Create a new release . . . 83
12.2.1 Merge into main . . . 83
12.2.2 Create a release on GitHub . . . 84
12.3 Upload to PyPI . . . 84

13 Indices and tables **85**

Python Module Index **87**

Index **89**

Source code: https://gitlab.com/fitmulticell/fit

An Integrated Platform for Data-Driven Modeling of Multi-Cellular Processes. It delivers that by combining the simulation tool morpheus and the likelihood-free inference tool pyABC.

The overall aim of FitMultiCell is to build and validate an open platform for modelling, simulation and parameter estimation of multicellular systems, which will be utilised to mechanistically answer biomedical questions based on imaging data.

CHAPTER

ONE

INSTALL

1.1 Requirements

This package requires Python 3.7 or later. It is tested on Linux, but should also run on MacOS and Windows, with some limitations regarding parallelization.

1.2 Dependencies

This package relies on the tools morpheus and pyABC.

- To install morpheus, checkout https://morpheus.gitlab.io/#download.
- To install pyabc, checkout https://pyabc.readthedocs.io/en/latest/installation.html.

1.3 Install from PIP

The package can be installedd from the Python Package Index PyPI via pip:

```
pip3 install fitmulticell
```

1.4 Install from GIT

If you want the bleeding edge version, install directly from gitlab:

```
pip3 install git+https://gitlab.com/fitmulticell/fit
```

If you need to have access to the source code, you can instead download it via:

```
git clone https://gitlab.com/fitmulticell/fit
```

and then install from the local repository via:

```
cd fit
pip3 install -e .
```

1.5 Install as user into your home directory (recommended)

Installing fitmulticell into your system's Python distribution can be problematic as you might not want to change your system's Python installation or you don't have root rights. The recommended alternative is to install fitmulticell into your home directory with:

```
pip3 install --user fitmulticell
```

1.6 Install on AWS

Here is a link to a video that explains how to install the FitMultiCell pipeline in the AWS Amazon cloud:

https://cloudstore.zih.tu-dresden.de/index.php/s/NMiLZmpktaGGgWp

Moreover, here is the list of commands used in the video:

```
sudo apt update
sudo apt install --yes libgl1-mesa-glx libegl1-mesa libxrandr2 libxrandr2 libxss1␣
↪libxcursor1 \
libxcomposite1 libasound2 libxi6 libxtst6 gnuplot-nox joe
wget https://repo.anaconda.com/archive/Anaconda3-2020.11-Linux-x86_64.sh
bash Anaconda3-2020.11-Linux-x86_64.sh
source ~/.bashrc
pip install jupyter
pip install pyabc
pip install fitmulticell
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout mykey.key -out mycert.pem
jupyter notebook password
jupyter notebook --generate-config
jupyter notebook &
```

1.7 Upgrade

If you want to upgrade an installation, replace `install` by `install --upgrade` in the above pip commands.

CHAPTER

TWO

RUNNING FITMULTICELL PIPELINE IN AN HPC INFRASTRUCTURE

To run the pipeline in an HPC infrastructure, you need first to install pyABC, fitmulticell, Morpheus, and Redis server on the cluster.

2.1 Install fitmulticell

To install fitmulticell on an HPC infrastructure, you need to install it under the user domain without the need for root privileges. Please be aware that on such systems, the user usually doesn't have roots right. To check how you can do that, please check: https://gitlab.com/fitmulticell/fit/-/blob/master/doc/install.rst#install-as-user-into-your-home-directory-recommended

2.2 Install pyABC

A similar installation causing should be followed here to install pyABC. WE should also install it in the user domain. To check how you can do that, please check: https://pyabc.readthedocs.io/en/latest/installation.html#install-as-user-into-your-home-directory-recommended

2.3 Install Morpheus

Installing Morpheus is a tricky part. You can find detailed information about how you can install Morpheus in your machine in the following link: https://morpheus.gitlab.io/. Before starting Morpheus installation, be sure that you Install/load all dependencies. You can see the list of dependencies here: https://gitlab.com/morpheus.lab/morpheus#install. to check available module in your system, you can use `module avail`.

Please be aware that we will mainly work with the CLI version of Morpehsu and that there is no need to install the GUI version. Be sure to replace `make && sudo make install` with `make && make install` since you usually don't have root privileges.

After the installation complete, we can then use Morpheus executable which is located usually on ~/morpheus/build/morpheus/morpheus

2.4 Install Redis

In order to run pyABC on a distributed system, we need to use Redis server. It might already be installed in your cluster by the IT team. You need to check that first. You can check the available module in your system by using `module avail`. If Redis is available, then you can load it, e.g `module load Redis`.

Nonetheless, there is a chance that Redis is et not installed in the system. A detailed guide about how to install, setup, and use Redis server is provided in the following links: https://pyabc.readthedocs.io/en/latest/sampler.html#how-to-set-up-a-redis-based-distributed-cluster, https://redis.io/topics/quickstart.

2.5 Running fitmulticell

After installing all required components of the pipeline, you can now start using it in your cluster.

CHAPTER

THREE

EXAMPLES

3.1 Getting started

3.1.1 Basic usage

In this notebook, we demonstrate how to use FitMultiCell to interface the model development and simulation tool Morpheus with the parameter inference tool pyABC.

[1]:
```
import fitmulticell as fmc
import pyabc

import scipy as sp
import numpy as np
import pandas as pd
import os
import tempfile

%matplotlib inline
pyabc.settings.set_figure_params('pyabc')  # for beautified plots
```

[2]:
```
fmc.log.print_version()
```

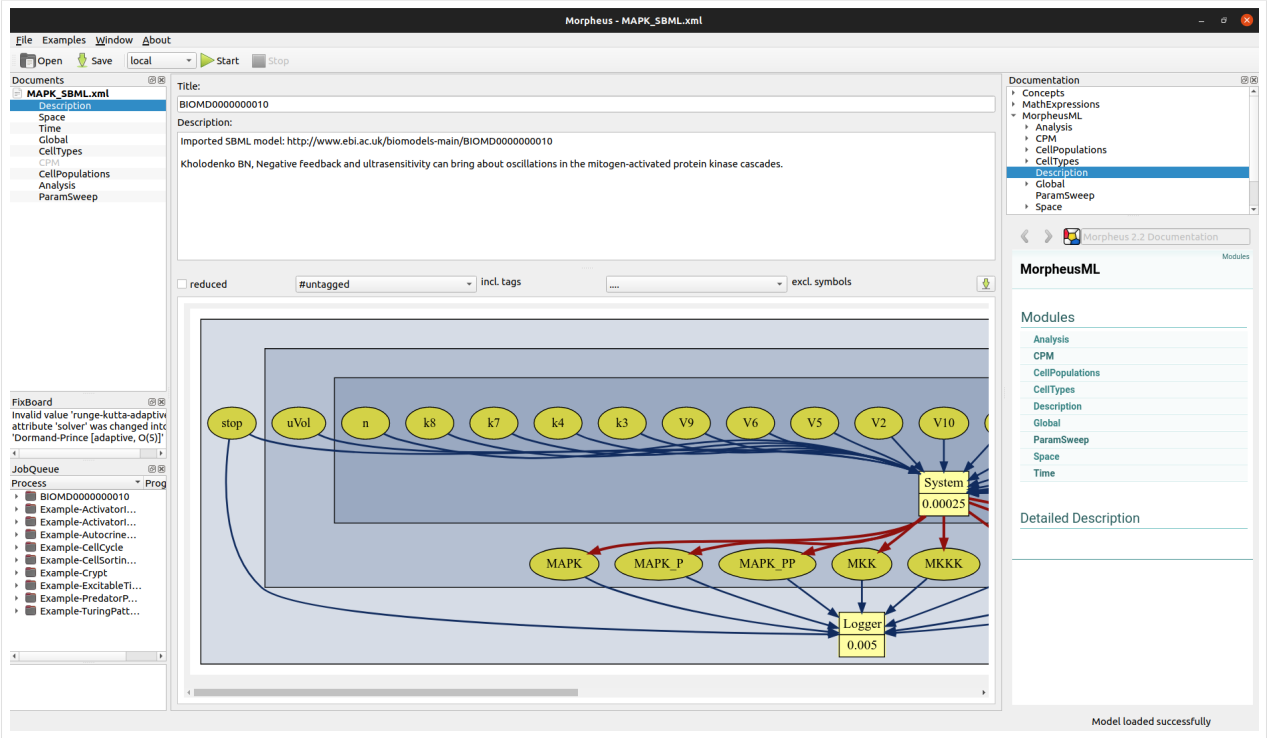
```
Running FitMultiCell 0.0.7 with Morpheus 2.2.3, pyABC 0.11.11 on Python 3.7.5 at 2022-01-
↪20 12-37.
```

Inference problem definition

We consider the problem of fitting various parameters for a simple ODE model of MAPK signaling, taken from Morpheus' built-in examples, with synthetic data. Model construction and refinement can be done via Morpheus' GUI, see its homepage for tutorials and help.

[3]:
```
# model file
model_file = os.path.join("models", "MAPK_SBML.xml")

# run morpheus
# !morpheus-gui --url=$model_file

# display gui
from IPython.display import Image
Image("_static/morpheus_gui.png", width=800)
```

[3]:

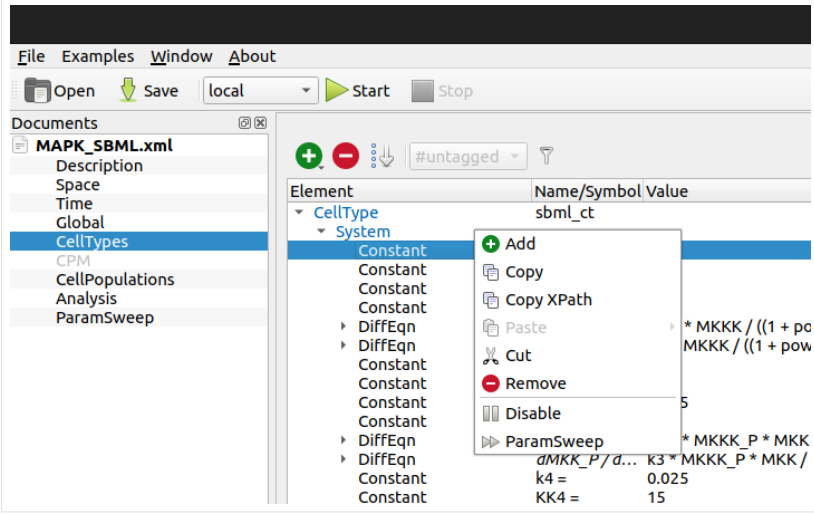


The Morpheus model specification is contained in a single XML file, which can be read in by FitMultiCell via the *MorpheusModel <fitmulticell.model.MorpheusModel>* class. See the API documentation for configuration options, e.g. allowing to specify the Morpheus executable, summary statistics, and level of output. The default summary statistics are just the values logged in the "logger.csv" file generated by Morpheus.

Assuming various of the model parameters to be unknown, we intend to infer them from observed data via pyABC. Therefore, we specify the mapping of parameter ids to XPaths in the model file. XPaths of model entities can be easily obtained via the Morpheus GUI "Copy XPath" context menu.

[4]:
```
Image("_static/morpheus_xpath.png", width=400)
```

[4]:



[5]:
```
# parameter mapping
par_map = {
```

(continues on next page)

(continued from previous page)

```
    "V1": "./CellTypes/CellType/System/Constant[@symbol='V1']",
    "V2": "./CellTypes/CellType/System/Constant[@symbol='V2']",
    "k3": "./CellTypes/CellType/System/Constant[@symbol='k3']",
}

# Morpheus model
model = fmc.model.MorpheusModel(
    model_file=model_file,
    par_map=par_map,
    raise_on_error=False,
)
print(model)
```

```
MorpheusModel {
  name          : models/MAPK_SBML.xml
  par_map       : {'V1': "./CellTypes/CellType/System/Constant[@symbol='V1']", 'V2': "./
↪CellTypes/CellType/System/Constant[@symbol='V2']", 'k3': "./CellTypes/CellType/System/
↪Constant[@symbol='k3']"}
  sumstat       : None
  executable    : morpheus
}
```

A short check whether everything works, prior to the analysis, is usually useful.

[6]:
```
model.sanity_check()
```

```
FitMultiCell.Model INFO: Sanity check successful
```

Let us generate some dummy observed data from synthetic ground-truth parameters `true_pars`. The parameters we define here must correspond to Constants in the Morpheus XML file.

[7]:
```
true_pars = {'V1': 2.7, 'V2': 0.25, 'k3': 0.025}
limits = {key: (0.5 * val, 2 * val) for key, val in true_pars.items()}

# generate data
observed_data = model.sample(true_pars)
print(observed_data.keys())
```

```
dict_keys(['time', 'cell.id', 'MAPK', 'MAPK_P', 'MAPK_PP', 'MKK', 'MKK_P', 'MKK_PP',
↪'MKKK', 'MKKK_P'])
```

As usual, we have to define a prior for our parameters.

[8]:
```
prior = pyabc.Distribution(
    **{
        key: pyabc.RV("uniform", lb, ub - lb)
        for key, (lb, ub) in limits.items()
    }
)
```

Also, we have to define a distance which computes a scalar value quantifying the discrepancy of generated and observed data. Note that also this step can be customized, e.g. for arbitrary summary statistics. Here, we use a simple L1 distance between selected outputs.

[9]:
```
def distance(val1, val2):
    d = np.sum(
        [
            np.sum(np.abs(val1[key] - val2[key]))
            for key in ['MAPK_P', 'MKK_P']
        ]
    )
    return d
```

Running the ABC inference

Now, we are able to run our ABC analysis. See the pyABC documentation for configuration options, e.g. regarding parallelization, distances, thresholds, or proposal distributions.

[10]:
```
abc = pyabc.ABCSMC(model, prior, distance, population_size=100)
db_path = "sqlite:///" + os.path.join(tempfile.gettempdir(), "test.db")
abc.new(db_path, observed_data)
```

```
ABC.Sampler INFO: Parallelize sampling on 12 processes.
ABC.History INFO: Start <ABCSMC id=1, start_time=2022-01-20 12:38:11>
```

[10]:
```
<pyabc.storage.history.History at 0x7f969c715190>
```

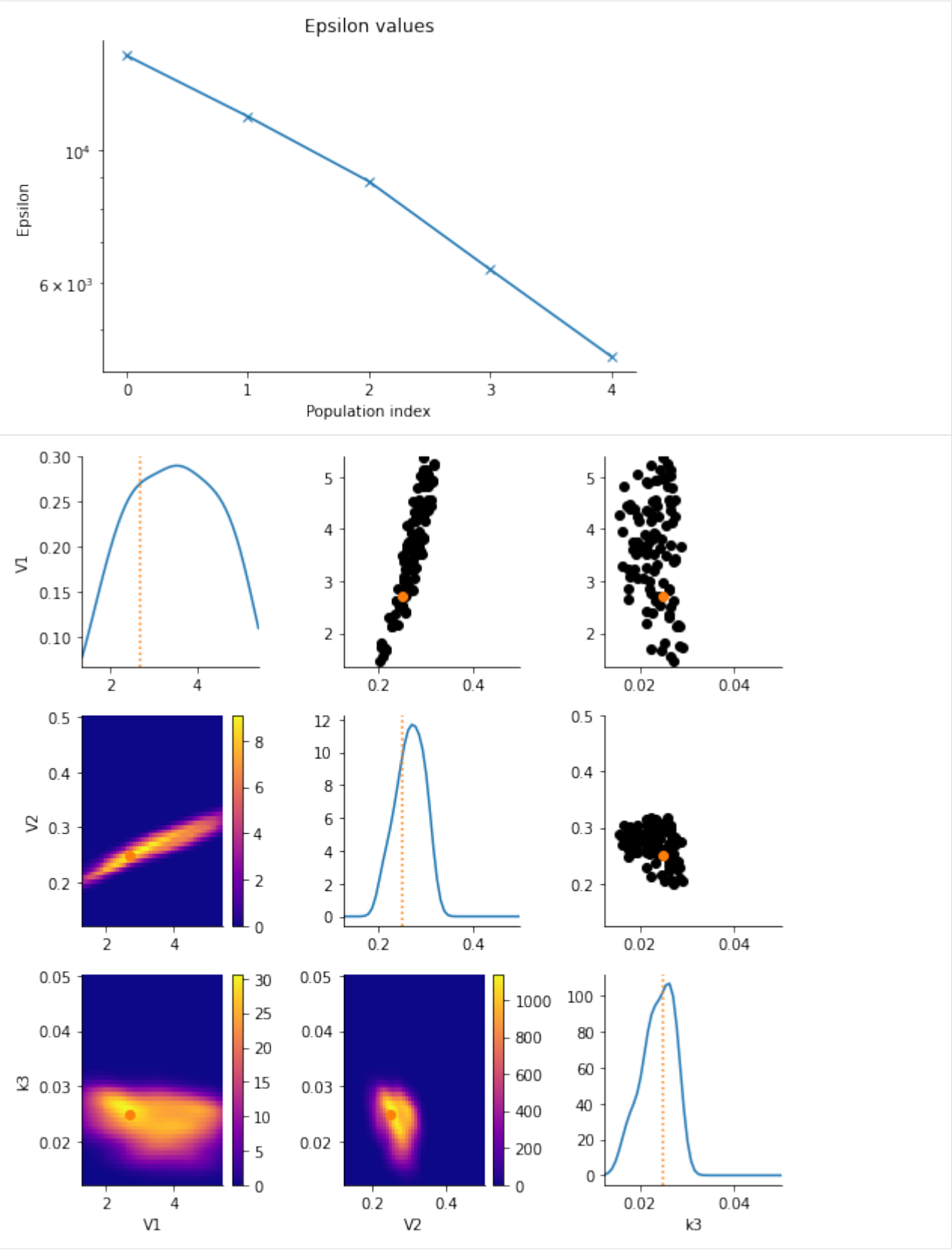
[11]:
```
history = abc.run(max_nr_populations=5)
```

```
ABC INFO: Calibration sample t = -1.
ABC INFO: t: 0, eps: 1.42432419e+04.
ABC INFO: Accepted: 100 / 186 = 5.3763e-01, ESS: 1.0000e+02.
ABC INFO: t: 1, eps: 1.14215269e+04.
ABC INFO: Accepted: 100 / 235 = 4.2553e-01, ESS: 7.9988e+01.
ABC INFO: t: 2, eps: 8.96405597e+03.
ABC INFO: Accepted: 100 / 237 = 4.2194e-01, ESS: 8.9190e+01.
ABC INFO: t: 3, eps: 6.12055219e+03.
ABC INFO: Accepted: 100 / 178 = 5.6180e-01, ESS: 8.2634e+01.
ABC INFO: t: 4, eps: 4.79957563e+03.
ABC INFO: Accepted: 100 / 199 = 5.0251e-01, ESS: 8.6064e+01.
ABC INFO: Stop: Maximum number of generations.
ABC.History INFO: Done <ABCSMC id=1, duration=0:00:40.425872, end_time=2022-01-20 12:38:
↪51>
```

Visualization and analysis

The `history` file contains the analysis results. We can use pyABC's visualization routines to analyse the result.

[12]:
```
pyabc.visualization.plot_epsilons(history)
pyabc.visualization.plot_kde_matrix_highlevel(history, limits=limits, refval=true_pars);
```


Epsilon values
Epsilon
Population index
V1
V2
k3

3.1.2 Fit SBML model

In this example, we will try another feature in the FitMulticell pipeline, which enable users to use the already existing feature of Morpheus, which is to use SBML model directly by internally converting them into MorpheusML.

For this example, we will use an SBML model from BioModels, https://www.ebi.ac.uk/biomodels/BIOMD0000000254. The first step will be to install the model and copy the path to the model's xml file. Then we will construct a model object using the recently installed SBML model.

[1]:
```
import pyabc
import fitmulticell as fmc
import scipy as sp
import numpy as np
import pandas as pd
import os
import tempfile
from pathlib import Path
import matplotlib.pyplot as plt

file_ = str((Path(os.getcwd())) / 'models' / 'BIOMD0000000254_url.xml')

par_map = {'V_in': './Global/Constant[@symbol="V_in"]',
           'kp': './Global/Constant[@symbol="kp"]'}
model = fmc.model.MorpheusModel(
    file_, par_map=par_map,
    executable="/usr/local/bin/morpheus",
    show_stdout=False, show_stderr=True,
    raise_on_error=False)
```

Then, we will convert the SBML model into MorpheusML model.

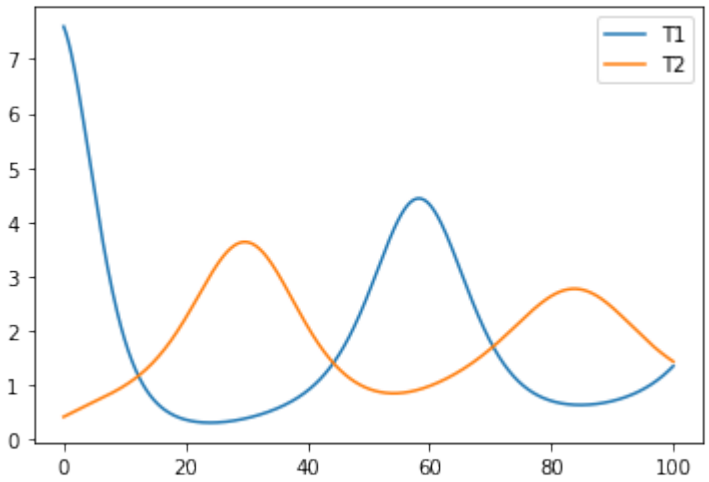
[2]:
```
MLmodel = model.SBML_to_MorpheusML()
```
```
FitMultiCell.Model INFO: SBML model successfully convert to MorpheusML
```

Now, our model is properly constructed and we can simulate directly from it using some ground truth parameters.

[3]:
```
true_pars = {'V_in': 0.36, 'kp': 6}
observed_data = model.sample(true_pars)
```

lets plot the simulation output for two species, namely T1 and T2.

[4]:
```
plt.plot(observed_data["time"], observed_data["T1"], label="T1")
plt.plot(observed_data["time"], observed_data["T2"], label="T2")
plt.legend()
```
[4]:
```
<matplotlib.legend.Legend at 0x7ff692d75f10>
```



Now, let's start the inference process for two parameters (V_in, kp), but before that, let's define all required variables.

[5]:
```
limits = {key: (0, 10) for key, val in true_pars.items()}

prior = pyabc.Distribution(**{key: pyabc.RV("uniform", lb, ub - lb)
                              for key, (lb, ub) in limits.items()})

def distance(val1, val2):
    d = np.sum([np.sum(np.abs(val1[key] - val2[key])) \
                for key in ['T1', 'T2']])
    return d
```

[6]:
```
limits
```

[6]:
```
{'V_in': (0, 10), 'kp': (0, 10)}
```

We can now start the inference process.

[7]:
```
abc = pyabc.ABCSMC(model, prior, distance, population_size=20)
db_path = "sqlite:///" + os.path.join(tempfile.gettempdir(), "test.db")
history = abc.new(db_path, observed_data)
abc.run(max_nr_populations=8)
```

```
ABC.Sampler INFO: Parallelize sampling on 12 processes.
ABC.History INFO: Start <ABCSMC id=7, start_time=2022-04-07 12:34:33>
ABC INFO: Calibration sample t = -1.
ABC INFO: t: 0, eps: 4.78966521e+05.
ABC INFO: Accepted: 20 / 46 = 4.3478e-01, ESS: 2.0000e+01.
ABC INFO: t: 1, eps: 1.44892639e+05.
ABC INFO: Accepted: 20 / 58 = 3.4483e-01, ESS: 1.6766e+01.
ABC INFO: t: 2, eps: 1.05292104e+04.
ABC INFO: Accepted: 20 / 60 = 3.3333e-01, ESS: 9.2184e+00.
ABC INFO: t: 3, eps: 2.37777556e+03.
ABC INFO: Accepted: 20 / 62 = 3.2258e-01, ESS: 1.9520e+01.
ABC INFO: t: 4, eps: 1.45106353e+03.
ABC INFO: Accepted: 20 / 55 = 3.6364e-01, ESS: 1.8209e+01.
```

(continues on next page)

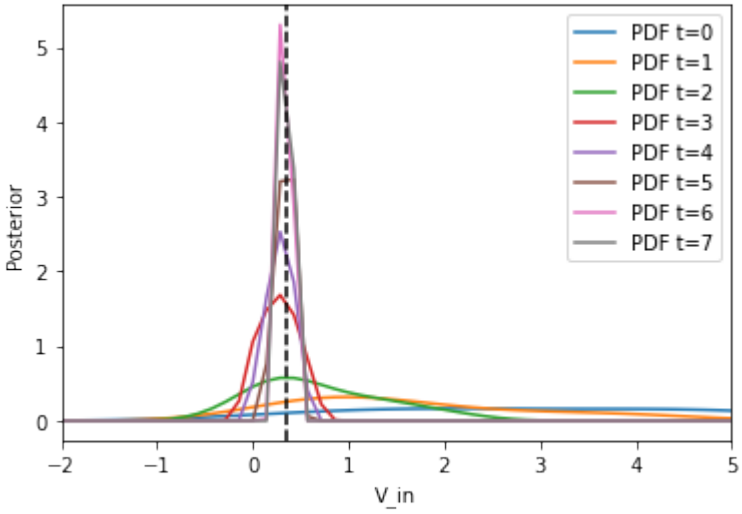
(continued from previous page)

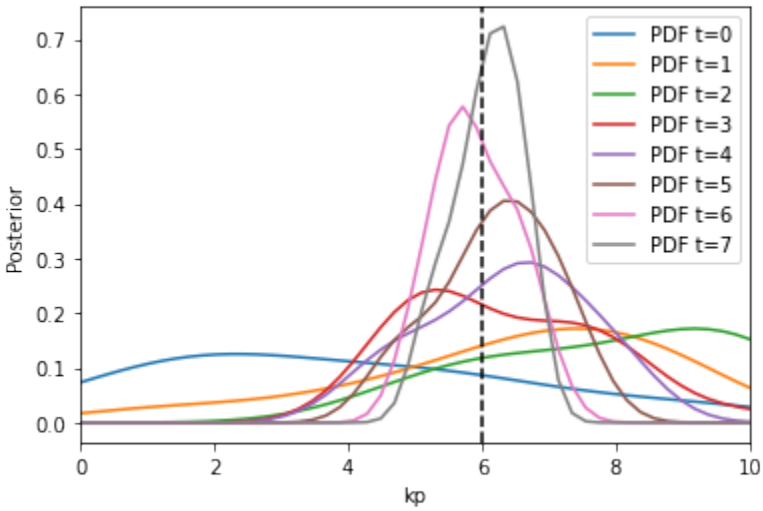
```
ABC INFO: t: 5, eps: 1.23634676e+03.
ABC INFO: Accepted: 20 / 55 = 3.6364e-01, ESS: 1.7455e+01.
ABC INFO: t: 6, eps: 9.27132245e+02.
ABC INFO: Accepted: 20 / 63 = 3.1746e-01, ESS: 1.7877e+01.
ABC INFO: t: 7, eps: 6.60341867e+02.
ABC INFO: Accepted: 20 / 70 = 2.8571e-01, ESS: 1.8117e+01.
ABC INFO: Stop: Maximum number of generations.
ABC.History INFO: Done <ABCSMC id=7, duration=0:00:11.668199, end_time=2022-04-07 12:34:
↪45>
```

[7]:
```
<pyabc.storage.history.History at 0x7ff711765210>
```

Now, we can visualize the obtained Bayesian posterior approximation along with the acceptance threshold

[9]:
```
fig, ax = plt.subplots()
for t in range(history.max_t + 1):
    df, w = history.get_distribution(m=0, t=t)
    pyabc.visualization.plot_kde_1d(
        df,
        w,
        xmin=-2,
        xmax=5,
        x="V_in",
        xname="V_in",
        ax=ax,
        label=f"PDF t={t}",
    )
ax.axvline(true_pars["V_in"], color="k", linestyle="dashed")
ax.legend();
plt.show()

fig, ax = plt.subplots()
for t in range(history.max_t + 1):
    df, w = history.get_distribution(m=0, t=t)
    pyabc.visualization.plot_kde_1d(
        df,
        w,
        xmin=0,
        xmax=10,
        x="kp",
        xname="kp",
        ax=ax,
        label=f"PDF t={t}",
    )
ax.axvline(true_pars["kp"], color="k", linestyle="dashed")
ax.legend();
```





[ ]:

3.2 PEtab

3.2.1 PEtab extention for FitMultiCell

In this example, we will show how a PEtab problem can be defined in FitMultiCell pipeline. For this, we will use a 1D activator-inhibitor model1. First, we will try to define the problem without the use of PEtab format.

Regular FitMultiCell problem

At first, lets's import the required packages.

[1]:
```
import petab_MS
from fitmulticell.PEtab.base import PetabImporter
from fitmulticell.model import MorpheusModel as morpheus_model
from fitmulticell.model import MorpheusModels as morpheus_models
from fitmulticell.sumstat import SummaryStatistics
from pyabc.sampler import RedisEvalParallelSampler
from pyabc.sampler import MulticoreEvalParallelSampler
from pyabc import QuantileEpsilon
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math
import pyabc

import matplotlib.pylab as plt
from pathlib import Path
import os
import tempfile
```

Now, let's define the xpath for each parameter.

[2]:
```
par_map = {'rho_a': './Global/System/Constant[@symbol="rho_a"]',
           'mu_i': './Global/System/Constant[@symbol="mu_i"]',
           'mu_a': './Global/System/Constant[@symbol="mu_a"]',
           }
```

Here, we define the ground truth value for each parameter which we used to generate the senthitic data.

[3]:
```
obs_pars = {'rho_a': 0.01,
            'mu_i': 0.03,
            'mu_a': 0.02,
           }
obs_pars_log = {key: math.log10(val) for key, val in obs_pars.items()}
```

Next, we will import the senthitic data and give the path to Morpheus model.

[4]:
```
condition1_obs = str((Path(os.getcwd())) / 'PEtab_problem_1' / 'Small.csv')

model_file = str((Path(os.getcwd())) / 'PEtab_problem_1' / 'ActivatorInhibitor_1D.xml')
```

[5]:
```
data = pd.read_csv(condition1_obs, sep='\t')
dict_data = {}
for col in data.columns:
    dict_data[col] = data[col].to_numpy()
```

Then, we will construct Morpheus model and SummaryStatistic object.

[6]:
```
sumstat = SummaryStatistics(ignore=["time", "loc"])
```

(continues on next page)

(continued from previous page)

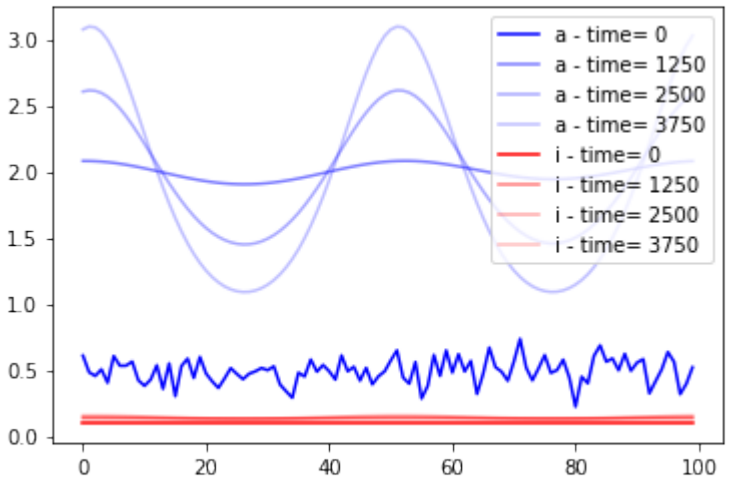
```
model = morpheus_model(
    model_file, par_map=par_map, clean_simulation=True,
    par_scale="lin",
    sumstat=sumstat)
```

Let's now run a simulation with the ground truth parameters and plot it to see how it looks.

[7]:
```
tryjectory = model.sample(obs_pars)
```

[8]:
```
color=['b','r']

color_index=0
size = 100
for key, val in tryjectory.items():
    if key == "loc": continue
    alpha_index = 1
    if key == "space.x": continue
    for i,time in zip(range (0,int(len(val)),50*size),range(0,4000,50*25)):
        plt.plot(tryjectory["space.x"][i:i+size], val[i:i+size],label=key +" - time=
↪"+str(time),color=color[color_index],alpha=1/alpha_index)
        alpha_index += 1
    color_index+=1
plt.legend()
```

[8]:
```
<matplotlib.legend.Legend at 0x7f509964ff10>
```



Then, we will further define the limits and the scale for parameters space.

[9]:
```
limits = {key: (-3,-1) for
          key, val in obs_pars.items()}
```

[10]:
```
model.par_scale="log10"
```

[11]:
```
prior = pyabc.Distribution(**{key: pyabc.RV("uniform", lb, ub - lb)
                              for key, (lb, ub) in limits.items()})
```

Now, we will define our objective function:

[12]:
```
def eucl_dist(sim, obs):
    total = 0
    for key in sim:
        if key in (
            'loc', "time", "space.x"):
                continue
        x = np.array(sim[key])
        y = np.array(obs[key])
        if x.size != y.size:
            return np.inf

        total += np.sum((x - y) ** 2)
    return total
```

Now, we are ready to start the fitting.

[13]:
```
abc = pyabc.ABCSMC(model, prior, eucl_dist, population_size=100)
```

```
ABC.Sampler INFO: Parallelize sampling on 12 processes.
```

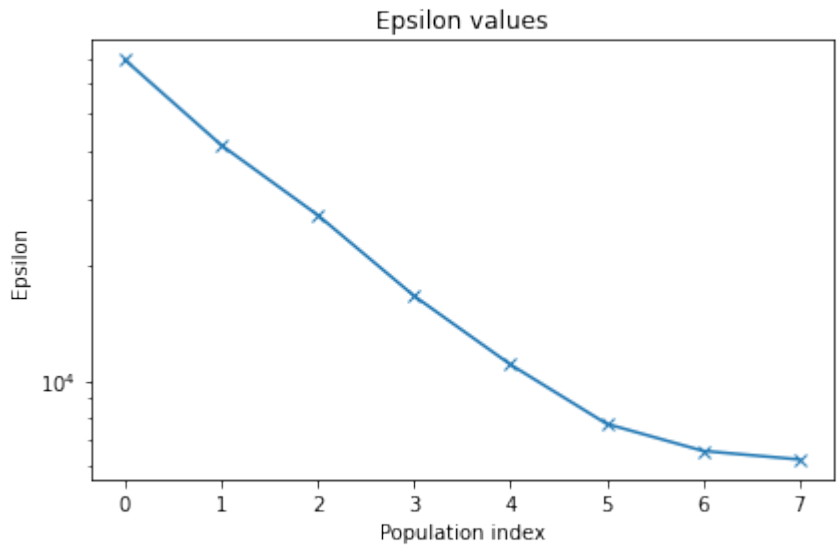
[14]:
```
db_path = "sqlite:///" + os.path.join(tempfile.gettempdir(), "ActivatorInhibitor_1D.db")
history = abc.new(db_path, dict_data)
history = abc.run(max_nr_populations=8)
```

```
ABC.History INFO: Start <ABCSMC id=2, start_time=2022-01-20 11:45:18>
ABC INFO: Calibration sample t = -1.
ABC INFO: t: 0, eps: 6.92881712e+04.
ABC INFO: Accepted: 100 / 215 = 4.6512e-01, ESS: 1.0000e+02.
ABC INFO: t: 1, eps: 4.14571844e+04.
ABC INFO: Accepted: 100 / 254 = 3.9370e-01, ESS: 9.0927e+01.
ABC INFO: t: 2, eps: 2.72167705e+04.
ABC INFO: Accepted: 100 / 216 = 4.6296e-01, ESS: 9.0174e+01.
ABC INFO: t: 3, eps: 1.67243558e+04.
ABC INFO: Accepted: 100 / 244 = 4.0984e-01, ESS: 8.9091e+01.
ABC INFO: t: 4, eps: 1.10596384e+04.
ABC INFO: Accepted: 100 / 209 = 4.7847e-01, ESS: 7.9877e+01.
ABC INFO: t: 5, eps: 7.73285579e+03.
ABC INFO: Accepted: 100 / 290 = 3.4483e-01, ESS: 9.1969e+01.
ABC INFO: t: 6, eps: 6.57135021e+03.
ABC INFO: Accepted: 100 / 278 = 3.5971e-01, ESS: 8.3232e+01.
ABC INFO: t: 7, eps: 6.24093899e+03.
ABC INFO: Accepted: 100 / 265 = 3.7736e-01, ESS: 7.2012e+01.
ABC INFO: Stop: Maximum number of generations.
ABC.History INFO: Done <ABCSMC id=2, duration=0:05:08.607414, end_time=2022-01-20 11:50:
↪26>
```

Let's now plot the epsilon over different iteration.

[15]:
```
pyabc.visualization.plot_epsilons(history)
```

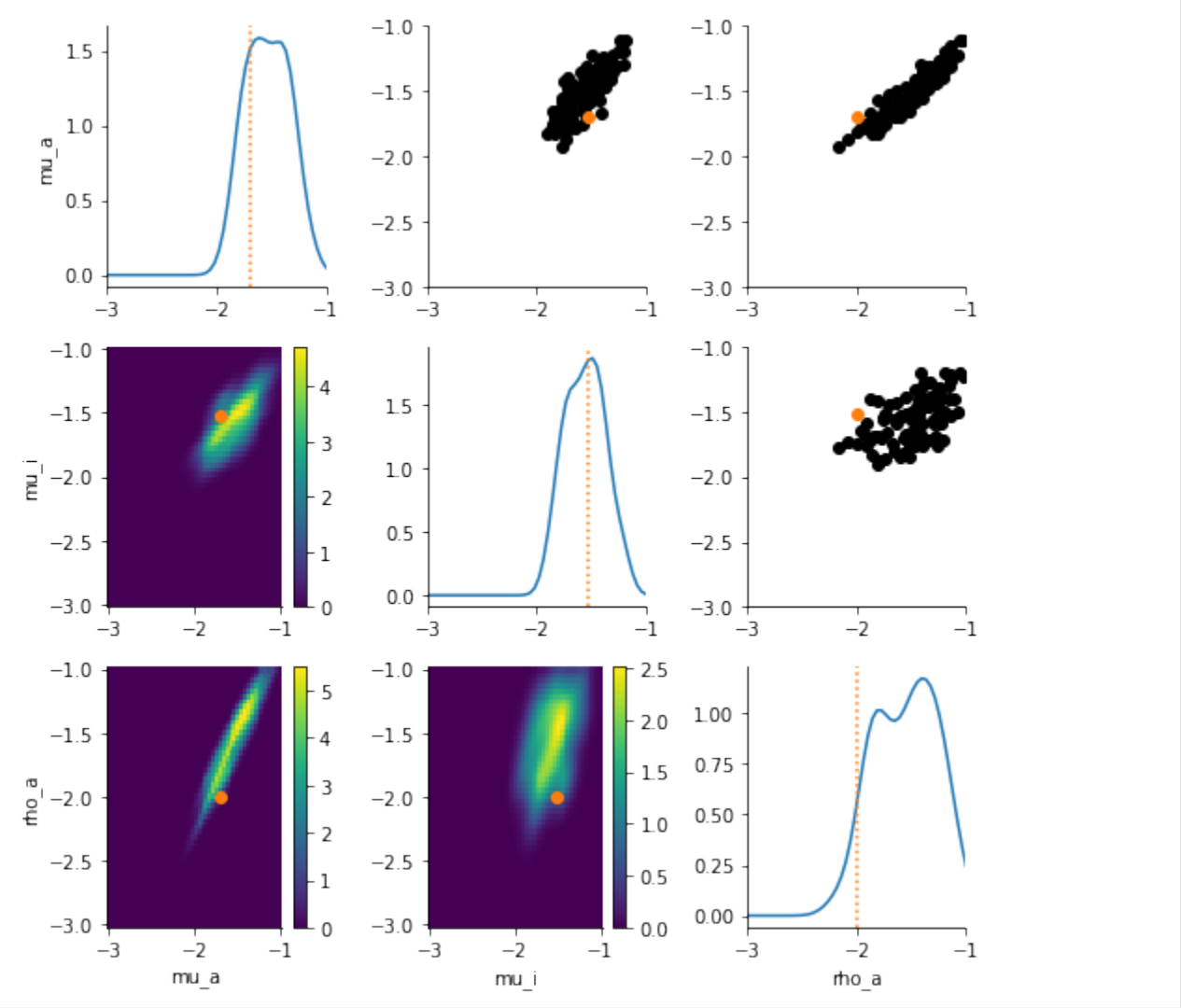
[15]:
```
<AxesSubplot:title={'center':'Epsilon values'}, xlabel='Population index', ylabel=
↪'Epsilon'>
```



Also, let's plot the kernal density function for our parameters of interest.

[16]:
```
df, w = history.get_distribution(t=history.max_t)
pyabc.visualization.plot_kde_matrix(df, w, limits=limits, refval=obs_pars_log)
```

[16]:
```
array([[<AxesSubplot:ylabel='mu_a'>, <AxesSubplot:>, <AxesSubplot:>],
       [<AxesSubplot:ylabel='mu_i'>, <AxesSubplot:>, <AxesSubplot:>],
       [<AxesSubplot:xlabel='mu_a', ylabel='rho_a'>,
        <AxesSubplot:xlabel='mu_i'>, <AxesSubplot:xlabel='rho_a'>]],
      dtype=object)
```



Define the problem using PEtab extention

Next, we want to fit the same model but now we want to use PEtab format to construct our problem.

We will start by import a PEtab problem from the yamel file that we already define according to PEtab standared.

[17]:
```
petab_problem_path = str((Path(os.getcwd())) / 'PEtab_problem_1' / 'ActivatorInhibitor_
↪1D.yaml')
petab_problem = petab_MS.Problem.from_yaml(petab_problem_path)
```

[18]:
```
importer = PetabImporter(petab_problem)
```

Then, we will generate all required component from the PEtab problem, such as prior, measurement, parameters' scale, etc.

[19]:
```
PEtab_prior = importer.create_prior()
```

```
/home/emad/Insync/blackhand.3@gmail.com/Google_Drive/Bonn/Github/libpetab-python-MS/
↪petab_MS/parameters.py:375: RuntimeWarning: invalid value encountered in log10
  return np.log10(parameter)
```

[20]:
```
par_map_imported = importer.get_par_map()
```

[21]:
```
obs_pars_imported = petab_problem.get_x_nominal_dict(scaled=True)
```

[22]:
```
PEtab_par_scale = petab_problem.get_optimization_parameter_scales()
```

[23]:
```
dict_data_imported = petab_problem.get_measurement_dict()
```

After that, we will be able to construct our model form the PEtab importer

[24]:
```
PEtab_model = importer.create_model()
```

Then, we can modify any of the model's attributes.

[25]:
```
PEtab_model.sumstat.ignore = ["time", "loc"]
PEtab_model.sumstat.sum_stat_calculator = None
```

let's now run a simulation tryjectory for the imported PEtab problem.

[26]:
```
PEtab_tryjectory = PEtab_model.sample(obs_pars_imported)
```

Let's now visulise the model tryjectory for both regular and PEtab problem

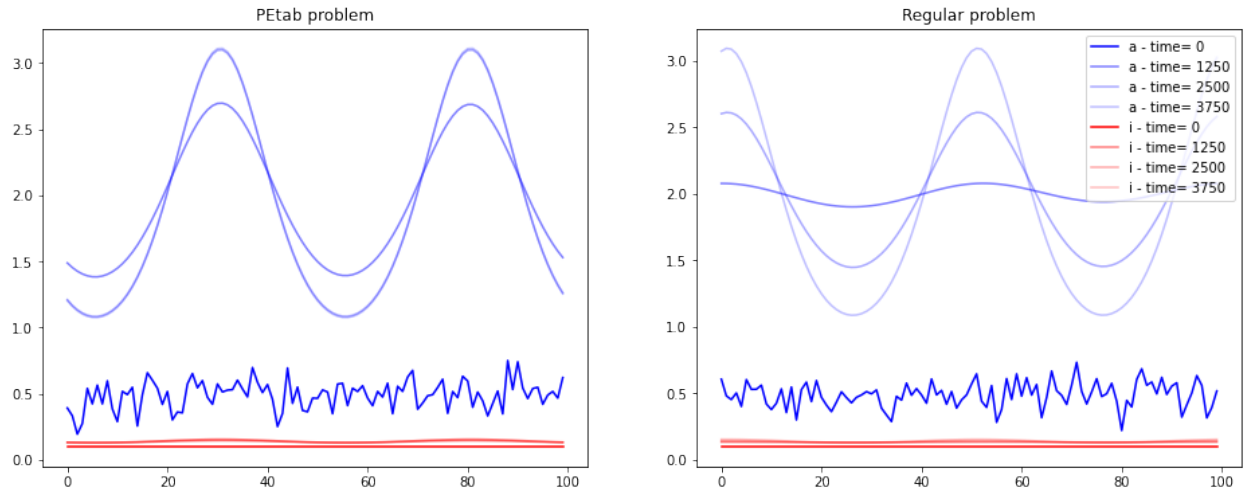
[27]:
```
fig, axes = plt.subplots(1,2, figsize=(16, 6))


color=['b','r']


ax = axes[0]
color_index=0
size = 100
for key, val in PEtab_tryjectory.items():
    if key == "loc": continue
    alpha_index = 1
    if key == "condition1__space.x": continue
    for i,time in zip(range (0,int(len(val)),50*size),range(0,4000,50*25)):
        ax.plot( PEtab_tryjectory["condition1__space.x"][i:i+size], val[i:i+size],
↪label=key +" - time= "+str(time),color=color[color_index],alpha=1/alpha_index)
        alpha_index += 1
    color_index+=1
ax.set_title("PEtab problem")

ax = axes[1]
color_index=0
for key, val in tryjectory.items():
    if key == "loc": continue
```

(continues on next page)

(continued from previous page)

```
    alpha_index = 1
    if key == "space.x": continue
    for i,time in zip(range (0,int(len(val)),50*size),range(0,4000,50*25)):
        ax.plot( tryjectory["space.x"][i:i+size], val[i:i+size],label=key +" - time=
↪"+str(time),color=color[color_index],alpha=1/alpha_index)
        alpha_index += 1
    color_index+=1
ax.legend()
ax.set_title("Regular problem")
```

[27]:
```
Text(0.5, 1.0, 'Regular problem')
```



Let's get one final component before starting the fitting, which is the objective function

[28]:
```
PEtab_obj_function = importer.get_objective_function()
```

Now, we can go ahead and start the fitting of the PEtab problem.

[28]:
```
abc = pyabc.ABCSMC(PEtab_model, PEtab_prior, PEtab_obj_function, population_size=100)
db_path = "sqlite:///" + os.path.join(tempfile.gettempdir(), "PEtab_ActivatorInhibitor_
↪1D.db")
PEtab_history = abc.new(db_path, dict_data_imported)
PEtab_history = abc.run(max_nr_populations=8)
```

```
ABC.Sampler INFO: Parallelize sampling on 12 processes.
ABC.History INFO: Start <ABCSMC id=3, start_time=2022-01-19 12:18:04>
ABC INFO: Calibration sample t = -1.
ABC INFO: t: 0, eps: 1.86342779e+05.
ABC INFO: Accepted: 100 / 209 = 4.7847e-01, ESS: 1.0000e+02.
ABC INFO: t: 1, eps: 4.15795513e+04.
ABC INFO: Accepted: 100 / 243 = 4.1152e-01, ESS: 8.5515e+01.
ABC INFO: t: 2, eps: 2.38253314e+04.
ABC INFO: Accepted: 100 / 257 = 3.8911e-01, ESS: 8.2928e+01.
ABC INFO: t: 3, eps: 1.50475712e+04.
ABC INFO: Accepted: 100 / 263 = 3.8023e-01, ESS: 7.3097e+01.
```

(continues on next page)

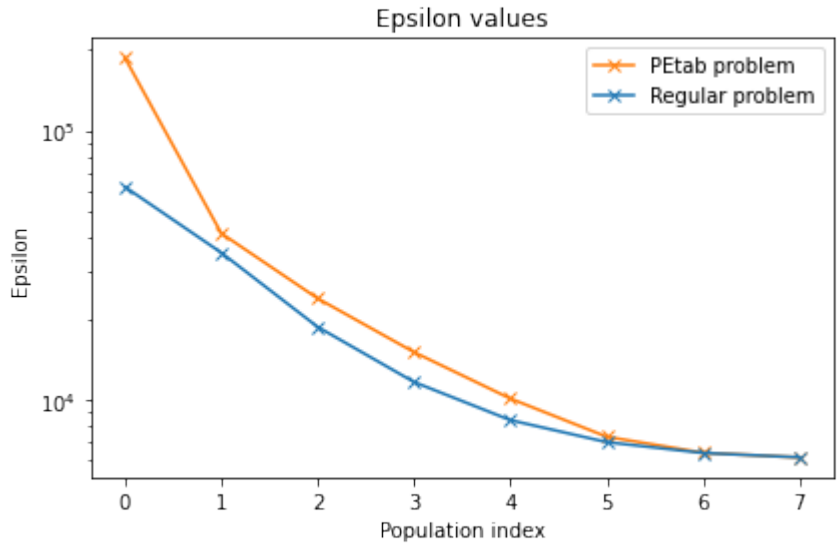
(continued from previous page)

```
ABC INFO: t: 4, eps: 1.01178034e+04.
ABC INFO: Accepted: 100 / 267 = 3.7453e-01, ESS: 7.1077e+01.
ABC INFO: t: 5, eps: 7.30028266e+03.
ABC INFO: Accepted: 100 / 274 = 3.6496e-01, ESS: 7.4931e+01.
ABC INFO: t: 6, eps: 6.36639092e+03.
ABC INFO: Accepted: 100 / 350 = 2.8571e-01, ESS: 8.0178e+01.
ABC INFO: t: 7, eps: 6.11468392e+03.
ABC INFO: Accepted: 100 / 344 = 2.9070e-01, ESS: 7.8303e+01.
ABC INFO: Stop: Maximum number of generations.
ABC.History INFO: Done <ABCSMC id=3, duration=0:07:13.584303, end_time=2022-01-19 12:25:
↪17>
```

Now, let us compare the regular fitting problem to that that was initialized by PEtab. lets start by plotting the epsilon of the two problems.

[29]:
```
ax= pyabc.visualization.plot_epsilons(PEtab_history, colors=['C1'])
pyabc.visualization.plot_epsilons(history,ax=ax, colors=['C0'])
ax.legend(["PEtab problem","Regular problem"])
```

[29]:
```
<matplotlib.legend.Legend at 0x7fb39a53db90>
```



Finally, let's plot and compare the final distribution of the two problems.

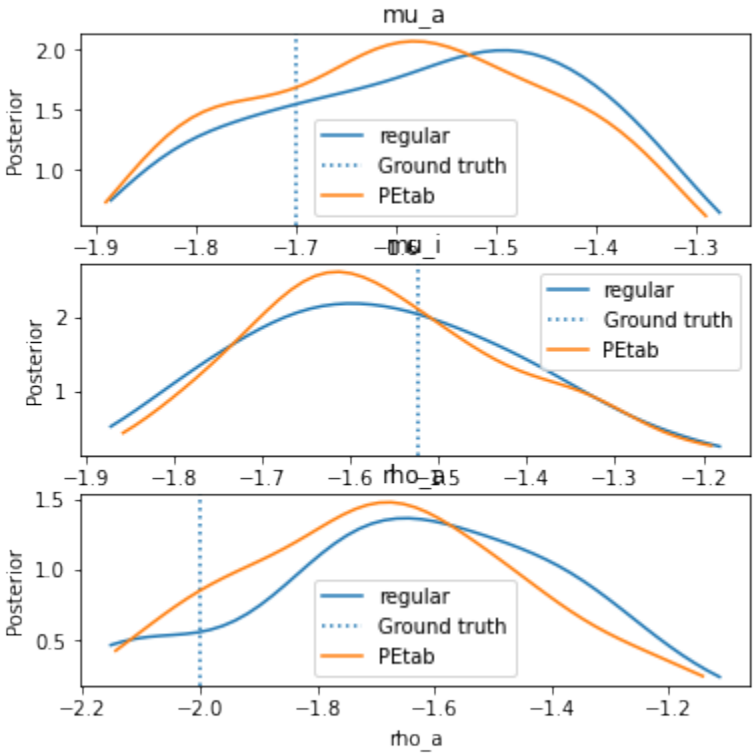
[30]:
```
fig, axes = plt.subplots(3,1, figsize=(6, 6))
ax =axes[0]
df, w = history.get_distribution(t=history.max_t)
pyabc.visualization.plot_kde_1d(df, w, 'mu_a',title="mu_a", refval=obs_pars_log, refval_
↪color='C0', ax=ax)
df, w = PEtab_history.get_distribution(t=PEtab_history.max_t)
pyabc.visualization.plot_kde_1d(df, w, 'mu_a', refval_color='C1', ax=ax)
ax.legend(["regular","Ground truth","PEtab"])

ax =axes[1]
df, w = history.get_distribution(t=history.max_t)
```

(continues on next page)

(continued from previous page)

```
pyabc.visualization.plot_kde_1d(df, w, 'mu_i', title="mu_i", refval=obs_pars_log, refval_
↪color='C0', ax=ax)
df, w = PEtab_history.get_distribution(t=PEtab_history.max_t)
pyabc.visualization.plot_kde_1d(df, w, 'mu_i', refval_color='C1', ax=ax)
ax.legend(["regular","Ground truth","PEtab"])

ax =axes[2]
df, w = history.get_distribution(t=history.max_t)
pyabc.visualization.plot_kde_1d(df, w, 'rho_a', title="rho_a", refval=obs_pars_log,␣
↪refval_color='C0', ax=ax)
df, w = PEtab_history.get_distribution(t=PEtab_history.max_t)
pyabc.visualization.plot_kde_1d(df, w, 'rho_a', refval_color='C1', ax=ax)
ax.legend(["regular","Ground truth","PEtab"])
```

[30]:
```
<matplotlib.legend.Legend at 0x7fb396198210>
```



References:

1 A. Gierer, H. Meinhardt: A Theory of Biological Pattern Formation. Kybernetik 12: 30-39, 1972.

[ ]:

3.2.2 Fitting multiple conditions using PEtab format

In this example, we are going to perform the parameter inference using the FitMultiCell pipeline using a cell motility model. This example has two experimental conditions which will be defined using the PEtab format.

Run the first condition without PEtab

First, lets try to only consider the first condition.

[1]:
```
import petab_MS
from fitmulticell.PEtab.base import PetabImporter
from fitmulticell.model import MorpheusModel as morpheus_model
from fitmulticell.model import MorpheusModels as morpheus_models
from fitmulticell.sumstat import SummaryStatistics
from pyabc.sampler import RedisEvalParallelSampler
from pyabc.sampler import MulticoreEvalParallelSampler
from pyabc import QuantileEpsilon
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math
import pyabc
import matplotlib.pylab as plt
from pathlib import Path
import os
import scipy
import tempfile
```

Next, we will define the xpath for each parameters and its groud truth value.

[2]:
```
par_map = {'move.duration.median': './CellTypes/CellType/Constant[@symbol="move.duration.
↪median"]'}
obs_pars = {'move.duration.median': 150.0}
```

Then, we will define the attributes for our first condition.

[3]:
```
condition1_external_file = str((Path(os.getcwd())) / 'PEtab_problem_2' / 'env_noise_0_
↪900a.tif')

exp_cond_map = {'condition1': {'./Global/Field/TIFFReader': ['filename', condition1_
↪external_file]}}
```

Next, we define the ground truth value for each parameter which we will tue to generate the senthitic data.

[4]:
```
obs_pars_log = {key: math.log10(val) for key, val in obs_pars.items()}
```

Next, we will import the senthitic data and give the path to Morpheus model.

[5]:
```
obs_pars_log = {key: math.log10(val) for key, val in obs_pars.items()}
condition1_obs = str((Path(os.getcwd())) / 'PEtab_problem_2' / 'center.x_end_900a.csv')


model_file = str((Path(os.getcwd())) / 'PEtab_problem_2' / 'cell_motility_model.xml')
```

[6]:
```
data1 = pd.read_csv(condition1_obs, sep='\t')
dict_data1 = {}
for col in data1.columns:
    dict_data1[col] = data1[col].to_numpy()
```

Then, we will define the `summary_statistics` and `model` object as follow:

[7]:
```
sumstat = SummaryStatistics(output_file="logger_2.csv", ignore=["time", "cell.id", ])
```

then, we will construct Morpheus model.

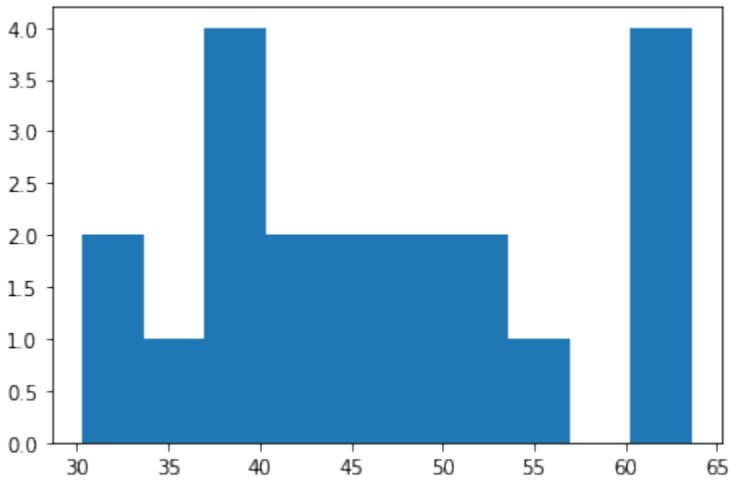
[8]:
```
model_cond1 = morpheus_model(
    model_file, par_map=par_map, clean_simulation=True,
    par_scale="lin", timeout=150, exp_cond_map=exp_cond_map,
    show_stdout=False, show_stderr=False,
    raise_on_error=False,
    executable="/home/emad/morpheus-2.2.5",
    sumstat=sumstat)
```

Now, lets run the model and check the model's output

[9]:
```
condition1_tryjectory = model_cond1.sample(obs_pars_log)
```

[10]:
```
plt.hist(condition1_tryjectory["condition1__cell.center.x"], bins=10)
```

[10]:
```
(array([2., 1., 4., 2., 2., 2., 2., 1., 0., 4.]),
 array([30.32  , 33.6495, 36.979 , 40.3085, 43.638 , 46.9675, 50.297 ,
        53.6265, 56.956 , 60.2855, 63.615 ]),
 <BarContainer object of 10 artists>)
```



Then, we will further define the parameters space

[11]:
```
limits = {key: (0,3) for
          key, val in obs_pars.items()}
```

[12]:
```
model_cond1.par_scale="log10"
```

[13]:
```
prior = pyabc.Distribution(**{key: pyabc.RV("uniform", lb, ub - lb)
                              for key, (lb, ub) in limits.items()})
```

Now, we will define our objective function:

[14]:
```
def eucl_dist(sim, obs):

    if sim == -15:
        ks = np.inf
    else:
        for (k1,v1), (k2,v2) in zip(sim.items(), obs.items()):
            if k1 == "loc":
                continue
            else:
                ks, p_val = scipy.stats.ks_2samp(sim[k1], obs[k2])
    return ks
```

Now, we are ready to start the fitting.

[15]:
```
abc = pyabc.ABCSMC(model_cond1, prior, eucl_dist, population_size=25)
```

```
ABC.Sampler INFO: Parallelize sampling on 12 processes.
```

[16]:
```
db_path = "sqlite:///" + os.path.join(tempfile.gettempdir(), "cell_movement_cond1.db")

history_cond1 = abc.new(db_path, dict_data1)
history_cond1 = abc.run(max_nr_populations=10, minimum_epsilon=0.2)
```

```
ABC.History INFO: Start <ABCSMC id=4, start_time=2022-01-20 14:45:34>
ABC INFO: Calibration sample t = -1.
ABC INFO: t: 0, eps: 1.00000000e+00.
ABC INFO: Accepted: 25 / 36 = 6.9444e-01, ESS: 2.5000e+01.
ABC INFO: t: 1, eps: 9.90000000e-01.
ABC INFO: Accepted: 25 / 63 = 3.9683e-01, ESS: 2.4400e+01.
ABC INFO: t: 2, eps: 5.66210789e-01.
ABC INFO: Accepted: 25 / 71 = 3.5211e-01, ESS: 2.4338e+01.
ABC INFO: t: 3, eps: 3.20000000e-01.
ABC INFO: Accepted: 25 / 68 = 3.6765e-01, ESS: 2.2515e+01.
ABC INFO: t: 4, eps: 2.50000000e-01.
ABC INFO: Accepted: 25 / 104 = 2.4038e-01, ESS: 1.9887e+01.
ABC INFO: t: 5, eps: 2.20000000e-01.
ABC INFO: Accepted: 25 / 211 = 1.1848e-01, ESS: 2.3234e+01.
ABC INFO: t: 6, eps: 2.06597276e-01.
ABC INFO: Accepted: 25 / 204 = 1.2255e-01, ESS: 2.3156e+01.
ABC INFO: t: 7, eps: 1.90000000e-01.
```

(continues on next page)

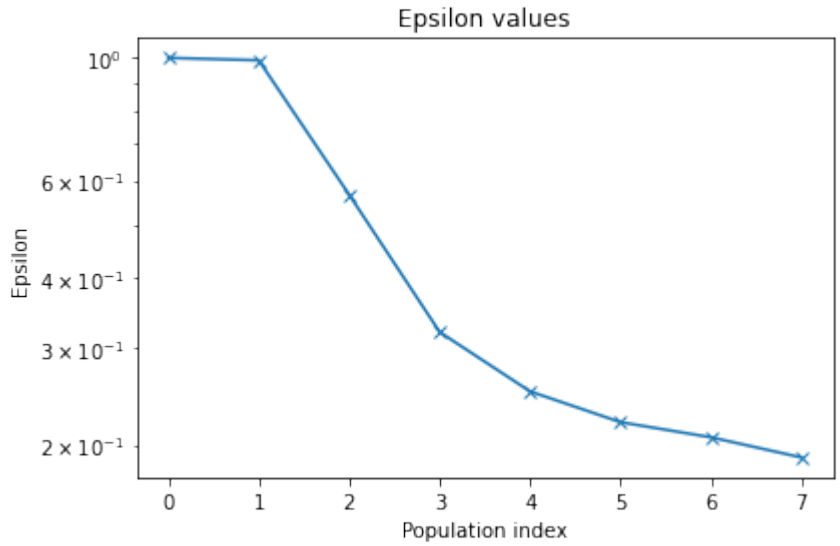
(continued from previous page)

```
ABC INFO: Accepted: 25 / 302 = 8.2781e-02, ESS: 1.7138e+01.
ABC INFO: Stop: Minimum epsilon.
ABC.History INFO: Done <ABCSMC id=4, duration=1:20:11.054993, end_time=2022-01-20 16:05:
↪45>
```

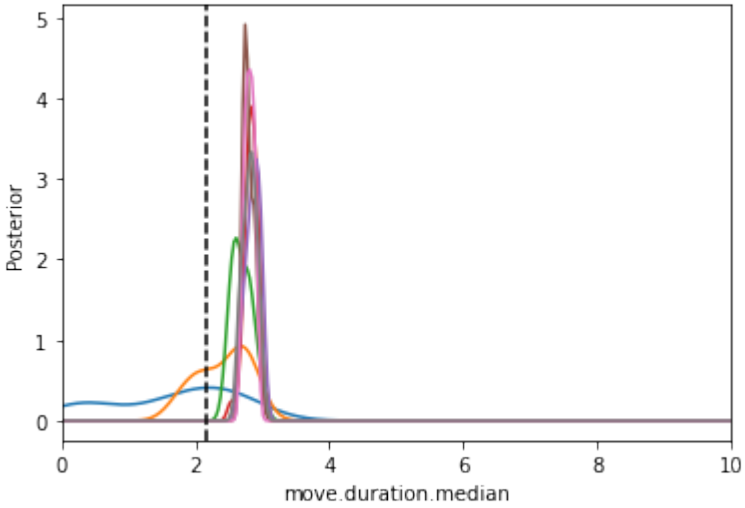
Let us plot some diagnostic plot for the fit.

[17]:
```
pyabc.visualization.plot_epsilons(history_cond1)
```

[17]:
```
<AxesSubplot:title={'center':'Epsilon values'}, xlabel='Population index', ylabel=
↪'Epsilon'>
```



[18]:
```
from pyabc.visualization import plot_kde_1d
fig, ax = plt.subplots()

for t in range(history_cond1.max_t+1):
    particles = history_cond1.get_distribution(m=0, t=t)
    plot_kde_1d(*particles, "move.duration.median",
                label="t={}".format(t), ax=ax,
                xmin=0, xmax=10, numx=300)
ax.axvline(math.log10(150), color="k", linestyle="dashed");
```



Run the second condition without PEtab

Now, we will do the same, but this time only for the second condition.

[19]:
```
condition2_external_file = str((Path(os.getcwd())) / 'PEtab_problem_2' / 'env_noise_0_
↪900b.tif')

exp_cond_map = {'condition2': {'./Global/Field/TIFFReader': ['filename', condition2_
↪external_file]}}
condition2_obs = str((Path(os.getcwd())) / 'PEtab_problem_2' / 'center.x_end_900b.csv')
```

[20]:
```
data2 = pd.read_csv(condition2_obs, sep='\t')
dict_data2 = {}
for col in data2.columns:
    dict_data2[col] = data2[col].to_numpy()
```

[21]:
```
sumstat = SummaryStatistics(output_file="logger_2.csv", ignore=["time", "cell.id", ])

model_cond2 = morpheus_model(
    model_file, par_map=par_map, clean_simulation=True,
    par_scale="lin", timeout=150, exp_cond_map=exp_cond_map,
    show_stdout=False, show_stderr=False,
    raise_on_error=False,
    executable="/home/emad/morpheus-2.2.5",
    sumstat=sumstat)
```

[22]:
```
condition2_tryjectory = model_cond2.sample(obs_pars_log)
```

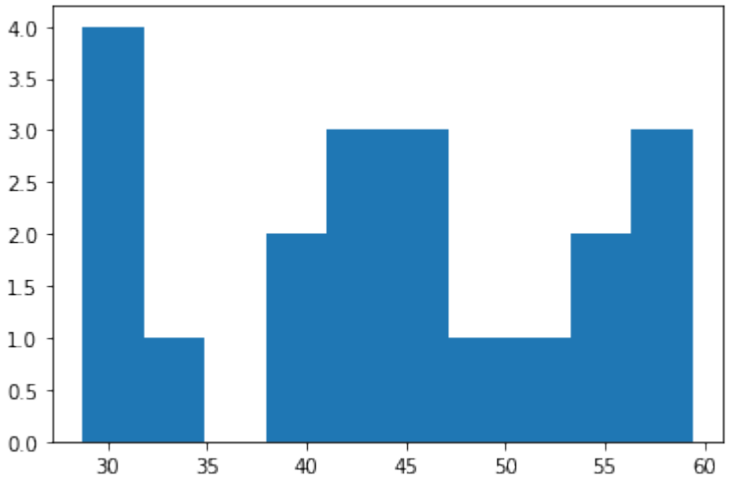
[23]:
```
plt.hist(condition2_tryjectory["condition2__cell.center.x"], bins=10)
```

[23]:
```
(array([4., 1., 0., 2., 3., 3., 1., 1., 2., 3.]),
 array([28.755 , 31.8215, 34.888 , 37.9545, 41.021 , 44.0875, 47.154 ,
```

(continues on next page)

(continued from previous page)

```
       50.2205, 53.287 , 56.3535, 59.42  ]),
<BarContainer object of 10 artists>)
```



Now that we have define the model for the second condition, lets repeat the fitting process for the selected parameter

[24]:
```
abc = pyabc.ABCSMC(model_cond2, prior, eucl_dist, population_size=25)
```

```
ABC.Sampler INFO: Parallelize sampling on 12 processes.
```

[43]:
```
db_path = "sqlite:///" + os.path.join(tempfile.gettempdir(), "cell_movement_cond2.db")
history_cond2 = abc.new(db_path, dict_data2)
history_cond2 = abc.run(max_nr_populations=10,minimum_epsilon=0.2)
```

```
ABC.History INFO: Start <ABCSMC id=1, start_time=2022-01-20 13:41:52>
ABC INFO: Calibration sample t = -1.
ABC INFO: t: 0, eps: 9.80000000e-01.
ABC INFO: Accepted: 50 / 62 = 8.0645e-01, ESS: 5.0000e+01.
ABC INFO: t: 1, eps: 9.80000000e-01.
ABC INFO: Accepted: 50 / 61 = 8.1967e-01, ESS: 4.6432e+01.
ABC INFO: t: 2, eps: 9.80000000e-01.
ABC INFO: Accepted: 50 / 61 = 8.1967e-01, ESS: 4.8013e+01.
ABC INFO: t: 3, eps: 9.80000000e-01.
ABC INFO: Accepted: 50 / 61 = 8.1967e-01, ESS: 4.4342e+01.
ABC INFO: t: 4, eps: 9.80000000e-01.
Process Process-173:
Process Process-178:
Process Process-180:
Process Process-169:
Process Process-170:
Process Process-176:
Process Process-179:
Process Process-175:
Process Process-171:
Traceback (most recent call last):
Traceback (most recent call last):
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
ABC.History INFO: Done <ABCSMC id=1, duration=0:17:09.294207, end_time=2022-01-20 13:59:
↪02>
Process Process-172:
Traceback (most recent call last):
Traceback (most recent call last):
Process Process-177:
Traceback (most recent call last):
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
Traceback (most recent call last):
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
Traceback (most recent call last):
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
Process Process-174:
Traceback (most recent call last):
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
Traceback (most recent call last):
Traceback (most recent call last):
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
Traceback (most recent call last):
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/sampler/multicore_evaluation_
↪parallel.py", line 42, in work
    new_sim = simulate_one()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
```

(continues on next page)

(continued from previous page)

```
    self._target(*self._args, **self._kwargs)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/sampler/multicore_evaluation_
↪parallel.py", line 42, in work
    new_sim = simulate_one()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 452,
↪ in simulate_one
    proposal_id=proposal_id,
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/sampler/multicore_evaluation_
↪parallel.py", line 42, in work
    new_sim = simulate_one()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/sampler/multicore_evaluation_
↪parallel.py", line 42, in work
    new_sim = simulate_one()
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 452,
↪ in simulate_one
    proposal_id=proposal_id,
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/sampler/multicore_evaluation_
↪parallel.py", line 42, in work
    new_sim = simulate_one()
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/sampler/multicore_evaluation_
↪parallel.py", line 42, in work
    new_sim = simulate_one()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 452,
↪ in simulate_one
    proposal_id=proposal_id,
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 452,
↪ in simulate_one
    proposal_id=proposal_id,
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/sampler/multicore_evaluation_
↪parallel.py", line 42, in work
    new_sim = simulate_one()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/sampler/multicore_evaluation_
↪parallel.py", line 42, in work
    new_sim = simulate_one()
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 452,
↪ in simulate_one
    proposal_id=proposal_id,
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 236,
↪ in evaluate_proposal
    t, theta_ss, summary_statistics, distance_function, eps, acceptor, x_0
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 236,
↪ in evaluate_proposal
```

(continues on next page)

(continued from previous page)

```
    t, theta_ss, summary_statistics, distance_function, eps, acceptor, x_0
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/sampler/multicore_evaluation_
↪parallel.py", line 42, in work
    new_sim = simulate_one()
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 236,
↪ in evaluate_proposal
    t, theta_ss, summary_statistics, distance_function, eps, acceptor, x_0
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 452,
↪ in simulate_one
    proposal_id=proposal_id,
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 236,
↪ in evaluate_proposal
    t, theta_ss, summary_statistics, distance_function, eps, acceptor, x_0
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/sampler/multicore_evaluation_
↪parallel.py", line 42, in work
    new_sim = simulate_one()
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/sampler/multicore_evaluation_
↪parallel.py", line 42, in work
    new_sim = simulate_one()
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 452,
↪ in simulate_one
    proposal_id=proposal_id,
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 236,
↪ in evaluate_proposal
    t, theta_ss, summary_statistics, distance_function, eps, acceptor, x_0
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 205, in␣
↪accept
    result = self.summary_statistics(t, pars, sum_stat_calculator)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 452,
↪ in simulate_one
    proposal_id=proposal_id,
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 452,
↪ in simulate_one
    proposal_id=proposal_id,
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 205, in␣
↪accept
    result = self.summary_statistics(t, pars, sum_stat_calculator)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 205, in␣
↪accept
    result = self.summary_statistics(t, pars, sum_stat_calculator)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 236,
↪ in evaluate_proposal
    t, theta_ss, summary_statistics, distance_function, eps, acceptor, x_0
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 452,
↪ in simulate_one
    proposal_id=proposal_id,
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 205, in␣
↪accept
    result = self.summary_statistics(t, pars, sum_stat_calculator)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 236,
↪ in evaluate_proposal
```

(continues on next page)

(continued from previous page)

```
    t, theta_ss, summary_statistics, distance_function, eps, acceptor, x_0
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 205, in␣
↪accept
    result = self.summary_statistics(t, pars, sum_stat_calculator)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 116, in␣
↪summary_statistics
    raw_data = self.sample(pars)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/sampler/multicore_evaluation_
↪parallel.py", line 42, in work
    new_sim = simulate_one()
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 116, in␣
↪summary_statistics
    raw_data = self.sample(pars)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 236,
↪ in evaluate_proposal
    t, theta_ss, summary_statistics, distance_function, eps, acceptor, x_0
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 116, in␣
↪summary_statistics
    raw_data = self.sample(pars)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 452,
↪ in simulate_one
    proposal_id=proposal_id,
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 236,
↪ in evaluate_proposal
    t, theta_ss, summary_statistics, distance_function, eps, acceptor, x_0
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 205, in␣
↪accept
    result = self.summary_statistics(t, pars, sum_stat_calculator)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 116, in␣
↪summary_statistics
    raw_data = self.sample(pars)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 205, in␣
↪accept
    result = self.summary_statistics(t, pars, sum_stat_calculator)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 236,
↪ in evaluate_proposal
    t, theta_ss, summary_statistics, distance_function, eps, acceptor, x_0
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/external/base.py", line 243,␣
↪in sample
    return self(pars)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 452,
↪ in simulate_one
    proposal_id=proposal_id,
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 116, in␣
↪summary_statistics
    raw_data = self.sample(pars)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/external/base.py", line 243,␣
↪in sample
    return self(pars)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 116, in␣
↪summary_statistics
    raw_data = self.sample(pars)
```

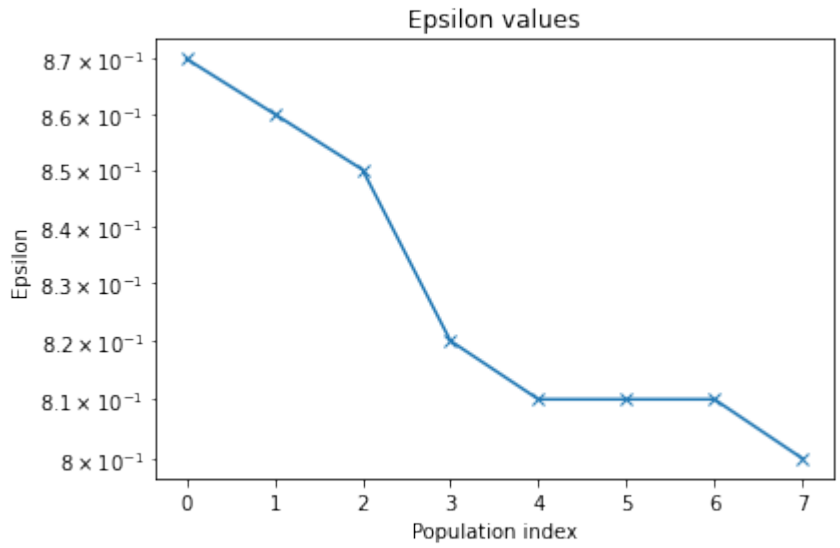
(continues on next page)

(continued from previous page)

```
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/external/base.py", line 243,␣
↪in sample
    return self(pars)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 236,
↪ in evaluate_proposal
    t, theta_ss, summary_statistics, distance_function, eps, acceptor, x_0
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 116, in␣
↪summary_statistics
    raw_data = self.sample(pars)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/model.py", line 205, in␣
↪accept
    result = self.summary_statistics(t, pars, sum_stat_calculator)
  File "/home/emad/Insync/blackhand.3@gmail.com/Google_Drive/Bonn/Gitlab/fit/
↪fitmulticell/model/base.py", line 173, in __call__
    status = self.eh.run(cmd=cmd, loc=loc)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/external/base.py", line 243,␣
↪in sample
    return self(pars)
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/inference_util.py", line 236,
↪ in evaluate_proposal
    t, theta_ss, summary_statistics, distance_function, eps, acceptor, x_0
  File "/home/emad/.local/lib/python3.7/site-packages/pyabc/external/base.py", line 243,␣
↪in sample
    return self(pars)
```

[25]:
```
pyabc.visualization.plot_epsilons(history_cond2)
```

[25]:
```
<AxesSubplot:title={'center':'Epsilon values'}, xlabel='Population index', ylabel=
↪'Epsilon'>
```

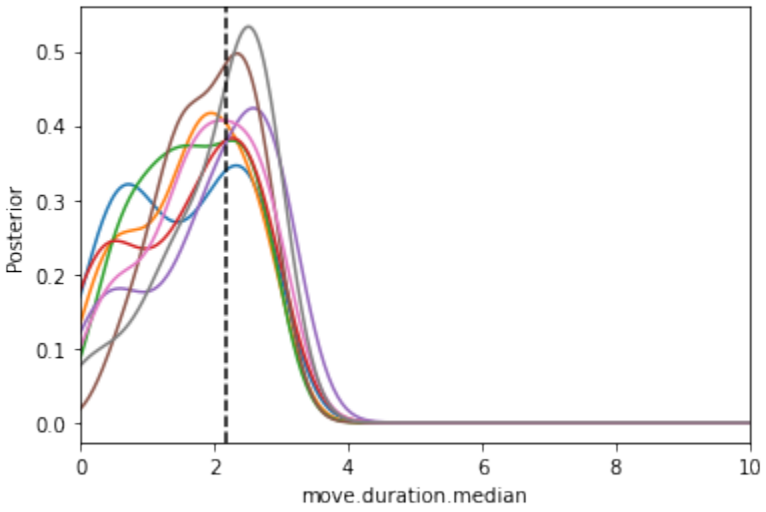


[26]:
```
from pyabc.visualization import plot_kde_1d
fig, ax = plt.subplots()
```

(continues on next page)

(continued from previous page)

```
for t in range(history_cond2.max_t+1):
    particles = history_cond2.get_distribution(m=0, t=t)
    plot_kde_1d(*particles, "move.duration.median",
                label="t={}".format(t), ax=ax,
                xmin=0, xmax=10, numx=300)
ax.axvline(math.log10(150), color="k", linestyle="dashed");
```



Define the problem using PEtab extention

Multiple condition

Now, let's try some features from using the PEtab format. PEtab can facilitate the generation of multiple conditions problem.

Wil will import a PEtab problem where we already specify more than one experimental condition.

[2]:
```
mc_PEtab_problem_path = str((Path(os.getcwd())) / 'PEtab_problem_2' / 'cell_motility.yaml
↪')
mc_PEtab_problem = petab_MS.Problem.from_yaml(mc_PEtab_problem_path)
```

From the PEtab problem, we will generate our priors, parameter maps, observables, parameter scale, and the measerment data.

[3]:
```
mc_importer = PetabImporter(mc_PEtab_problem)
mc_PEtab_prior = mc_importer.create_prior()
mc_par_map_imported = mc_importer.get_par_map()
mc_obs_pars_imported = mc_PEtab_problem.get_x_nominal_dict(scaled=True)
mc_PEtab_par_scale = mc_PEtab_problem.get_optimization_parameter_scales()
mc_dict_data_imported = mc_PEtab_problem.get_measurement_dict()
```

```
/home/emad/Insync/blackhand.3@gmail.com/Google Drive/Bonn/Github/libpetab-python-MS/
↪petab_MS/parameters.py:375: RuntimeWarning: divide by zero encountered in log10
  return np.log10(parameter)
```

Then, we will construct our two models from the PEtab problem.

[4]:
```
PEtab_models = mc_importer.create_model()
PEtab_models.models_list[0].ignore_list=["time","cell.id"]
PEtab_models.models_list[1].ignore_list=["time","cell.id"]
PEtab_models.models_list[0].output_file="logger_2.csv"
PEtab_models.models_list[1].output_file="logger_2.csv"
```

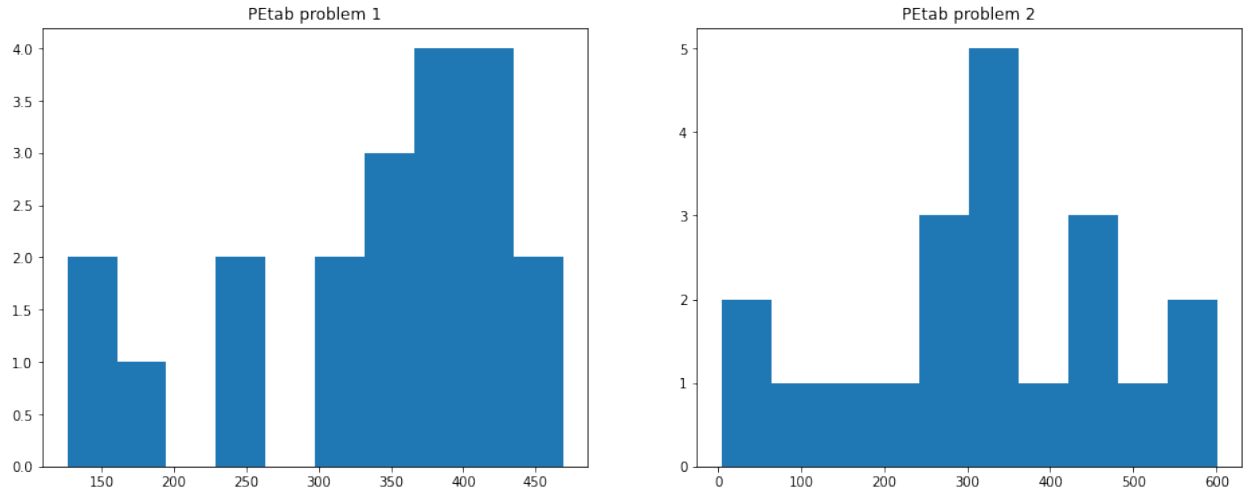
```
FitMultiCell.Model INFO: Successfully loaded model
FitMultiCell.Model INFO: Successfully loaded model
```

[5]:
```
PEtab_mc_tryjectory1 = PEtab_models.models_list[0].sample(mc_obs_pars_imported)
PEtab_mc_tryjectory2 = PEtab_models.models_list[1].sample(mc_obs_pars_imported)
```

Now, let's plot a single tryjectory for both conditions:

[6]:
```
fig, axes = plt.subplots(1,2, figsize=(16, 6))
ax = axes[0]

ax.hist(PEtab_mc_tryjectory1["condition1__IdSumstat__cell.center.x"], bins=10)
ax.set_title("PEtab problem 1")

ax = axes[1]
ax.hist(PEtab_mc_tryjectory2["condition2__IdSumstat__cell.center.x"], bins=10)
ax.set_title("PEtab problem 2")
```

[6]:
```
Text(0.5, 1.0, 'PEtab problem 2')
```



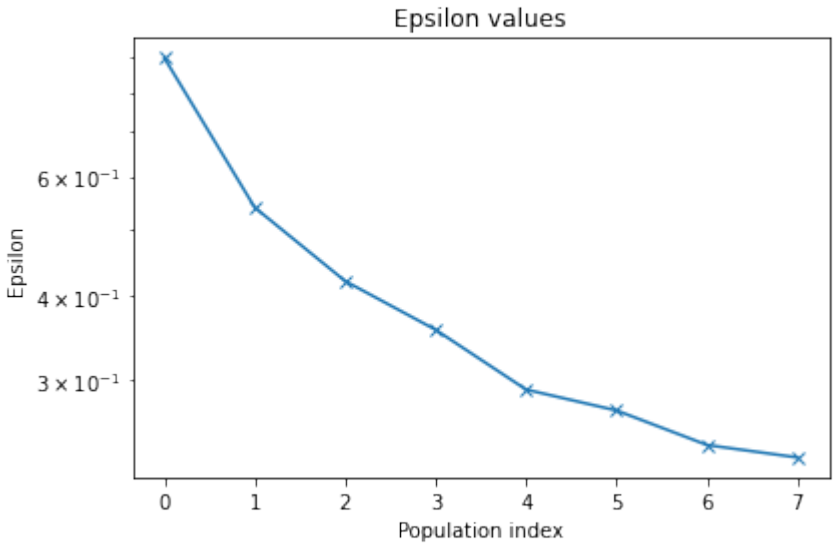
Now, every thing is prepared and we are ready to start the fittig.

[10]:
```
abc = pyabc.ABCSMC(PEtab_models, mc_PEtab_prior, eucl_dist, population_size=50)
db_path = "sqlite:///" + os.path.join(tempfile.gettempdir(), "PEtab_cell_movement.db")
mc_PEtab_history = abc.new(db_path, mc_dict_data_imported)
history = abc.run(max_nr_populations=8,minimum_epsilon=0.2)
```

```
ABC.Sampler INFO: Parallelize sampling on 12 processes.
ABC.History INFO: Start <ABCSMC id=9, start_time=2021-10-05 12:44:12>
ABC INFO: Calibration sample t = -1.
ABC INFO: t: 0, eps: 9.00000000e-01.
ABC INFO: Accepted: 50 / 106 = 4.7170e-01, ESS: 5.0000e+01.
ABC INFO: t: 1, eps: 5.40000000e-01.
ABC INFO: Accepted: 50 / 137 = 3.6496e-01, ESS: 4.7985e+01.
ABC INFO: t: 2, eps: 4.20000000e-01.
ABC INFO: Accepted: 50 / 115 = 4.3478e-01, ESS: 4.8740e+01.
ABC INFO: t: 3, eps: 3.55675552e-01.
ABC INFO: Accepted: 50 / 123 = 4.0650e-01, ESS: 4.7949e+01.
ABC INFO: t: 4, eps: 2.90000000e-01.
ABC INFO: Accepted: 50 / 270 = 1.8519e-01, ESS: 4.5600e+01.
ABC INFO: t: 5, eps: 2.70000000e-01.
ABC INFO: Accepted: 50 / 375 = 1.3333e-01, ESS: 4.7949e+01.
ABC INFO: t: 6, eps: 2.40000000e-01.
ABC INFO: Accepted: 50 / 509 = 9.8232e-02, ESS: 4.2199e+01.
ABC INFO: t: 7, eps: 2.30000000e-01.
ABC INFO: Accepted: 50 / 615 = 8.1301e-02, ESS: 4.5660e+01.
ABC INFO: Stop: Maximum number of generations.
ABC.History INFO: Done <ABCSMC id=9, duration=2:41:43.649276, end_time=2021-10-05 15:25:
↪55>
```

[11]:
```
pyabc.visualization.plot_epsilons(mc_PEtab_history)
```

[11]:
```
<AxesSubplot:title={'center':'Epsilon values'}, xlabel='Population index', ylabel=
↪'Epsilon'>
```

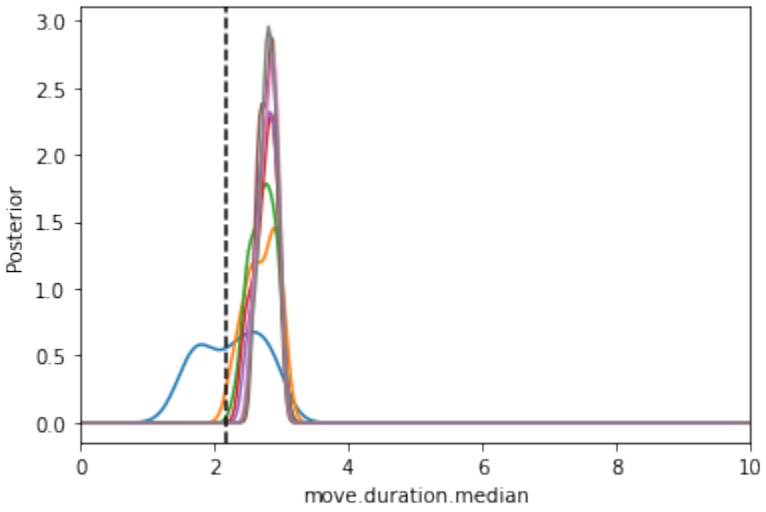


[12]:
```
# df, w = history.get_distribution(t=history.max_t)
from pyabc.visualization import plot_kde_1d
fig, ax = plt.subplots()

for t in range(mc_PEtab_history.max_t+1):
```

(continues on next page)

(continued from previous page)

```
    particles = mc_PEtab_history.get_distribution(m=0, t=t)
    plot_kde_1d(*particles, "move.duration.median",
                label="t={}".format(t), ax=ax,
                xmin=0, xmax=10, numx=300)
ax.axvline(math.log10(150), color="k", linestyle="dashed");
```



[ ]:

3.3 Application examples

3.3.1 Single cell motility

Model setup

The model definition file `Demo_CellMotility.xml` initializes a 2D hexagonal lattice of `Lx=200` and `Ly=200` with a single cell of target volume `200` $nodelength^2$ at starting position `x=100`, `y=100` and initial orientation (defined by PropertyVector dir) along the x-axis (`dir.phi=0`). From `t=0...500` with `dt=1` per Monte Carlo lattice update of the CPM, this cell performs a persistent random walk imposed by the DirectedMotion plugin with parameters Global/motion_strength and PropertyVector dir. The angle of dir is changed every dt=1 by a normally distributed random variable with `mean = 0` and `standard deviation` given by the parameter Global/noise_level. For analysis purposes, a MotilityReporter calculates finite differences (`dt=1`) of cell center coordinates along the evolving trajectory and provides a PropertyVector velocity. The Logger Plugin writes a text file with columns `time`,`cell.id`,`cell.center.x`,`cell.center.y`,`velocity.x`,`velocity.y`,`velocity.abs` and one row only every `100` time steps to emulate the coarser time resolution of image acquisition workflows. On a single CPU core this model costs approximately 0.1sec total wall clock time (plots being disabled). The two parameters Global/motion_strength and Global/noise_level control the statistics of the cell's persistent random walk.

Ground truth data For the parameter choice `Global/motion_strength=1.0` and `Global/noise_level=0.1`, `N=100` repetitions for different random number sequences have been simulated and mean square displacement (MSD) and direction autocorrelation functions (DAC) have been computed from the 100fold coarser time sampling of the simulated cell trajectories. Note, these summary statistics are not (yet) computed within Morpheus but post-hoc in Python and manually entered as np.array measured_data. For plots of these data, see the bottom of the Python notebook.

Parameter recovery study

Using FitMultiCell and adapting the workflow from MinimalExample.ipynb, the parameters are recovered from uniform priors over both parameter intervals from `0.1` to `10` fold the true values. There is some sign of parameter correlations stemming from the effective diffusion coefficient in the MSD statistics such that more random trajectories need a higher speed. To obtain MSD and DAC summary statistics, an own distance measure is computed by the FFT algorithm from the tidynamics module for each single cell trajectory (`N=1` and not an ensemble of `N=100` as for the ground truth data) with sliding time windows over the `5` temporal data points (`tau=0,100,200,300,400` over pairs of data from the trajectory sampled at `t=0,100,200,300,400,500` after discarding a burn-in interval of the CPM cell shape initialisation till `t=100`, hence the data table arguments [1:] in the Python notebook). The meta parameters are set to population_size=20 and 7 epochs.

Details of summary statistics

```
def distanceMSD(val1, val2):
    d = np.sum(np.abs(tidynamics.msd(np.column_stack([val1['IdSumstat__cell.center.x'][1:
↪],val1['IdSumstat__cell.center.y'][1:]]))\
                      - val2['IdSumstat__MSD']))
    return d
def distanceDAC(val1, val2):
    d = np.sum(np.abs(tidynamics.acf(np.column_stack([val1['IdSumstat__velocity.x'][1:]/
↪val1['IdSumstat__velocity.abs'][1:],val1['IdSumstat__velocity.y'][1:]/val1['IdSumstat__
↪velocity.abs'][1:]]))\
                      - val2['IdSumstat__DAC']))
    return d
distance = pyabc.AggregatedDistance([distanceMSD, distanceDAC])
```

[1]:
```
import pyabc
import fitmulticell as fmc
import scipy as sp
from scipy import stats
import numpy as np
import pandas as pd
import os
import tempfile
%matplotlib inline
import matplotlib.pyplot as plt
from pathlib import Path
import tidynamics                    # to get sliding history stats in N*logN instead␣
↪of N^2
```

[2]:
```
file_ = str((Path(os.getcwd())) / 'models' / 'Demo_CellMotility.xml')
par_map = {'motion_strength': './Global/Constant[@symbol="motion_strength"]',
           'noise_level': './Global/Constant[@symbol="noise_level"]'}
model = fmc.model.MorpheusModel(
    file_, par_map=par_map,
    executable="/usr/local/bin/morpheus",
    show_stdout=False, show_stderr=True,
    raise_on_error=False)
```

[3]:
```
true_pars = {'motion_strength': 1.0, 'noise_level': 0.1}
limits = {key: (0.1 * val, 10 * val) for key, val in true_pars.items()}

# generate ground truth data for recovery study
# simulated_data = model.sample(true_pars)

# experimentally measured data can be entered manually in this format
# the coarse time resolution of 100 times the simulation step size emulates␣
↪observability limitations of experiments
measured_data = {'time': np.array([0, 100, 200, 300, 400]),\
                 'MSD': np.array([0, 1640, 5330, 10400, 15800]),\
                 'DAC': np.array([1, 0.49, 0.19, 0.07, 0.07])}
```

[4]:
```
prior = pyabc.Distribution(**{key: pyabc.RV("uniform", lb, ub - lb)
                              for key, (lb, ub) in limits.items()})

# manually defined summary statistics

def distanceMSD(val1, val2):
    d = np.sum(np.abs(tidynamics.msd(np.column_stack([val1['cell.center.x'][1:],val1[
↪'cell.center.y'][1:]]))\
                      - val2['MSD']))
    return d

def distanceDAC(val1, val2):
    d = np.sum(np.abs(tidynamics.acf(np.column_stack([val1['velocity.x'][1:]/val1[
↪'velocity.abs'][1:],val1['velocity.y'][1:]/val1['velocity.abs'][1:]]))\
                      - val2['DAC']))
    return d

distance = pyabc.AggregatedDistance([distanceMSD, distanceDAC])
```

[5]:
```
abc = pyabc.ABCSMC(model, prior, distance, population_size=20)
db_path = "sqlite:///" + os.path.join(tempfile.gettempdir(), "test1.db")
history = abc.new(db_path, measured_data)
```

```
ABC.Sampler INFO: Parallelize sampling on 12 processes.
ABC.History INFO: Start <ABCSMC id=10, start_time=2022-01-20 12:36:30>
```

[6]:
```
abc.run(max_nr_populations=7)
```

```
ABC INFO: Calibration sample t = -1.
ABC INFO: t: 0, eps: 2.46221755e+04.
ABC INFO: Accepted: 20 / 58 = 3.4483e-01, ESS: 2.0000e+01.
ABC INFO: t: 1, eps: 2.28795148e+04.
ABC INFO: Accepted: 20 / 56 = 3.5714e-01, ESS: 1.8279e+01.
ABC INFO: t: 2, eps: 2.11752622e+04.
ABC INFO: Accepted: 20 / 72 = 2.7778e-01, ESS: 1.5953e+01.
ABC INFO: t: 3, eps: 1.96163649e+04.
ABC INFO: Accepted: 20 / 131 = 1.5267e-01, ESS: 1.2051e+01.
ABC INFO: t: 4, eps: 1.78708199e+04.
ABC INFO: Accepted: 20 / 360 = 5.5556e-02, ESS: 1.8161e+01.
ABC INFO: t: 5, eps: 1.69895975e+04.
```

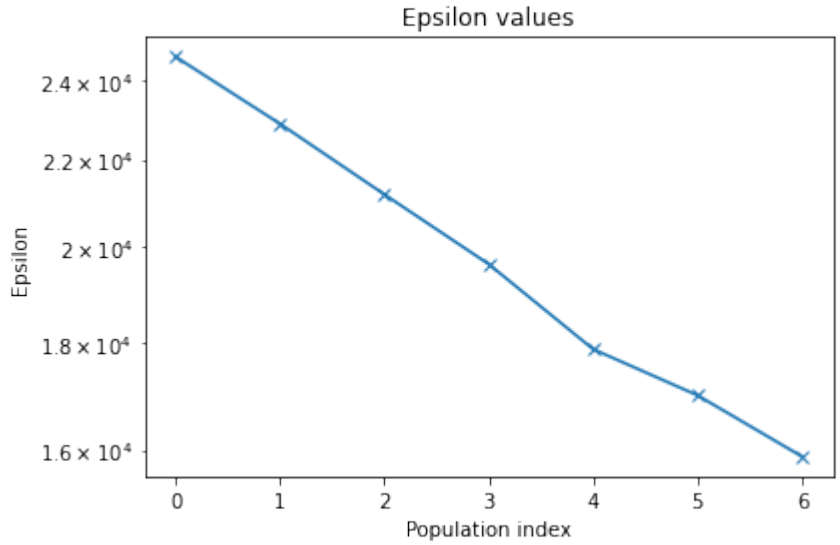
(continues on next page)

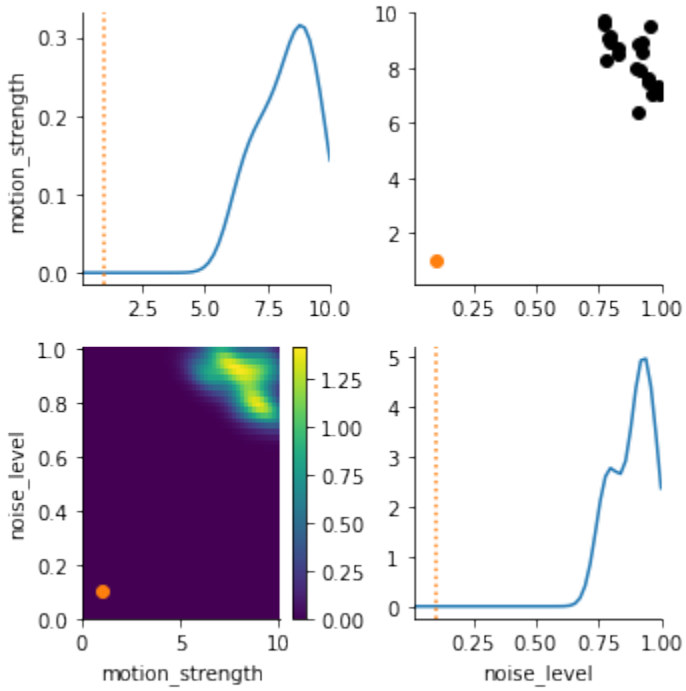
(continued from previous page)

```
ABC INFO: Accepted: 20 / 485 = 4.1237e-02, ESS: 1.4763e+01.
ABC INFO: t: 6, eps: 1.58884327e+04.
ABC INFO: Accepted: 20 / 795 = 2.5157e-02, ESS: 1.6402e+01.
ABC INFO: Stop: Maximum number of generations.
ABC.History INFO: Done <ABCSMC id=10, duration=0:00:24.166020, end_time=2022-01-20 12:36:
↪55>
```

[6]:
```
<pyabc.storage.history.History at 0x7ff8ad297a10>
```

[7]:
```
h = pyabc.History(db_path)
pyabc.visualization.plot_epsilons(h)
df, w = h.get_distribution(t=h.max_t)
pyabc.visualization.plot_kde_matrix(df, w, limits=limits, refval=true_pars)
plt.show()
```





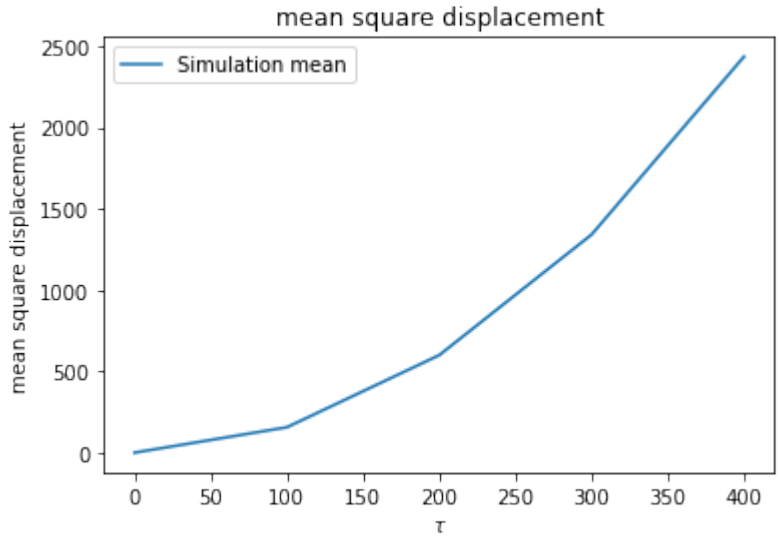
[8]:
```
# summary statistics for N repetitions of the model simulation and plotting

fit_pars = {'motion_strength': 1.0, 'noise_level': 0.1}
N = 100 # runs to average over, make sure random_seed is off in Morpheus
time = np.arange(5)
meantival1 = np.zeros(5)
meantival2 = np.zeros(5)
semtival1 = np.zeros(5)
semtival2 = np.zeros(5)
count = 0

for i in range(N):
    simulated_data = model.sample(fit_pars)
    val1vec = np.column_stack([simulated_data['cell.center.x'][1:],simulated_data['cell.
↪center.y'][1:]])
    meantival1 += tidynamics.msd(val1vec)
    semtival1 += (tidynamics.msd(val1vec))**2
    val2vec = np.column_stack([simulated_data['velocity.x'][1:]/simulated_data['velocity.
↪abs'][1:],simulated_data['velocity.y'][1:]/simulated_data['velocity.abs'][1:]])
    meantival2 += tidynamics.acf(val2vec)
    semtival2 += (tidynamics.acf(val2vec))**2
    count += 1

meantival1 /= count
semtival1 = np.sqrt(((semtival1 / count) - (meantival1)**2) / count)
meantival2 /= count
semtival2 = np.sqrt(((semtival2 / count) - (meantival2)**2) / count)

print('mean MSD = ',meantival1)
```

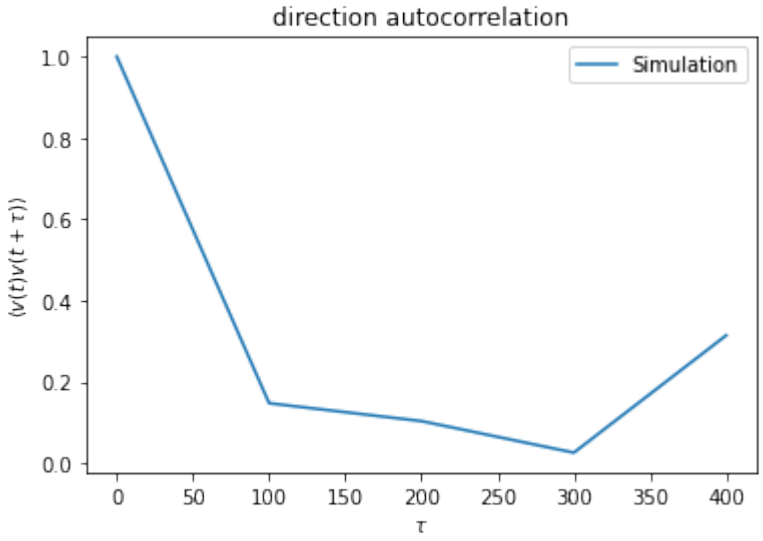
(continues on next page)

(continued from previous page)

```
print( 'sem MSD = ',semtival1)
print('mean DAC = ',meantival2)
print( 'sem DAC = ',semtival2)

# plot MSD
plt.plot(measured_data['time'], meantival1, label='Simulation mean')
plt.legend()
plt.title('mean square displacement')
plt.xlabel(r'$\tau$')
plt.ylabel(r'mean square displacement')
plt.show()

# plot DAC
plt.plot(measured_data['time'], meantival2, label='Simulation')
plt.legend()
plt.title('direction autocorrelation')
plt.xlabel(r'$\tau$')
plt.ylabel(r'$\langle v(t) v(t+\tau) \rangle$')
plt.show()
```

```
mean MSD =  [-2.56113708e-11  1.55992635e+02  5.99555390e+02  1.34131782e+03
  2.43575981e+03]
sem MSD =  [2.88935230e-12 2.22951285e+00 1.00462531e+01 2.27205059e+01
 3.43561491e+01]
mean DAC =  [1.         0.14711839 0.10313133 0.02550863 0.31417727]
sem DAC =  [6.14390615e-09 3.53107323e-02 5.32866381e-02 6.12744495e-02
 5.66760041e-02]
```





[ ]:

3.3.2 Classification

In this notebook, we will try to classify the data in the logger file based on a classification value. To do so, we will use a function called `classify_based_on_value`. This function takes file containing the Morpheus log output, the field of interest that we want to classify, the value of classification, the time step, and the time symbol, as specified in the Morpheus model. The output will be a dictionary with cell type as a key, and the list of occurrences of the cell types as a value for different time intervals. The result will be classified to less or larger than the `value_of_interest`.

Let try to use the function now, but before we start, let's define some required parameters. First, let's create a sample logger file in a CSV format:

[1]:
```
import csv
import tempfile
import os

full_path=os.path.join(tempfile.gettempdir(), "logger.csv")

csvfile=open(full_path,'w', newline='')
obj=csv.writer(csvfile, delimiter='\t')
obj.writerow(['t', 'cell.id', 'RNA_concentration', 'cell_type'])
obj.writerow(['0', '1', '0.1' ,'0'])
obj.writerow(['0', '2', '0.2' ,'0'])
obj.writerow(['0', '3', '0.3' ,'0'])
obj.writerow(['1', '4', '0.4' ,'1'])
obj.writerow(['1', '5', '0.5' ,'2'])
obj.writerow(['1', '6', '0.6' ,'1'])
obj.writerow(['2', '7', '0.7' ,'2'])
obj.writerow(['2', '8', '0.8' ,'3'])
obj.writerow(['2', '9', '0.9' ,'2'])
csvfile.close()
```

As we can see from the above example, the time step is =1 (if not specify by user, the value will be extracted from

Morpheus file. Moreover, the time symbol that was used in the model is =t. The time symbol by default is 't'. So, even if we didn't specify the time_step and time_symbol here, the code will run fine.

[2]:
```
t_step=1
t_symbol='t'
```

We want to classify the result based on the RNA_concentration. We will use value_of_interest to be = 0.5. The field_of_interest will be =RNA_concentration. Fianlly, the value_of_interest = 0.5

[3]:
```
field_of_interest = 'RNA_concentration'
value_of_interest = 0.5
```

Now, we are ready to call the function classify_based_on_value(), but before that, let's import the function's package. Don't forget to install the package first.

[ ]:
```
from fitmulticell.sumstat import cell_types_cout as ss
```

We also need to import util module form fitmulticell to read the CSV file as pandas df

[4]:
```
import fitmulticell.util as util
```

No, we will read the CSV file using the "tsv_to_df" function form the fitmulticell library

[5]:
```
logger_file = util.tsv_to_df("/tmp")
```

Let's see how the logger_file loks like

[6]:
```
logger_file
```

[6]:

	t	cell.id	RNA_concentration	cell_type
0	0	1	0.1	0
1	0	2	0.2	0
2	0	3	0.3	0
3	1	4	0.4	1
4	1	5	0.5	2
5	1	6	0.6	1
6	2	7	0.7	2
7	2	8	0.8	3
8	2	9	0.9	2

[8]:
```
classification_result=ss.classify_based_on_value(logger_file, field_of_interest, value_
↪of_interest)
```

[9]:
```
print(f'The classification result is = {classification_result}')
```

```
The classification result is = {0: [3, 0], 1: [1, 2], 2: [0, 3]}
```

As we can see from the above result, the RNA_concentration of the first cell type "0" has three occurrences that are less than the value_of_interest and none that are larger or equal to it. Whereas the second cell type "1", has only one occurrence that is less than the value_of_interest and two that are larger or equal to value_of_interest.

3.4 Summary statistics

3.4.1 Cluster Count

In this notebook, we will try to extract further information from Morpheus logger file. We will try to count the number of cluster in a hexagonal grid at different time points. To do so, we will use a function called `get_clusters_count()`.

The function takes the Morpheus log output as pd.df, the field of interest that contains clusters data, a list of cell type that are member of the cluster formation, the time step between time points, the time interval where we want to count the number of clusters (optional), the time symbol (optional) as specified in the Morpheus model. The output will be the total number of clusters for a specific time point.

Let try to use the function now, but before we start, let's define some required parameters. First, let's create a simple logger file in a CSV format:

[11]:
```
import csv
import tempfile
import os

full_path=os.path.join(tempfile.gettempdir(), "logger.csv")

csvfile=open(full_path,'w', newline='')
obj=csv.writer(csvfile, delimiter='\t')
obj.writerow(['t', 'cell.id', 'RNA_concentration', 'infection'])
obj.writerow(['0', '1', '0.1' ,'1'])
obj.writerow(['0', '2', '0.2' ,'0'])
obj.writerow(['0', '3', '0.3' ,'1'])
obj.writerow(['0', '4', '0.4' ,'1'])
obj.writerow(['0', '5', '0.5' ,'1'])
obj.writerow(['0', '6', '0.6' ,'0'])
obj.writerow(['0', '7', '0.7' ,'0'])
obj.writerow(['0', '8', '0.8' ,'0'])
obj.writerow(['0', '9', '0.9' ,'0'])
obj.writerow(['0', '10', '0.9' ,'0'])
obj.writerow(['0', '11', '0.9' ,'1'])
obj.writerow(['0', '12', '0.9' ,'1'])
obj.writerow(['0', '13', '0.9' ,'1'])
obj.writerow(['0', '14', '0.9' ,'0'])
obj.writerow(['0', '15', '0.9' ,'0'])
obj.writerow(['0', '16', '0.9' ,'0'])
obj.writerow(['0', '17', '0.9' ,'0'])
obj.writerow(['0', '18', '0.9' ,'0'])
obj.writerow(['0', '19', '0.9' ,'1'])

obj.writerow(['1', '1', '0.1' ,'1'])
obj.writerow(['1', '2', '0.2' ,'1'])
obj.writerow(['1', '3', '0.3' ,'1'])
obj.writerow(['1', '4', '0.4' ,'1'])
obj.writerow(['1', '5', '0.5' ,'1'])
obj.writerow(['1', '6', '0.6' ,'1'])
obj.writerow(['1', '7', '0.7' ,'0'])
obj.writerow(['1', '8', '0.8' ,'0'])
obj.writerow(['1', '9', '0.9' ,'0'])
```

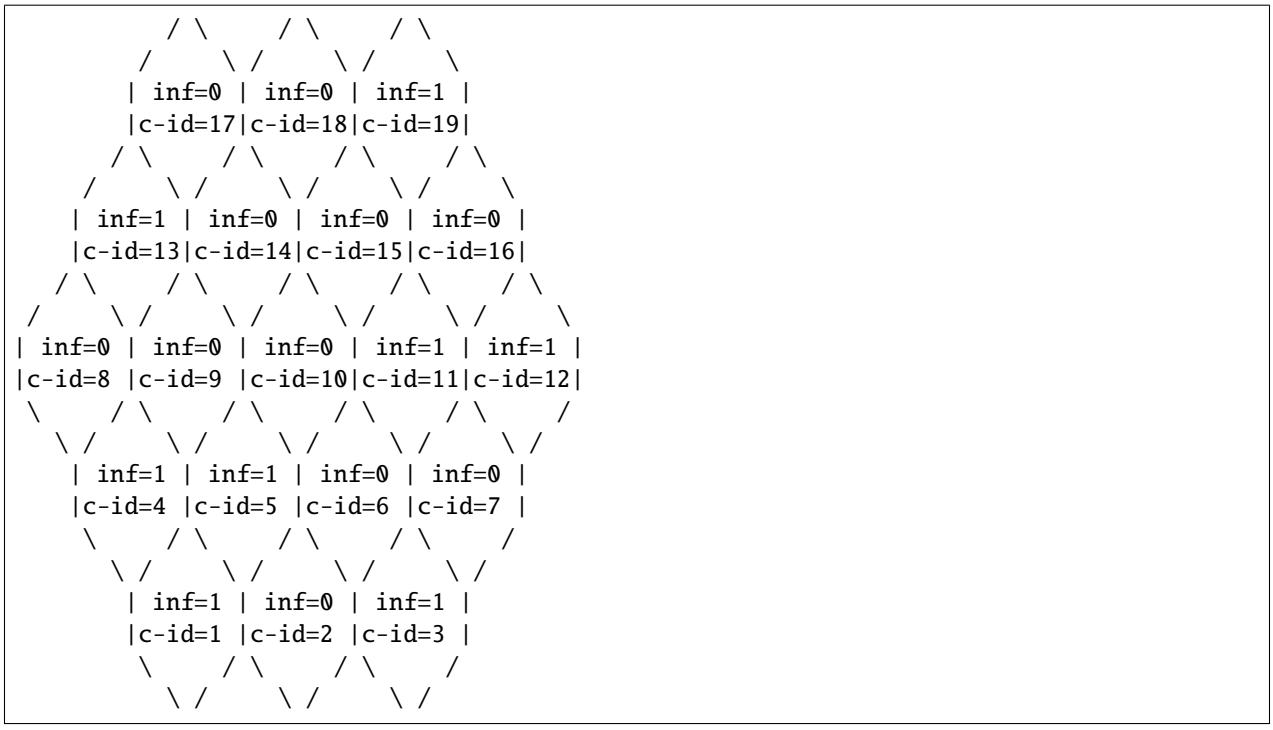
(continues on next page)

(continued from previous page)

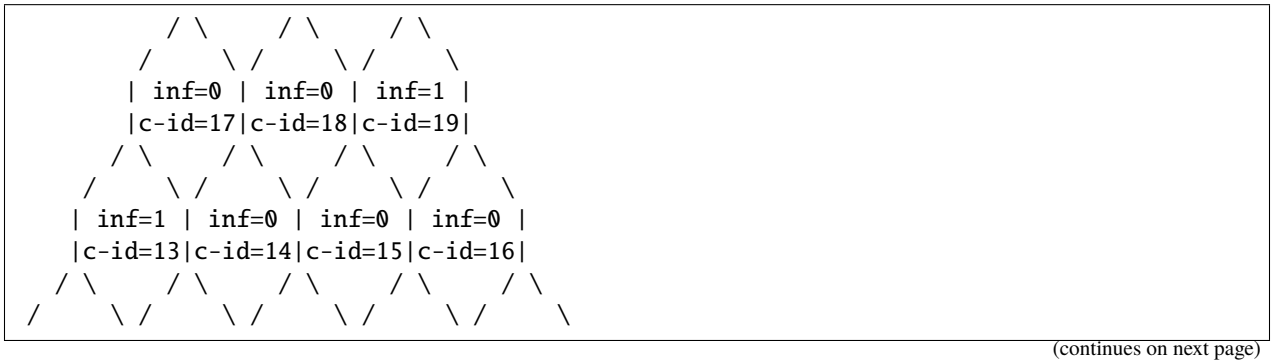
```
obj.writerow(['1', '10', '0.9' ,'0'])
obj.writerow(['1', '11', '0.9' ,'1'])
obj.writerow(['1', '12', '0.9' ,'0'])
obj.writerow(['1', '13', '0.9' ,'1'])
obj.writerow(['1', '14', '0.9' ,'0'])
obj.writerow(['1', '15', '0.9' ,'0'])
obj.writerow(['1', '16', '0.9' ,'0'])
obj.writerow(['1', '17', '0.9' ,'0'])
obj.writerow(['1', '18', '0.9' ,'0'])
obj.writerow(['1', '19', '0.9' ,'1'])


csvfile.close()
```

the hexagonal grid we look like this:

Time_step=0:



Time_step=1:



(continues on next page)

(continued from previous page)

```
| inf=0 | inf=0 | inf=0 | inf=1 | inf=1 |
|c-id=8 |c-id=9 |c-id=10|c-id=11|c-id=12|
 \     / \     / \     / \     / \     /
   \ /     \ /     \ /     \ /     \ /
    | inf=1 | inf=1 | inf=1 | inf=0 |
    |c-id=4 |c-id=5 |c-id=6 |c-id=7 |
     \     / \     / \     / \     /
       \ /     \ /     \ /     \ /
        | inf=1 | inf=1 | inf=1 |
        |c-id=1 |c-id=2 |c-id=3 |
         \     / \     / \     /
           \ /     \ /     \ /
```

We need to define the field of interest

[15]:
```
field_of_interest='infection'
```

We also need to define the list of the cell types that are members of the cluster formation. In this simple example, we consider a cell to be infected only if it has a value of 1.

[16]:
```
cluster_cell_types=[1]
```

Finally, we will specify the `time_symbol`, as specified in the Morpheus model.

[4]:
```
t_symbol='t'
```

After defining all the required parameter, let's run the function. But before that, let's import the function's library. Don't forget to install the package first.

[17]:
```
from fitmulticell.sumstat import hexagonal_cluster_sumstat as css
```

We also need to import external library form pyABC to read the CSV file as pandas df

[18]:
```
import fitmulticell.util as util
```

No, we will read the CSV file using the "read_morpheus_log_file" function form the external library

[19]:
```
logger_file = util.tsv_to_df("/tmp")
```

Let's see how the logger_file loks like

[20]:
```
logger_file
```

[20]:

	t	cell.id	RNA_concentration	infection
0	0	1	0.1	1
1	0	2	0.2	0
2	0	3	0.3	1
3	0	4	0.4	1
4	0	5	0.5	1
5	0	6	0.6	0
6	0	7	0.7	0
7	0	8	0.8	0
8	0	9	0.9	0
9	0	10	0.9	0

(continues on next page)

(continued from previous page)

```
10  0       11                0.9          1
11  0       12                0.9          1
12  0       13                0.9          1
13  0       14                0.9          0
14  0       15                0.9          0
15  0       16                0.9          0
16  0       17                0.9          0
17  0       18                0.9          0
18  0       19                0.9          1
19  1        1                0.1          1
20  1        2                0.2          1
21  1        3                0.3          1
22  1        4                0.4          1
23  1        5                0.5          1
24  1        6                0.6          1
25  1        7                0.7          0
26  1        8                0.8          0
27  1        9                0.9          0
28  1       10                0.9          0
29  1       11                0.9          1
30  1       12                0.9          0
31  1       13                0.9          1
32  1       14                0.9          0
33  1       15                0.9          0
34  1       16                0.9          0
35  1       17                0.9          0
36  1       18                0.9          0
37  1       19                0.9          1
```

[23]:
```
cluster_count_result=css.get_clusters_count(logger_file, field_of_interest,cluster_cell_
↪types,time_symbol=t_symbol)
print(f'the total number of cluster = {cluster_count_result}')
```

```
the total number of cluster = {0: 5, 1: 3}
```

We can also use the function with a specific time interval:

[24]:
```
cluster_count_result=css.get_clusters_count(logger_file, field_of_interest,cluster_cell_
↪types,time_interval=[0,1],time_symbol=t_symbol)
print(f'the total number of cluster = {cluster_count_result}')
```

```
the total number of cluster = {0: 5, 1: 3}
```

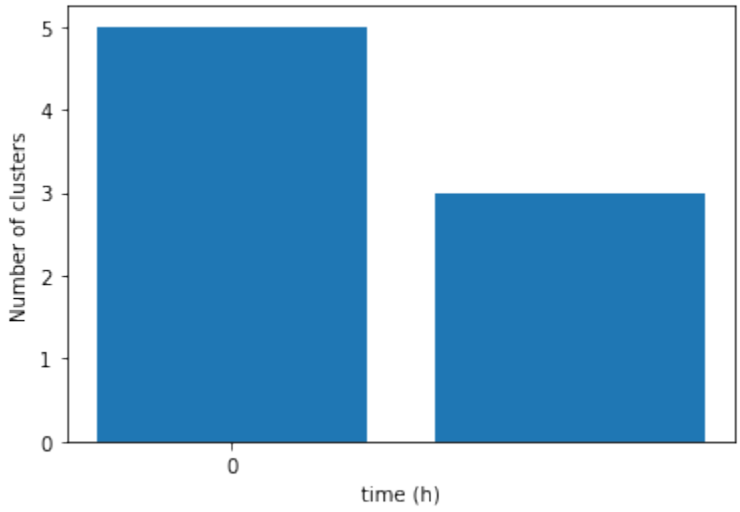
As stated in the output above, at `time_point` = 0 we had 5 clusters, but at `time_point` = 1 we had only 3 clusters.

We can plot the above result by using a function called `plot_cluster_count_all_time_point()`.

This function will takes the dictionary output of the `get_clusters_count_alltp()`. But before using the function, let's import its library:

[27]:
```
from fitmulticell.sumstat import plot_sumstat as pss
```

[29]:
```
ax = pss.plot_cluster_count_all_time_point(cluster_count_result)
```



3.4.2 Cluster size

In this notebook, we will see how we can find out the size of different clusters at different single time point.

To do so, we will use a function called `get_clusters_size_tp()`. This function will take the Morpheus log file as pd.df, the field of interest that contains the clusters data, a list of cell types that are member of the cluster, the time point where we want count the number of clusters, and the time symbol, as specified in the Morpheus model. The output will be a dictionary that contains the index and the member of each cluster.

Let try to use the function now, but before we start, let's define some required parameters. First, let's create a simple logger file in a CSV format:

[1]:
```
import csv
import tempfile
import os

full_path=os.path.join(tempfile.gettempdir(), "logger.csv")

csvfile=open(full_path,'w', newline='')
obj=csv.writer(csvfile, delimiter='\t')
obj.writerow(['t', 'cell.id', 'RNA_concentration', 'infection'])
obj.writerow(['0', '1', '0.1' ,'1'])
obj.writerow(['0', '2', '0.2' ,'0'])
obj.writerow(['0', '3', '0.3' ,'1'])
obj.writerow(['0', '4', '0.4' ,'1'])
obj.writerow(['0', '5', '0.5' ,'1'])
obj.writerow(['0', '6', '0.6' ,'0'])
obj.writerow(['0', '7', '0.7' ,'0'])
obj.writerow(['0', '8', '0.8' ,'0'])
obj.writerow(['0', '9', '0.9' ,'0'])
obj.writerow(['0', '10', '0.9' ,'0'])
obj.writerow(['0', '11', '0.9' ,'1'])
obj.writerow(['0', '12', '0.9' ,'1'])
obj.writerow(['0', '13', '0.9' ,'1'])
```

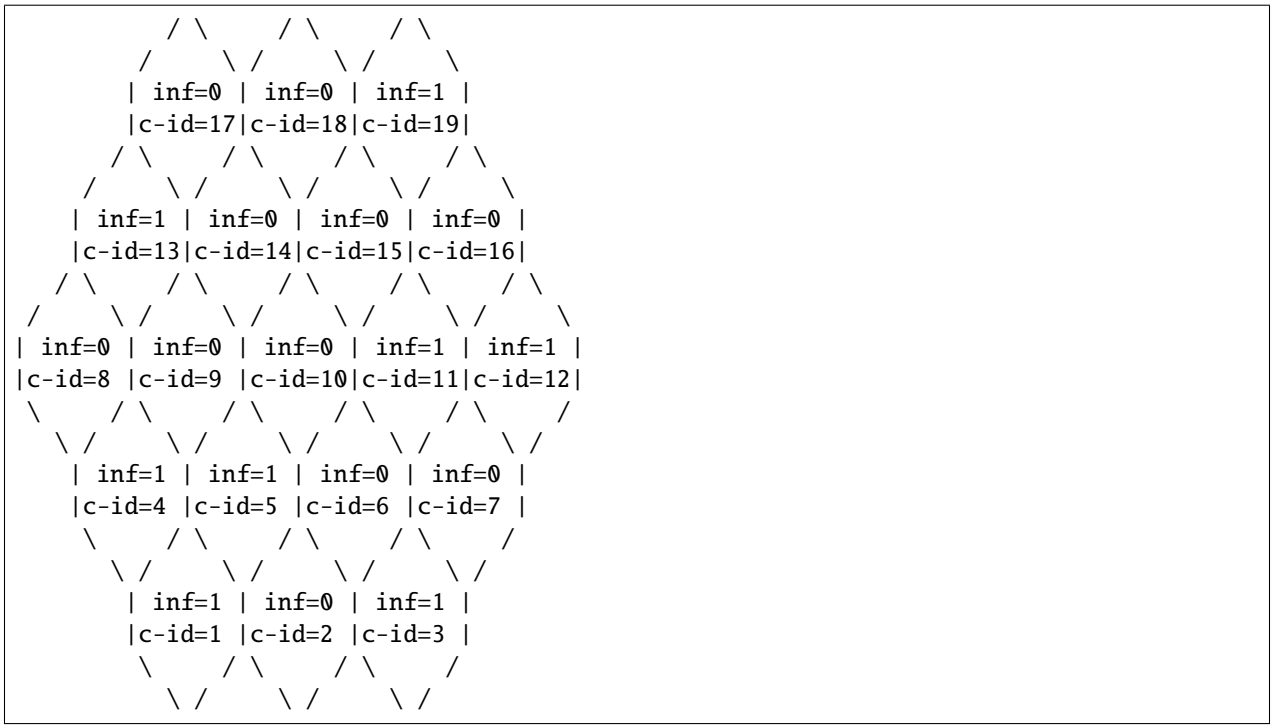
(continues on next page)

(continued from previous page)

```
obj.writerow(['0', '14', '0.9' ,'0'])
obj.writerow(['0', '15', '0.9' ,'0'])
obj.writerow(['0', '16', '0.9' ,'0'])
obj.writerow(['0', '17', '0.9' ,'0'])
obj.writerow(['0', '18', '0.9' ,'0'])
obj.writerow(['0', '19', '0.9' ,'1'])

csvfile.close()
```

The data above describe a hexagonal grid with two types of cell, infected or not infected. The infected cell will have `infection` value = 1.

the hexagonal grid we look like this:



where inf represents the infection and c-id represent the cell ID.

So, let's define the required parameters and run the function. The field of interest will be `infection`:

[2]:
```
field_of_interest='infection'
```

We also need to define the list of cell types that are members of the cluster formation. In this simple example, we consider a cell to be infected only if it has a value of 1.

[3]:
```
cluster_cell_types=[1]
```

Finally, we will specify the `time_point` and the `time_symbol`, both as specified in the Morpheus model.

[4]:
```
time_point=0
t_symbol='t'
```

After defining all the required parameter, let's run the function. But before that, let's import the function's library. Don't forget to install the package first.

[6]:
```
import fitmulticell.sumstat.hexagonal_cluster_sumstat as css
```

We also need to import `util` library form fitmulticell to read the CSV file as pandas df

[16]:
```
import fitmulticell.util as util
```

No, we will read the CSV file using the `tsv_to_df` function form the fitmulticell package

[17]:
```
logger_file = util.tsv_to_df("/tmp")
```

Let's see how the logger_file loks like

[18]:
```
logger_file
```

[18]:

	t	cell.id	RNA_concentration	infection
0	0	1	0.1	1
1	0	2	0.2	0
2	0	3	0.3	1
3	0	4	0.4	1
4	0	5	0.5	1
5	0	6	0.6	0
6	0	7	0.7	0
7	0	8	0.8	0
8	0	9	0.9	0
9	0	10	0.9	0
10	0	11	0.9	1
11	0	12	0.9	1
12	0	13	0.9	1
13	0	14	0.9	0
14	0	15	0.9	0
15	0	16	0.9	0
16	0	17	0.9	0
17	0	18	0.9	0
18	0	19	0.9	1

[19]:
```
cluster_size_result=css.get_clusters_sizes_tp(logger_file, field_of_interest,cluster_
↪cell_types,time_point,time_symbol=t_symbol)
print(f'The size of clusters in the specific time point = {time_point} is = {cluster_
↪size_result}')
```

```
The size of clusters in the specific time point = 0 is = {1: 3, 3: 1, 11: 2, 13: 1, 19:␣
↪1}
```

As the result shows, the dictionary key specifies the smallest cell ID in the cluster to act as an index for the cluster, and the value (a list data structure) specify the cell ID for the member cells.

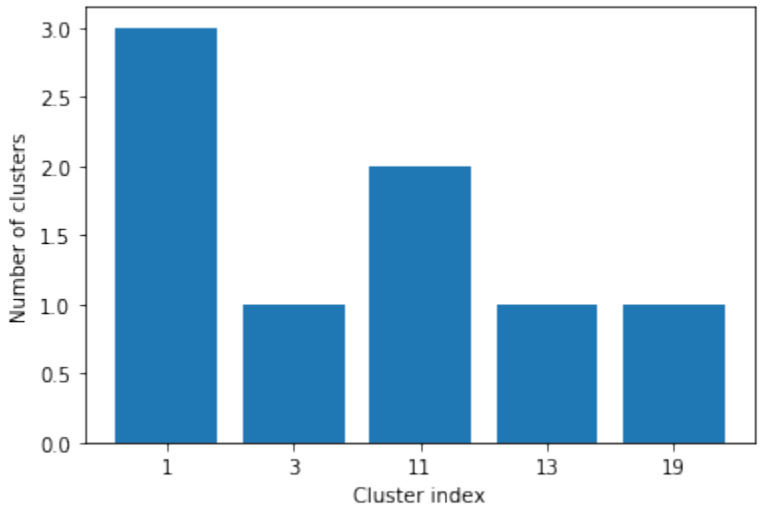
We can plot this result using a function called `plot_cluster_size_all_time_point()`. This function take the ouput of the `get_clusters_size_tp()` as input.

But before using the function, let's import its library:

[14]:
```
import fitMultiCellSumStat.plot_sumstat as pss
```

[20]:
```
ax = pss.plot_cluster_size_all_time_point(cluster_size_result)
```

```
INFO:matplotlib.category:Using categorical units to plot a list of strings that are all␣
↪parsable as floats or dates. If these strings should be plotted as numbers, cast to␣
↪the appropriate data type before plotting.
INFO:matplotlib.category:Using categorical units to plot a list of strings that are all␣
↪parsable as floats or dates. If these strings should be plotted as numbers, cast to␣
↪the appropriate data type before plotting.
INFO:matplotlib.category:Using categorical units to plot a list of strings that are all␣
↪parsable as floats or dates. If these strings should be plotted as numbers, cast to␣
↪the appropriate data type before plotting.
```



3.4.3 Count Active infected Cell

In this notebook, we will try to count the number of CC contributor in hexagonal grid.

The CC contributor cells are those who contribute to cell-to-cell transmission.

To do so, we will use a function called `get_count_cc_contributors_alltp`. This function takes the Morpheus log output as pd.df, the field of interest that contain the cell information, the list of cell types that are member of the cluster, the time interval of which we want to count the CC contributor cells (optional), the time step (optional), and the time symbol, as specified in the Morpheus model. The output will be a dictionary with time points as a keys, and the count of the CC contributor cells as values.

Let try to use the function now, but before we start, let's define some required parameters. First, let's create a sample logger file in a CSV format:

[14]:
```
import csv
import tempfile
import os

full_path=os.path.join(tempfile.gettempdir(), "logger.csv")

csvfile=open(full_path,'w', newline='')
obj=csv.writer(csvfile, delimiter='\t')
obj.writerow(['t', 'cell.id', 'RNA_concentration', 'infection'])
obj.writerow(['0', '1', '0.1' ,'1'])
obj.writerow(['0', '2', '0.2' ,'1'])
```

(continues on next page)

(continued from previous page)

```
obj.writerow(['0', '3', '0.3' ,'0'])
obj.writerow(['0', '4', '0.4' ,'1'])
obj.writerow(['0', '5', '0.5' ,'1'])
obj.writerow(['0', '6', '0.6' ,'0'])
obj.writerow(['0', '7', '0.7' ,'0'])
obj.writerow(['0', '8', '0.8' ,'0'])
obj.writerow(['0', '9', '0.9' ,'0'])
obj.writerow(['0', '10', '0.9' ,'0'])
obj.writerow(['0', '11', '0.9' ,'0'])
obj.writerow(['0', '12', '0.9' ,'0'])
obj.writerow(['0', '13', '0.9' ,'0'])
obj.writerow(['0', '14', '0.9' ,'0'])
obj.writerow(['0', '15', '0.9' ,'0'])
obj.writerow(['0', '16', '0.9' ,'0'])
obj.writerow(['0', '17', '0.9' ,'1'])
obj.writerow(['0', '18', '0.9' ,'0'])
obj.writerow(['0', '19', '0.9' ,'1'])

obj.writerow(['1', '1', '0.1' ,'1'])
obj.writerow(['1', '2', '0.2' ,'1'])
obj.writerow(['1', '3', '0.3' ,'0'])
obj.writerow(['1', '4', '0.4' ,'1'])
obj.writerow(['1', '5', '0.5' ,'1'])
obj.writerow(['1', '6', '0.6' ,'0'])
obj.writerow(['1', '7', '0.7' ,'0'])
obj.writerow(['1', '8', '0.8' ,'1'])
obj.writerow(['1', '9', '0.9' ,'1'])
obj.writerow(['1', '10', '0.9' ,'0'])
obj.writerow(['1', '11', '0.9' ,'0'])
obj.writerow(['1', '12', '0.9' ,'0'])
obj.writerow(['1', '13', '0.9' ,'1'])
obj.writerow(['1', '14', '0.9' ,'0'])
obj.writerow(['1', '15', '0.9' ,'1'])
obj.writerow(['1', '16', '0.9' ,'1'])
obj.writerow(['1', '17', '0.9' ,'1'])
obj.writerow(['1', '18', '0.9' ,'0'])
obj.writerow(['1', '19', '0.9' ,'1'])


csvfile.close()
```

The data above describe a hexagonal grid with two types of cell, infected or not infected. The infected cell will have `infection` value = 1. The data describe the grid in two time steps (0 and 1).

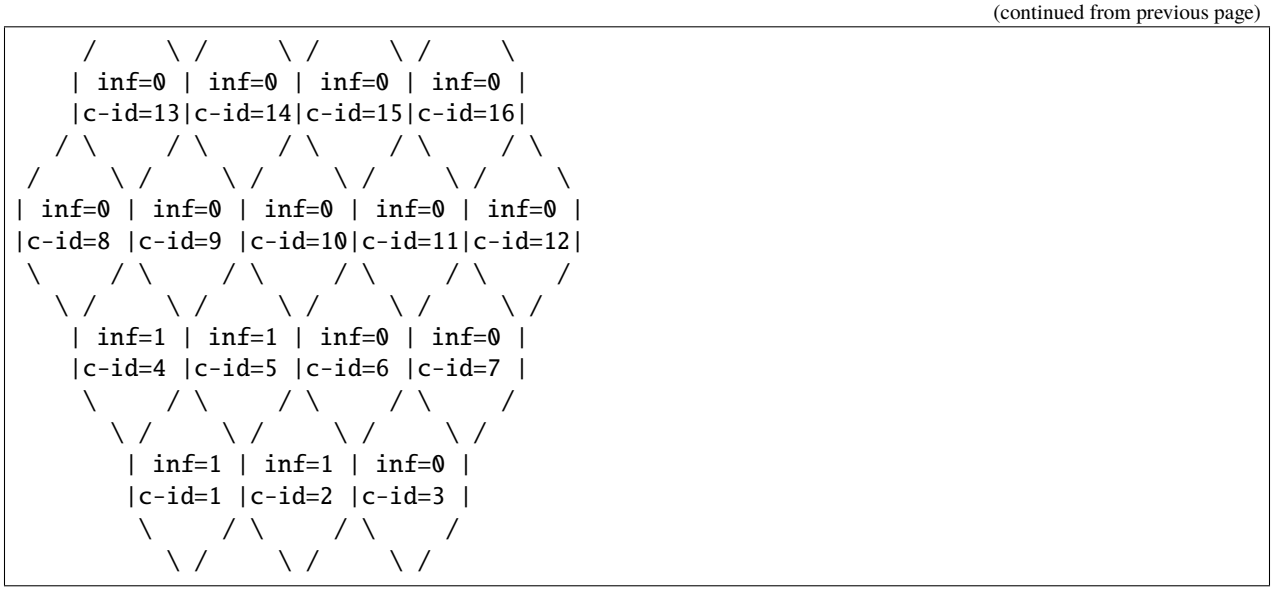
the hexagonal grid we look like this:

Time_step=0:

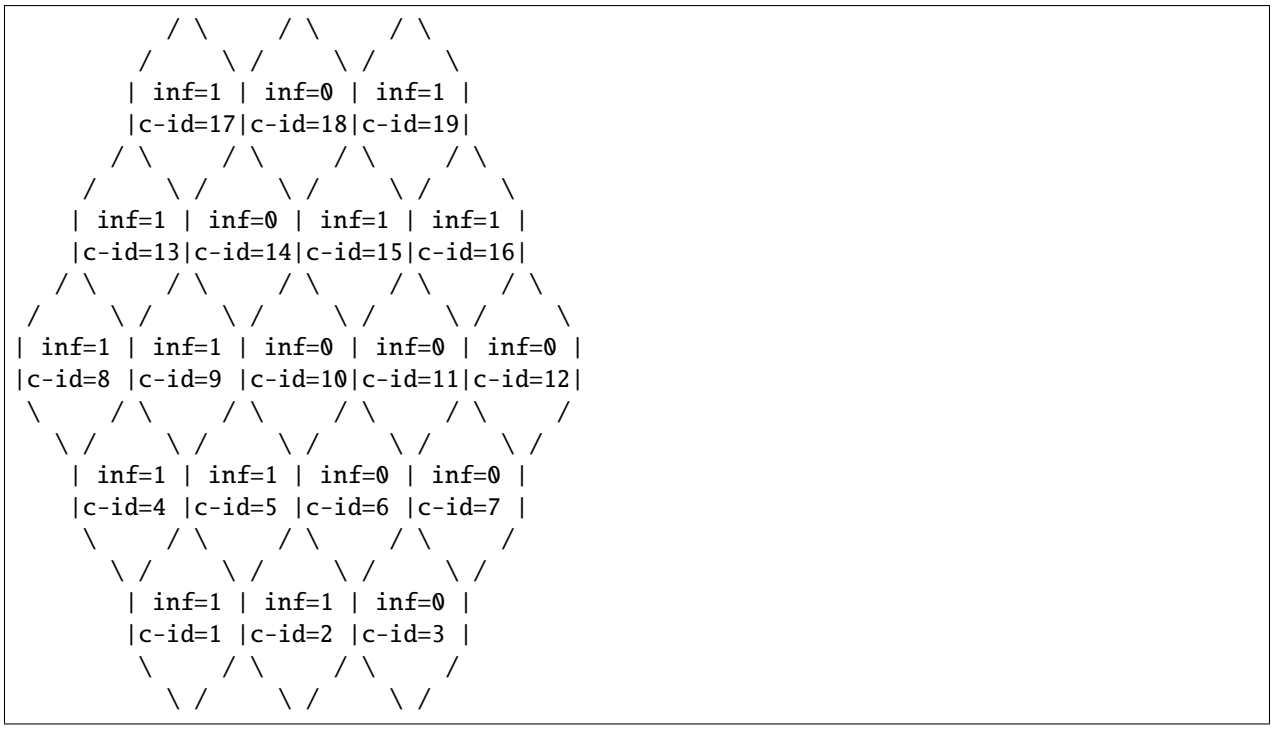
```
          / \     / \     / \
        /     \ /     \ /     \
       | inf=1 | inf=0 | inf=1 |
       |c-id=17|c-id=18|c-id=19|
      / \     / \     / \     / \
```

(continues on next page)

(continued from previous page)



Time_step=1:



where inf represents the infection and c-id represent the cell ID.

So, let's define the required parameters and run the function. The field of interest will be `infection`:

[15]:
```
field_of_interest='infection'
```

We also need to define the list of the cell types that are members of the cluster formation. In this simple example, we consider a cell to be infected only if it has a value of 1.

[16]:
```
cluster_cell_types=[1]
```

Finally, we will specify the `time_interval`, and the `time_symbol`, both as specified in the Morpheus model.

[17]:
```
t_interval=[0,1]
t_symbol='t'
```

After defining all the required parameter, let's run the function. But before that, let's import the function's library. Don't forget to install the package first.

[18]:
```
import fitmulticell.sumstat.hexagonal_cluster_sumstat as css
```

We also need to import `util` library form fitmulticell to read the CSV file as pandas df

[19]:
```
import fitmulticell.util as util
```

No, we will read the CSV file using the `tsv_to_df` function form the external library

[20]:
```
logger_file = util.tsv_to_df("/tmp")
```

Let's see how the logger_file loks like

[21]:
```
logger_file
```

[21]:

	t	cell.id	RNA_concentration	infection
0	0	1	0.1	1
1	0	2	0.2	1
2	0	3	0.3	0
3	0	4	0.4	1
4	0	5	0.5	1
5	0	6	0.6	0
6	0	7	0.7	0
7	0	8	0.8	0
8	0	9	0.9	0
9	0	10	0.9	0
10	0	11	0.9	0
11	0	12	0.9	0
12	0	13	0.9	0
13	0	14	0.9	0
14	0	15	0.9	0
15	0	16	0.9	0
16	0	17	0.9	1
17	0	18	0.9	0
18	0	19	0.9	1
19	1	1	0.1	1
20	1	2	0.2	1
21	1	3	0.3	0
22	1	4	0.4	1
23	1	5	0.5	1
24	1	6	0.6	0
25	1	7	0.7	0
26	1	8	0.8	1
27	1	9	0.9	1
28	1	10	0.9	0
29	1	11	0.9	0
30	1	12	0.9	0
31	1	13	0.9	1

(continues on next page)

(continued from previous page)

```
32  1       14              0.9         0
33  1       15              0.9         1
34  1       16              0.9         1
35  1       17              0.9         1
36  1       18              0.9         0
37  1       19              0.9         1
```

[22]:
```
CC_Contributor_count_result=css.get_count_cc_contributors_alltp(logger_file, field_of_
↪interest,cluster_cell_types,time_interval=t_interval,time_symbol=t_symbol)
print(f'the total number of cluster = {CC_Contributor_count_result}')
```

```
the total number of cluster = {0: 5, 1: 8}
```

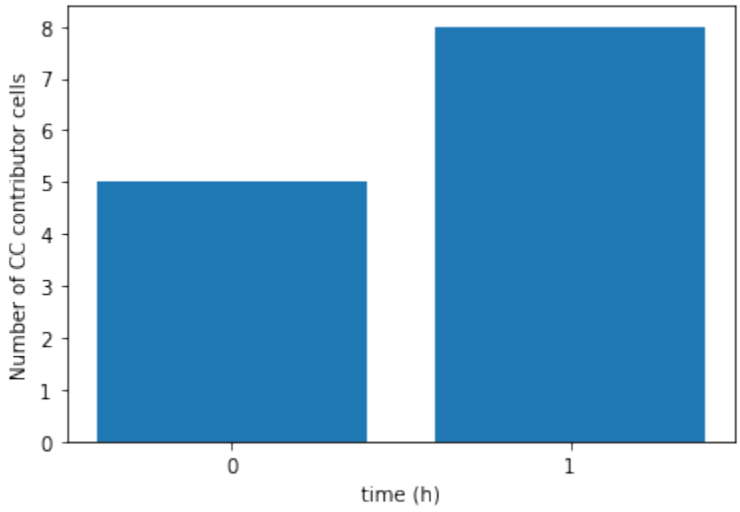
As stated in the output above, at `time_point` = 0 we had 5 CC contributor cells, but at `time_point` = 1 we had 8.

We can plot the above result by using a function called `plot_active_cell_all_time_point()`.

This function will takes the dictionary output of the `get_count_cc_contributors_tp()`. But before using the function, let's import its library:

[23]:
```
import fitMultiCellSumStat.plot_sumstat as pss
```

[24]:
```
ax = pss.plot_active_cell_all_time_point(CC_Contributor_count_result)
```



3.4.4 Count Cell types

In this notebook, we will see how we can collect summary statistics (SS) from Morpheus logger file. The SS that we are interested in will be to count the occurrence of different cell types.

To do so, we will use the function `count_cell_types()`. This function takes `morpheus_log_file` as a parameter and return the number of occurrences of each cell types. The function also asks for time step between the time points. This should be the same as specified in the logger file. Also, the `field_of_interest` should be described. This field should contain the necessary information about different cell types. In addition, the `cell_types_list` should be specified, which is a list that contains the cell types that we are interested in. Finally, specifying the time symbol used in the Morpheus model (default value will be 't').

Let's first start by using a simple example. Let's create a sample logger file in a CSV format:

[1]:
```
import csv
import tempfile
import os

full_path=os.path.join(tempfile.gettempdir(), "logger.csv")

csvfile=open(full_path,'w', newline='')
obj=csv.writer(csvfile, delimiter='\t')
obj.writerow(['t', 'cell.id', 'RNA_concentration', 'cell_type'])
obj.writerow(['0', '1', '0.1' ,'0'])
obj.writerow(['0', '2', '0.2' ,'0'])
obj.writerow(['0', '3', '0.3' ,'0'])
obj.writerow(['1', '4', '0.4' ,'1'])
obj.writerow(['1', '5', '0.5' ,'2'])
obj.writerow(['1', '6', '0.6' ,'1'])
obj.writerow(['2', '7', '0.7' ,'2'])
obj.writerow(['2', '8', '0.8' ,'3'])
obj.writerow(['2', '9', '0.9' ,'2'])
csvfile.close()
```

As we can see from the above example, the time step between time points is =1. IF we didn't specify the time_step, then the value will be calculated form the Morpheus file.

[2]:
```
t_step=1
```

and the time symbol that was used in the model is =t. The time symbol by default is 't'. So, even if we didn't specify the time point here, the code will run fine.

[3]:
```
t_symbol ='t'
```

and the `field_of_interest` will be =`cell_type`

[4]:
```
field_of_interest = 'cell_type'
```

Finally, let's create a list of cell types that we are interested in. We have in the above file four different cell type (0, 1, 2, 3). However, let's assume that we are interested only on three of them (1, 2, 3)

[5]:
```
cell_types_list = [1, 2, 3]
```

Now, we are ready to call the function `count_cell_types()`, but before that, let's import the function's library. Don't forget to install the package first.

[6]:
```
import fitmulticell.sumstat.cell_types_cout as cs
```

We also need to import `util` library form fitmulticell to read the CSV file as pandas df

[7]:
```
import fitmulticell.util as util
```

No, we will read the CSV file using the `tsv_to_df` function form the external library

[8]:
```
logger_file = util.tsv_to_df("/tmp")
```

Let's see how the logger_file loks like

[9]:
```
logger_file
```

[9]:
```
   t  cell.id  RNA_concentration  cell_type
0  0        1                0.1          0
1  0        2                0.2          0
2  0        3                0.3          0
3  1        4                0.4          1
4  1        5                0.5          2
5  1        6                0.6          1
6  2        7                0.7          2
7  2        8                0.8          3
8  2        9                0.9          2
```

Now, let's call the method.

[10]:
```
cell_type_result=cs.count_cell_types(logger_file, field_of_interest, cell_types_list,␣
↪time_symbol=t_symbol)
print(f'The cell types counts for all time points is =\n {cell_type_result}')
```

```
The cell types counts for all time points is =
 {0:            n_cells
cell_type
1                0
2                0
3                0, 1:            n_cells
cell_type
1                2
2                1
3                0, 2:            n_cells
cell_type
1                0
2                2
3                1}
```

As the output shows, the result will be a dictionary of dataframes. The outer keys specify the time point, whereas the inner key specifies the cell type. the value of the inner dataframe describes the number of occurrence of the cell type at the specific time point

Now, after calculating the occurrence of the different cell types, let's try to plot the result. To do so, we will use a function that was built for this purpose. the function called `plot_different_cell_type()`.

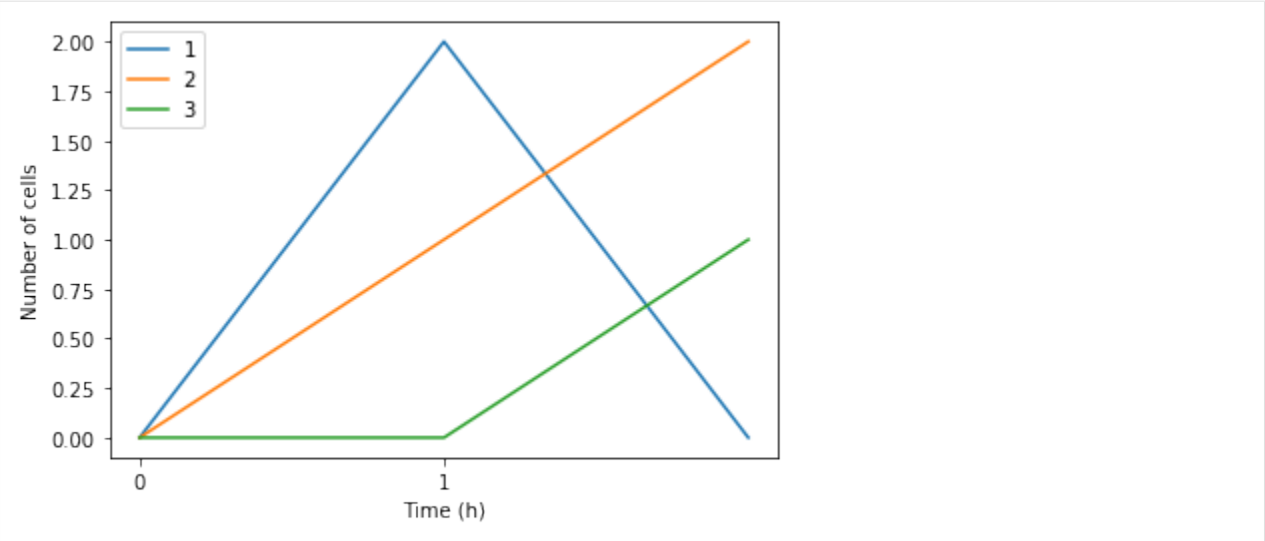
This function takes the output of the `count_cell_types()` function, the `cell_types_list`, and the `time_step` (optional) as an inputs. In addition, the output of this function will be the axis of the plot.

Before using the function, let's import it first.

[13]:
```
import fitmulticell.sumstat.plot_sumstat as pss
```

Let's now use the function:

[14]:
```
ax = pss.plot_different_cell_type(cell_type_result,cell_types_list)
```



As the figure illustrates, the occurrence of different cell types can be seen at different time points.

CHAPTER

FOUR

CONTAINERIZATION

To simplifiy and streamline the installation of morpheus, and to enable running it on high performance infrastructure with limited access rights, several docker images are available.

Instead of a standard system installation of Morpheus, these can be used by replacing the `executable` parameter of the `fmc.MorpheusModel` class. By default, it is set to `executable=morpheus`, targeting a standard installation of morpheus in the system and accessible via the path. Alternatively, the following container solutions can be used. The `MorpheusModel` copies a reparameterized version of your model file to `{loc}/model.xml`, where `{loc}` is given by the base directory specified in the `MorpheusModel` plus some random subfolder to contain the model and simulations. `{loc}` is a placeholder which the `MorpheusModel` internally replaces by the respective folder. In the following executable expressions, replace `$DIR` by the folder containing the container files.

4.1 Docker

- `executable="docker run -v {loc}:/sim --rm registry.gitlab.com/morpheus.lab/morpheus/morpheus-runner dockerrun_morpheus model.xml"`

4.2 Singularity

- in `$DIR`, run `singularity pull -F docker://registry.gitlab.com/morpheus.lab/morpheus/morpheus-runner`
- if your `{loc}` is not under an automounted directory, you need to mount it. $HOME is automounted in singularity
- `executable="singularity exec $DIR morpheus-runner_latest.sif /usr/bin/morpheus {loc}/model.xml -outdir {loc}"`

4.3 Charlie-cloud

- extract image to folder `$DIR`: `ch-pull2cir registry.gitlab.com/morpheus.lab/morpheus/morpheus-runner $DIR`
- note that charlie automounts your $HOME and e.g. also the /tmp directory
- `executable="ch-run -b {loc}:/sim $DIR -- /usr/bin/morpheus -file=sim/model.xml -outdir=sim"`

CHAPTER

FIVE

YOUTUBE TUTORIAL SERIES

In this tutorial series, we exlain different functionalities of the FitMultiCell pipeline. Here is a link to the playlist containing all parts of the series: https://youtube.com/playlist?list=PLRgo5axBHp5jWwqSHa_XpkAJ2YnxRphCi.

5.1 Part 1

The first part of this tutorial series is an explanation of how to install the pipeline with its different components: https://youtu.be/0qVro-hkyr0.

5.2 Part 2

The second part explains the different elements that need to be prepared by the user before starting the fitting process: https://youtu.be/Nq0yQI2gZRQ.

5.3 Part 3

The third part executes one of the application examples from the FitMultiCell documentation. This example is for fitting two parameters in a cell motility model: https://youtu.be/4tsNzU7dtdw. To read more about the model, please check this jupyter notebook: https://fitmulticell.readthedocs.io/en/latest/example/Demo_CellMotility.html.

5.4 Part 4

The fourth part explains how to run the FitMultiCell pipeline in a large cluster using the distributed memory system parallelization strategy: https://youtu.be/w9s0aS3sDVE.

CHAPTER

SIX

RELEASE NOTES

6.1 0.0 series

6.1.1 0.0.7 (2021-11-28)

- Fix CI triggers
- Consolidate badges

6.1.2 0.0.6 (2021-11-28)

- Add discrete distance to prior types
- Add basic unit tests
- Check code coverage via codecov.io and cobertura
- Apply the uncompromising Python code formatter black as pre-commit
- Apply automatic import sorting via isort as pre-commit

6.1.3 0.0.5 (2021-11-10)

- Modernize package and CI:
  - Add setup.cfg and pyproject.toml for package build and dependencies
  - Add tox and pre-commit hooks for testing and code quality checks via tox.ini and .flake8
  - Enforce pep8 code quality subset via flake8
  - Add API docs to documentation
  - Add CHANGELOG.rst and CONTRIBUTE.rst
  - Rename master branch to main
  - Streamline documentation build via .readthedocs.yaml
  - Add scripts for build of external dependencies
  - Add automatic deployment to PyPI
- Allow custom executables in model creation

6.1.4 0.0.4 (2021-11-03)

- Initial release on gitlab
- Add basic petab support

6.1.5 0.0.3 (2019-11-25)

- Fix error with utf-8 encoding
- Update notebooks to recent changes

6.1.6 0.0.2 (2019-10-10)

- Basic repository set up
- Added documentation on fitmulticell.readthedocs.io
- Implemented MorpheusModel class
- Designed a logo
- Added a basic usage example in doc/example

6.1.7 0.0.1 (2019-09-24)

Initial preparatory release, no functionality

CHAPTER

SEVEN

CONTACT

Discovered an error? Need help? Not sure if something works as intended? Please contact us!

If you think your problem could be of general interest, please consider creating an issue on gitlab, which will then also be helpful for other users: https://gitlab.com/fitmulticell/fit/issues

If you prefer to contact us via e-mail, please write to:

- Emad Alamoudi Emad Alamoudi (FitMultiCell) emad.alamoudi@uni-bonn.de
- Yannik Schaelte (rather parameter estimation) yannik.schaelte@gmail.com
- Jörn Starruß (rather modeling) joern.starruss@tu-dresden.de

CHAPTER

EIGHT

LICENSE

CHAPTER

NINE

LOGO

The FitMultiCell logo can be found in multiple variants in the doc/logo directory on gitlab, in svg and png format. It is made available under a creative commons CC0 license. You are encouraged to use it e.g. in presentations and posters.

CHAPTER

TEN

API REFERENCE

10.1 FitMultiCell

Fitting of multi-cellular models combining the tools morpheus and pyABC.

10.2 Model

Models wrap around a simulation engine (such as Morpheus) and making their results accessible to the parameter inference routine (such as pyABC).

class fitmulticell.model.**MorpheusModel**(*model_file: str*, *par_map: Dict[str, str]*, *sumstat: Optional[fitmulticell.sumstat.base.SummaryStatistics] = None*, *par_scale: Union[Dict[str, str], str] = 'lin'*, *exp_cond_map: Optional[Dict] = None*, *executable: str = 'morpheus'*, *gui_executable: str = 'morpheus-gui'*, *suffix: Optional[str] = None*, *prefix: str = 'morpheus_model_'*, *dir: Optional[str] = None*, *clean_simulation: bool = False*, *show_stdout: bool = False*, *show_stderr: bool = True*, *raise_on_error: bool = False*, *timeout: Optional[float] = None*, *name: Optional[str] = None*, *time_var: str = 'time'*, *outputdir: Optional[str] = None*, *ss_post_processing: Optional[Union[Callable, dict]] = None*)

Bases: pyabc.external.base.ExternalModel

Derived from pyabc.ExternalModel. Allows pyABC to call morpheus in order to do the model simulation, and then record the results for further processing.

Parameters

- **model_file** – The XML file containing the morpheus model.
- **par_map** – A dictionary from string to string, the keys being the parameter ids to be used in pyabc, and the values xpaths in the *morpheus_file*.
- **par_scale** – A dictionary or string to state the scale used to define the parameter space, e.g., lin, log10, log2
- **sumstat_funs** – List of functions to calculate summary statistics. The list entries are instances of fitmulticell.sumstat.SumstatFun.
- **executable** – The path to the morpheus executable. If None given, 'morpheus' is used.
- **suffix** – Suffix and prefix to use for the temporary folders created.
- **prefix** – Suffix and prefix to use for the temporary folders created.

- **dir** – Directory to put the temporary folders into. The default is the system's temporary files location. Note that these files are usually deleted upon system shutdown.
- **clean_simulation** – Whether to remove simulation files when they are no longer needed.
- **show_stdout** – Whether to show or hide the stdout and stderr streams.
- **show_stderr** – Whether to show or hide the stdout and stderr streams.
- **raise_on_error** – Whether to raise on an error in the model execution, or just continue.
- **name** – A name that can be used to identify the model, as it is saved to db. If None is passed, the model_file name is used.
- **time_var** – The name of the time variable as define in Morpheus model.
- **ignore_list** – A list of columns to ignore from Morpheus output. This is introduced to solve the issue with result that cannot be eliminated from morpheus output but yet are not used in the fitting process.
- **timeout** – Maximum execution time in seconds, after which Morpheus is stopped.
- **ss_post_processing** – A callable function to perform post processing on Morpheus output. If a dict is passed, then specific function will be applied to each summary statistics.
- **output_file** – A name of the file containing the simulation output.

__call__(*pars: pyabc.parameters.Parameter*)

Simulate data for parameters.

This function is used in ABCSMC (or rather the sample() function, which redirects here) to simulate data for given parameters *pars*.

__init__(*model_file: str*, *par_map: Dict[str, str]*, *sumstat: Optional[fitmulticell.sumstat.base.SummaryStatistics] = None*, *par_scale: Union[Dict[str, str], str] = 'lin'*, *exp_cond_map: Optional[Dict] = None*, *executable: str = 'morpheus'*, *gui_executable: str = 'morpheus-gui'*, *suffix: Optional[str] = None*, *prefix: str = 'morpheus_model_'*, *dir: Optional[str] = None*, *clean_simulation: bool = False*, *show_stdout: bool = False*, *show_stderr: bool = True*, *raise_on_error: bool = False*, *timeout: Optional[float] = None*, *name: Optional[str] = None*, *time_var: str = 'time'*, *outputdir: Optional[str] = None*, *ss_post_processing: Optional[Union[Callable, dict]] = None*)

Initialize the model.

Parameters

- **name** (*str, optional (default = "ExternalModel")*) – As in pyabc.Model.name.
- **ExternalHandler.** (*All other parameters as in*) –

compute_sumstats(*loc: str*) → dict

Compute summary statistics from the simulated data according to the provided list of summary statistics functions.

get_expcondmap_xpath_attr(*key*, *attrib='value'*)

Get the xpath and for the experimental conditions of interest

Parameters

- **key** (*str*) – name of experimental condition of interest.
- **attrib** (*str*) – the type of attribute that need to be changed on the xml file.

get_par_value_form_xml_file(*file_*, *param*)

Get a parameter value from the model's xml file. This is currently being used for testing purposes.

get_parmap_xpath_attr(*key*, *attrib='value'*)

Get the xpath and for the parameter of interest

Parameters

- **key** (*str*) – name of parameter of interest.
- **attrib** (*str*) – the type of attribute that need to be changed on the xml file.

sanity_check(*par: Optional[pyabc.parameters.Parameter] = None*)

Sanity check of the model.

In particular executes the model once.

Parameters **par** – Parameters at which to evaluate. If not specified, parameters are as in the model file.

write_modified_model_file(*file_: str*, *pars: Dict[str, float]*)

Write a modified version of the morpheus xml file to the target directory.

class fitmulticell.model.**MorpheusModels**(*models: Sequence[*fitmulticell.model.base.MorpheusModel*]*, *name: Optional[str] = None*)

Bases: pyabc.external.base.ExternalModel

Derived from pyabc.ExternalModel. Allows pyABC to call morpheus in order to do the models simulation, and then record the results for further processing.

Parameters

- **models** (*A list of MorpheusModel objects.*) –
- **name** (*Name of the joint model.*) –

__call__(*pars: pyabc.parameters.Parameter*)

This function is used in ABCSMC (or rather the sample() function, which redirects here) to simulate data for given parameters *pars* and given experimental conditions.

__init__(*models: Sequence[*fitmulticell.model.base.MorpheusModel*]*, *name: Optional[str] = None*)

Initialize the model.

Parameters

- **name** (*str, optional (default = "ExternalModel")*) – As in pyabc.Model.name.
- **ExternalHandler.** (*All other parameters as in*) –

get_parmap_xpath_attr(*key*, *attrib='value'*)

Get the xpath and for the parameter of interest

Parameters

- **key** (*str*) – name of parameter of interest.
- **attrib** (*str*) – the type of attribute that need to be changed on the xml file.

write_modified_models_file(*file_*, *pars*, *exp_cod*)

Write a modified version of the morpheus xml file to the target directory.

CHAPTER

ELEVEN

CONTRIBUTE

11.1 Workflow

If you start working on a new feature or a fix, please create an issue, shortly describing the issue, and assign yourself. Your startpoint should always be the `develop` branch, which contains the lastest updates.

Create an own branch or fork, on which you can implement your changes. To get your work merged, please:

1. create a pull request to the `develop` branch with a meaningful summary,
2. check that code changes are covered by tests, and all tests pass,
3. check that the documentation is up-to-date,
4. request a code review from the main developers.

Merge requests to develop should be squashed-and-merged, while merge requests (from develop) to main should be merged. This is to keep the history clean.

11.2 Environment

If you contribute to the development of FitMultiCell, consider installing developer requirements, for e.g. local testing, via:

```
pip install -r requirements-dev.txt
```

This installs the tools described below.

11.2.1 Pre-commit hooks

Firstly, this installs a pre-commit tool. To add those hooks to the *.git* folder of your local clone such that they are run on every commit, run:

```
pre-commit install
```

When adding new hooks, consider manually running `pre-commit run --all-files` once as usually only the diff is checked. The configuration is specified in `.pre-commit-config.yaml`.

Should it be necessary to perform commits without pre-commit verification, use `git commit --no-verify` or the shortform `-n`.

11.2.2 Tox

Secondly, this installs the virtual testing tool tox, which we use for all tests, format and quality checks. Its configuration is specified in `tox.ini`. To run it locally, simply execute:

```
tox [-e flake8,doc]
```

with optional `-e` options specifying the environments to run, see `tox.ini` for defaults. tox creates virtual images in a `.tox/` subfolder.

11.3 GitLab CI/CD

For automatic continuous integration testing, we use GitLab CI/CD All tests are run there on pull requests and required to pass. The configuration is specified in `.gitlab-ci.yml`.

11.4 Documentation

To make FitMultiCell easily usable, we try to provide good documentation, including code annotation and usage examples. The documentation is hosted at https://fitmulticell.readthedocs.io/en/latest/, a version for the `develop` branch at https://fitmulticell.readthedocs.io/en/develop/. These are updated automatically on merges to the main and develop branches. To create the documentation locally, run:

```
tox -e doc
```

The documentation is then undre `doc/_build`. Alternatively, install the requirements via:

```
pip install .[doc]
```

and then compile the documentation via:

```
cd doc
make html
```

When adding code, all modules, classes, functions, parameters, code blocks should be properly documented. When adding examples, these should be properly embedded.

11.5 Unit tests

Unit tests are located in the `test` folder. All files starting with `test_` contain tests and are automatically run on GitLab CI/CD. Run them locally via e.g.:

```
tox -e base
```

You can also run only specific tests, see e.g. the pytest documentation for details. Unit tests can be written with pytest or unittest.

Code changes should always be covered by unit tests. It might not always be easy to test code which is based on random sampling, but we still encourage general sanity and integration tests. We highly encourage a test-driven development style.

11.6 PEP8

We try to respect the PEP8 coding standards. We run flake8 as part of the tests. The flake8 plugins used are specified in `tox.ini` and the flake8 configuration is given in `.flake8`. You can run the checks locally via:

```
tox -e flake8
```

CHAPTER

TWELVE

DEPLOY

New production releases should be created every time the `main` branch is updated.

12.1 Versions

For version numbers, we use `A.B.C`, where

- `C` is increased for bug fixes,
- `B` is increased for new features and minor API breaking changes,
- `A` is increased for major API breaking changes,

as suggested by the Python packaging guide.

12.2 Create a new release

After new commits have been added via pull requests to the `develop` branch, changes can be merged to `main` and a new version of pyPESTO can be released.

12.2.1 Merge into main

1. create a pull request from `develop` to `main`,
2. check that all code changes are covered by tests and all tests pass,
3. check that the documentation is up-to-date,
4. adapt the version number in `fitmulticell/version.py` (see above),
5. update the release notes in `CHANGELOG.rst`,
6. request a code review,
7. merge into the origin `main` branch.

To be able to actually perform the merge, sufficient rights may be required. Also, at least one review is required.

12.2.2 Create a release on GitHub

After merging into `main`, create a new release on GitHub. This can be done either directly on the project GitHub website, or via the CLI as described in Git Basics - Tagging. In the release form,

- specify a tag with the new version as specified in `fitmulticell/version.py`,
- include the latest additions to `CHANGELOG.rst` in the release description.

12.3 Upload to PyPI

The upload to the python package index PyPI has been automatized via GitLab CI/CD and is triggered whenever a new release tag is published.

Should it be necessary to manually upload a new version to PyPI, proceed as follows: First, a so called "wheel" is created via:

```
python setup.py bdist_wheel
```

A wheel is essentially a zip archive which contains the source code and the binaries (if any).

This archive is uploaded using twine:

```
twine upload dist/fitmulticell-x.y.z-py3-non-any.wheel
```

replacing x.y.z by the respective version number.

For a more in-depth discussion see also the section on distributing packages of the Python packaging guide.

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

fitmulticell, 75
fitmulticell.model, 75

INDEX

Symbols

__call__() (*fitmulticell.model.MorpheusModel method*), 76
__call__() (*fitmulticell.model.MorpheusModels method*), 77
__init__() (*fitmulticell.model.MorpheusModel method*), 76
__init__() (*fitmulticell.model.MorpheusModels method*), 77

C

compute_sumstats() (*fitmulticell.model.MorpheusModel method*), 76

F

fitmulticell
    module, 75
fitmulticell.model
    module, 75

G

get_expcondmap_xpath_attr() (*fitmulticell.model.MorpheusModel method*), 76
get_par_value_form_xml_file() (*fitmulticell.model.MorpheusModel method*), 76
get_parmap_xpath_attr() (*fitmulticell.model.MorpheusModel method*), 77
get_parmap_xpath_attr() (*fitmulticell.model.MorpheusModels method*), 77

M

module
    fitmulticell, 75
    fitmulticell.model, 75
MorpheusModel (*class in fitmulticell.model*), 75
MorpheusModels (*class in fitmulticell.model*), 77

S

sanity_check() (*fitmulticell.model.MorpheusModel method*), 77

W

write_modified_model_file() (*fitmulticell.model.MorpheusModel method*), 77
write_modified_models_file() (*fitmulticell.model.MorpheusModels method*), 77