pleione Documentation

Rodrigo Santibáñez

Jun 14, 2019

CONTENTS

Pleione is a python3 package that implement methods that are common to traditional modeling frameworks, and apply them to analyze Rule-Based Models.

Here you'll find the necessary documentation to install and use the methods in Pleione. At the moment, Pleione parameterizes Rule-Based Models written either in *BioNetGen* (BioNetGen) or *kappa* language (Kappa). Models are simulated with BNG2 (BioNetGen2, PMID 27402907), NFsim (NFsim, PMID 26556387), KaSim (KaSim, PMID 29950016), or PISKaS (PISKaS, PMID 29175206). Please contact us or write an issue to include your favorite stochastic simulator to Pleione (https://github.com/glucksfall/pleione/issues).

Pleione implements a Genetic Algorithm with elitism, on the contrary to BioNetFit (BioNetFit, PMID 26556387) that implements a parents selection within a distribution probability that is inverse to the rank. Nonetheless, Pleione's methods to parameterize Rule-Based Models include both, a uniform or inverse to the rank probability to select models from within an elite or all models.

The plan to add methods into Pleiades (pleiades) includes a sensitivity analysis and a parameterization employing a Particle Swarm Optimization protocol. You could write us if you wish to add methods into pleione or aid in the development of them.

CHAPTER

ONE

INSTALLATION

There are two different ways to install pleione:

1. **Install pleione natively (Recommended).**

   OR

2. **Clone the Github repository.** If you are familiar with git, pleione can be cloned and the respective folder added to the python path. Further details are below.

Note: Need Help? If you run into any problems with installation, please visit our chat room: https://gitter.im/glucksfall/pleiades

1.1 Option 1: Install pleione natively on your computer

The recommended approach is to use system tools, or install them if necessary. To install python packages, you could use pip, or download the package from python package index.

1. **Install with system tools**

   With pip, you simple need to execute and pleione will be installed on `$HOME/.local/lib/python3.6/site-packages` folder or similar.

   ```
   pip3 install pleione --user
   ```

   If you have system rights, you could install pleione for all users with

   ```
   sudo -H pip3 install pleione
   ```

2. **Download from python package index**

   Alternatively, you could download the package (useful when pip fails to download the package because of lack of SSL libraries) and then install with pip. For instance:

   ```
   wget https://files.pythonhosted.org/packages/05/45/
   ↪fc5b44d5211f297ed7d66361cc7062935d7afece812b605b3f3d6a38c04d/pleione-1.0.4-py3-
   ↪none-any.whl
   pip3 install pleione-1.0.4-py3-none-any.whl --user
   ```

   Note: Why Python3?: Pleione is intended to be used with python3, despite the lack of incompatible functions with python2, because the latter won't receive further development past 2020. Although, the code could be

optimize with specific python3 functions over dictionaries, therefore making incompatible with python2.

Note: **pip, Python and Anaconda**: Be aware which pip you invoque. You could install pip3 with `sudo apt-get install python3-pip` if you have system rights, or install python3 from source, and adding `<python3 path>/bin/pip3` to the path, or linking it in a directory like `$HOME/bin` which is commonly added to the path at system login. Please be aware that, if you installed Anaconda, pip could be linked to the Anaconda specific version of pip, which will install pleione into Anaconda's installation folder. Type `which pip` or `which pip3` to find out the source of pip, and type `python -m site` or `python3 -m site` to find out where is more likely pleione would be installed.

1.2 Option 2: Clone the Github repository

1. **Clone with git**

   The source code is uploaded and maintained through Github at https://github.com/glucksfall/pleione. Therefore, you could clone the repository locally, and then add the folder to the `PYTHONPATH`. Beware that you should install the *pandas* package (pandas) by any means.

```
git clone https://github.com/glucksfall/pleione /opt
echo export PYTHONPATH="\$PYTHONPATH:/opt/pleione" >> $HOME/.profile
```

Note: Adding the path to `$HOME/.profile` allows python to find the package installation folder after each user login. Similarly, adding the path to `$HOME/.bashrc` allows python to find the package after each terminal invocation.

CHAPTER

TWO

PARAMETERS ESTIMATION

Pleione's parameterization methods rely on Computational Load Distribution. The naïve approach is to use the python's *multiprocessing* API and each simulation distributed within the Pool of available (minus one) cores. This approach would make pleione's methods compatible with Microsoft Windows and Apple OS X. However, to take fully advantage of High-Performance Computing architectures, pleione's methods rely on SLURM –*Simple Linux Utility for Resource Management*– (SLURM) to distribute simulations through your infrastructure, remote infrastructures, and cloud services like Google Compute Engine, Microsoft Azure, and Amazon Elastic Compute Cloud.

Up to date, pleione's parameterization methods rely on 4 simulations engines: KaSim and PISKaS simulate *kappa* language models. Unlike KaSim, PISKaS is able to simulate multiple compartment models distributing the calculation of each compartment through multiple cores. In the other hand, BioNetGen2 and NFsim simulate *BioNetGen* language models. Despite KaSim and PISKaS, BioNetGen2 does not provide a Command-Line Interface to especify simulation parameters and rather, the simulation parameters (e.g. time to simulation, number of points to report, ...) must be given inside the model specification. Moreover, you need to especify the simulation engine to use: Deterministic simulation through *CVODE*, the Stochastic Simulation Algorithm *SSA*, Exact Hybrid Particle/Population Algorithm *HPP*, and the Partition-Leap Algorithm *PLA*. Moreover, NFsim could be used by BioNetGen2 to simulate models or called externally after creating the model xml especification with BioNetGen2 –xml option.

Because the software requirements and differences, we provide specific documentation to all of them rather than provide common guidelines and then stating the differences.

Parameterization of kappa-language Rule-Based Models

2.1 Parameterization with KaSim

1. **Prepare the model**

   Pleione parmeterization methods find which variables will be calibrated using the symbol `//` (doble slash, as C/C++) followed by:

   - An initial distribution type: `uniform`, `loguniform`, `lognormal`
   - An initial search space: `[min max]` or `[mean standard_deviation]` in the case `lognormal` was selected.
   - A type of mutation: `uniform` or `loguniform` to use a new search space; or `factor` to perform a local mutation search
   - A search space for mutated parameters: `[min max]` or `[probability fold_change]`
   - An optional mutation rate per parameter. Without it, a global mutation rate is used.

   For instace:

```
%var: 'KD1__FREE__' 1.000000e+00 // loguniform[0.01 100] factor[0.2 0.1]
%var: 'km1__FREE__' 1.000000e+00 // loguniform[0.01 100] factor[0.2 0.1]
%var: 'K2RT__FREE__' 1.000000e+00 // loguniform[0.01 100] factor[0.2 0.1]
%var: 'km2__FREE__' 1.000000e+00 // loguniform[0.01 100] factor[0.2 0.1]
%var: 'kphos__FREE__' 1.000000e+00 // loguniform[0.01 100] factor[0.2 0.1]
%var: 'kdephos__FREE__' 1.000000e+00 // loguniform[0.01 100] factor[0.2 0.1]
```

or the following configuration if the model is written in syntax 3 (KaSim v3):

```
%var: 'KD1__FREE__' 1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
%var: 'km1__FREE__' 1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
%var: 'K2RT__FREE__' 1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
%var: 'km2__FREE__' 1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
%var: 'kphos__FREE__' 1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
%var: 'kdephos__FREE__' 1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
```

Note: **Factor mutation:** This type of mutation strategy comes from BioNetFit and selects a random value from the range `0.9 * old_value, 1.1 * old_value` if the declared value is `0.1` with probability `0.2`.

2. **Prepare the data files**

   KaSim produce simulations files with the following format. Please prepare data files with the same format to avoid incompatibilities.

```
"[T]","RLbonds","pR"
600.,0,355.3
610.,114.072,356.44
620.,139.1838,349.96
630.,149.1534,343.98
640.,156.8684,342.6
650.,156.788,335.62
660.,163.6668,337.48
```

Note: **About the example model:** The model has three parts: An equilibration of 600 seconds, then the model is modified to add a quantity of `L(r)` agents, and then perform the actual simulation for 60 seconds. Despite BNG2 and NFsim, KaSim reports the whole simulation, so to compare effectively, we must offset the time of the experimental data by 600.

2. **Prepare a sbatch configuration file**

   Use the following code as template to make a shell script and queue it with sbatch. Note that the `export` statement is inside the code to tell SLURM to add the path and ensure proper execution when pleione was cloned with git. Also, `python3` redirects to either the system installed executable (with pandas installed either as admin or user) or redirects to the user compiled executable if an alias exists for it.

```
#!/bin/sh

#SBATCH --no-requeue
#SBATCH --partition=cpu

#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1

```

(continues on next page)

(continued from previous page)

```
#SBATCH --job-name=pleione-kasim
#SBATCH --output=stdout.txt
#SBATCH --error=stderr.txt

export PYTHONPATH="$PYTHONPATH:$HOME/opt/git-glucksfall-pleione-master"

MODEL=pysbmodel-example6-kasim.kappa
FINAL=660
STEPS=10 # KaSim interprets as the period, not how many points to report!

PARTITION=$SLURM_JOB_PARTITION
DATA=../exp-data/kasim/data-*.txt

NUM_ITER=100
NUM_SIMS=10
POP_SIZE=100
POP_BEST=0

SWAP=0.5
RATE=0.5
ERROR="MWUT"
UTABLE=./ucrit.txt

python3 -m pleione.kasim --model=$MODEL --final=$FINAL --steps=$STEPS \
--iter=$NUM_ITER --pops=$POP_SIZE --sims=$NUM_SIMS --best=$POP_BEST \
--data=$DATA --rate=$RATE --swap=$SWAP --error=$ERROR --crit=$UTABLE \
--slurm=$PARTITION --syntax=4
```

Note: sbatch or python multiproccesing? To execute Pleione outside a SLURM queue, simple execute the shell script with `sh`, `bash` or any shell interpreter without the `slurm` option. Be aware that, if SLURM is running in the same machine, Pleione subprocess would impact negatively in other user's threads, and viceversa, since a cpu core could execute concurrently two threads.

Note: Need help? type `python3 -m pleione.kasim --help` to find out the available command options.

2.2 Parameterization with PISKaS

1. **Prepare the model**

   Pleione parmeterization methods find which variables will be calibrated using the symbol # (number sign, hash or pound sign) followed by:

   - An initial distribution type: `uniform`, `loguniform`, `lognormal`
   - An initial search space: `[min max]` or `[mean standard_deviation]` in the case `lognormal` was selected.
   - A type of mutation: `uniform` or `loguniform` to use a new search space; or `factor` to perform a local mutation search
   - A search space for mutated parameters: `[min max]` or `[probability fold_change]`
   - An optional mutation rate per parameter. Without it, a global mutation rate is used.

For instace:

```
%var: 'KD1__FREE__' 1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
%var: 'km1__FREE__' 1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
%var: 'K2RT__FREE__' 1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
%var: 'km2__FREE__' 1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
%var: 'kphos__FREE__' 1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
%var: 'kdephos__FREE__' 1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
```

Note: **Factor mutation:** This type of mutation strategy comes from BioNetFit and selects a random value from the range `0.9 * old_value, 1.1 * old_value` if the declared value is `0.1` with probability `0.2`.

2. **Prepare the data files**

   PISKaS produce simulations files with a similar format as follows. Please prepare data files, replacing the "#" with the name of the compartments, including the initial space in each subsequent row.

```
example time 'RLbonds' 'pR'
 6.000000E+02 0 355.3
 6.100000E+02 114.072 356.44
 6.200000E+02 139.1838 349.96
 6.300000E+02 149.1534 343.98
 6.400000E+02 156.8684 342.6
 6.500000E+02 156.788 335.62
 6.600000E+02 163.6668 337.48
```

Note: **An extra column name?** PISKaS produces one output for each compartment declared in the model. Therefore, adding the name of the compartment as the first column allows the code to identify the corresponding experimental data with the simulated compartment and apply correctly the fitness function. Finally, the model error is the sum of each fitness per compartment.

Note: **About the example model:** The model has three parts: An equilibration of 600 seconds, then the model is modified to add a quantity of `L(r)` agents, and then perform the actual simulation for 60 seconds. Despite BNG2 and NFsim, PISKaS reports the whole simulation, so to compare effectively, we must offset the time of the experimental data by 600.

2. **Prepare a sbatch configuration file**

   Use the following code as template to make a shell script and queue it with sbatch. Note that the `export` statement is inside the code to tell SLURM to add the path and ensure proper execution when pleione was cloned with git. Also, `python3` redirects to either the system installed executable (with pandas installed either as admin or user) or redirects to the user compiled executable if an alias exists for it.

```
#!/bin/sh

#SBATCH --no-requeue
#SBATCH --partition=cpu

#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1

```

(continues on next page)

(continued from previous page)

```
#SBATCH --job-name=pleione-piskas
#SBATCH --output=stdout.txt
#SBATCH --error=stderr.txt

export PYTHONPATH="$PYTHONPATH:$HOME/opt/git-glucksfall-pleione-master"

MODEL=pysbmodel-example6-piskas.kappa
FINAL=660
STEPS=66

PARTITION=$SLURM_JOB_PARTITION
DATA=../exp-data/piskas/data-*.txt

NUM_ITER=100
NUM_SIMS=10
POP_SIZE=100
POP_BEST=0

SWAP=0.5
RATE=0.5
ERROR="MWUT"
UTABLE=./ucrit.txt

python3 -m pleione.piskas --model=$MODEL --final=$FINAL --steps=$STEPS \
--iter=$NUM_ITER --pops=$POP_SIZE --sims=$NUM_SIMS --best=$POP_BEST \
--data=$DATA --rate=$RATE --swap=$SWAP --error=$ERROR --crit=$UTABLE \
--slurm=$PARTITION
```

Note: sbatch or python multiproccesing? To execute Pleione outside a SLURM queue, simple execute the shell script with `sh`, `bash` or any shell interpreter without the `slurm` option. Be aware that, if SLURM is running in the same machine, Pleione subprocess would impact negatively in other user's threads, and viceversa, since a cpu core could execute concurrently two threads.

Note: Need help? type `python3 -m pleione.piskas --help` to find out the available command options.

Parameterization of BioNetGen language Rule-Based Models

2.3 Parameterization with BioNetGen2

1. **Prepare the model**

   Pleione parmeterization methods find which variables will be calibrated using the symbol # (number sign, hash or pound sign) followed by:

   - An initial distribution type: `uniform`, `loguniform`, `lognormal`
   - An initial search space: `[min max]` or `[mean standard_deviation]` in the case `lognormal` was selected.
   - A type of mutation: `uniform` or `loguniform` to use a new search space; or `factor` to perform a local mutation search
   - A search space for mutated parameters: `[min max]` or `[probability fold_change]`

- An optional mutation rate per parameter. Without it, a global mutation rate is used.

For instace:

```
KD1__FREE__          1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
km1__FREE__          1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
K2RT__FREE__         1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
km2__FREE__          1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
kphos__FREE__        1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
kdephos__FREE__      1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
```

Note: **Factor mutation:** This type of mutation strategy comes from BioNetFit and selects a random value from the range `0.9 * old_value, 1.1 * old_value` if the declared value is `0.1` with probability `0.2`.

2. **Prepare the data files**

   BNG2 produce simulations files with the following format. Please prepare data files with the same format, including the initial space in each row.

```
#           time          RLbonds               pR
 0.00000000E+00  0.00000000E+00  3.55300000E+02
 1.00000000E+01  1.14072000E+02  3.56440000E+02
 2.00000000E+01  1.39183800E+02  3.49960000E+02
 3.00000000E+01  1.49153400E+02  3.43980000E+02
 4.00000000E+01  1.56868400E+02  3.42600000E+02
 5.00000000E+01  1.56788000E+02  3.35620000E+02
 6.00000000E+01  1.63666800E+02  3.37480000E+02
```

3. **Prepare a sbatch configuration file**

   Use the following code as template to make a shell script and queue it with sbatch. Note that the `export` statement is inside the code to tell SLURM to add the path and ensure proper execution when pleione was cloned with git. Also, `python3` redirects to either the system installed executable (with pandas installed either as admin or user) or redirects to the user compiled executable if an alias exists for it.

```
#!/bin/sh
export PYTHONPATH="$PYTHONPATH:$HOME/opt/git-glucksfall-pleione-master"

#SBATCH --no-requeue
#SBATCH --partition=cpu

#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1

#SBATCH --job-name=pleione-bng2
#SBATCH --output=stdout.txt
#SBATCH --error=stderr.txt

MODEL=pysbmodel-example6-bng2.bngl # the model should have the .bngl extension

PARTITION=$SLURM_JOB_PARTITION
DATA=../exp-data/bng2/data-*.txt

NUM_ITER=100
NUM_SIMS=10
POP_SIZE=100
```

(continues on next page)

(continued from previous page)

```
POP_BEST=0

SWAP=0.5
RATE=0.5
ERROR="MWUT"
UTABLE=./ucrit.txt

python3 -m pleione.bng2 --model=$MODEL \
--iter=$NUM_ITER --pops=$POP_SIZE --sims=$NUM_SIMS --best=$POP_BEST \
--data=$DATA --rate=$RATE --swap=$SWAP --error=$ERROR --crit=$UTABLE \
--slurm=$PARTITION
```

Note: sbatch or python multiproccesing? To execute Pleione outside a SLURM queue, simple execute the shell script with `sh`, `bash` or any shell interpreter without the `slurm` option. Be aware that, if SLURM is running in the same machine, Pleione subprocess would impact negatively in other user's threads, and viceversa, since a cpu core could execute concurrently two threads.

Note: Need help? type `python3 -m pleione.bng2 --help` to find out the available command options.

2.4 Parameterization with NFsim

1. **Prepare the model**

   Pleione parmeterization methods find which variables will be calibrated using the symbol `#` (number sign, hash or pound sign) followed by:

   - An initial distribution type: `uniform`, `loguniform`, `lognormal`
   - An initial search space: `[min max]` or `[mean standard_deviation]` in the case `lognormal` was selected.
   - A type of mutation: `uniform` or `loguniform` to use a new search space; or `factor` to perform a local mutation search
   - A search space for mutated parameters: `[min max]` or `[probability fold_change]`
   - An optional mutation rate per parameter. Without it, a global mutation rate is used.

   For instace:

```
KD1__FREE__         1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
km1__FREE__         1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
K2RT__FREE__        1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
km2__FREE__         1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
kphos__FREE__       1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
kdephos__FREE__     1.000000e+00 # loguniform[0.01 100] factor[0.2 0.1]
```

Note: Factor mutation: This type of mutation strategy comes from BioNetFit and selects a random value from the range `0.9 * old_value, 1.1 * old_value` if the declared value is `0.1` with probability `0.2`.

2. **Prepare the data files**

NFsim produce simulations files with the following format. Please prepare data files with the same format to avoid incompatibilities.

```
time, RLbonds, pR
0.00000000E+00, 0.00000000E+00, 3.55300000E+02
1.00000000E+01, 1.14072000E+02, 3.56440000E+02
2.00000000E+01, 1.39183800E+02, 3.49960000E+02
3.00000000E+01, 1.49153400E+02, 3.43980000E+02
4.00000000E+01, 1.56868400E+02, 3.42600000E+02
5.00000000E+01, 1.56788000E+02, 3.35620000E+02
6.00000000E+01, 1.63666800E+02, 3.37480000E+02
```

2. **Prepare a sbatch configuration file**

   Use the following code as template to make a shell script and queue it with sbatch. Note that the `export` statement is inside the code to tell SLURM to add the path and ensure proper execution when pleione was cloned with git. Also, `python3` redirects to either the system installed executable (with pandas installed either as admin or user) or redirects to the user compiled executable if an alias exists for it.

```
#!/bin/sh

#SBATCH --no-requeue
#SBATCH --partition=cpu

#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1

#SBATCH --job-name=pleione-nfsim
#SBATCH --output=stdout.txt
#SBATCH --error=stderr.txt

export PYTHONPATH="$PYTHONPATH:$HOME/opt/git-glucksfall-pleione-master"

MODEL=pysbmodel-example6-nfsim.bngl # the model should have the .bngl extension
FINAL=60
STEPS=6

PARTITION=$SLURM_JOB_PARTITION
DATA=../exp-data/nfsim/data-*.txt

NUM_ITER=100
NUM_SIMS=10
POP_SIZE=100
POP_BEST=0

SWAP=0.5
RATE=0.5
ERROR="MWUT"
UTABLE=./ucrit.txt

python3 -m pleione.nfsim --model=$MODEL --final=$FINAL --steps=$STEPS \
--iter=$NUM_ITER --pops=$POP_SIZE --sims=$NUM_SIMS --best=$POP_BEST \
--data=$DATA --rate=$RATE --swap=$SWAP --error=$ERROR --crit=$UTABLE \
--slurm=$PARTITION
```

Note: sbatch or python multiproccesing? To execute Pleione outside a SLURM queue, simple execute the shell script with `sh`, `bash` or any shell interpreter without the `slurm` option. Be aware that, if SLURM is running in the

same machine, Pleione subprocess would impact negatively in other user's threads, and viceversa, since a cpu core could execute concurrently two threads.

Note: **Need help?** type `python3 -m pleione.nfsim --help` to find out the available command options.

Common to all parameterization methods, there are 9 algebraic objective functions and one statistical function already implemented in the code. Moreover, the code sort the models by their rank and therefore, ranks can be added and sorted again, making the possibility to use a Multiple Objective Genetic Algorithm.

2.5 Model Validation

Pleione's parameter calibration scripts call an external script to calculate fitness to experimental data. You could use one of the following script to calculate the fitness of your parameterized model against an independent experimental data set:

```
python3 -m pleione.bng2-doerror --data foo --sims bar \
--file output.txt --error MWUT --crit utable.txt
```

OR

```
python3 -m pleione.kasim-doerror --data foo --sims bar \
--file output.txt --error MWUT --crit utable.txt
```

OR

```
python3 -m pleione.nfsim-doerror --data foo --sims bar \
--file output.txt --error MWUT --crit utable.txt
```

OR

```
python3 -m pleione.piskas-doerror --data foo --sims bar \
--file output.txt --error MWUT --crit utable.txt
```

Note: **Fitness Function** Pleione currently support ten goodness of fit functions. To calculate more than one function, include a comma-only separated list such as `MWUT,SSQ`. In doing so, this will calculate the contribution of both o more metrics to the overall error and aid to validate of dischard a model calibration. More information in *Objective Functions*

Note: **Need Help?** Type `python3 -m pleione.$STOCH_ENGINE-doerror --help` where `$STOCH_ENGINE` can be the currently supported stochastic engines: BNG2, NFsim, KaSim and PISKaS (all in lower cases, for instance `nfsim-doerror`)

2.6 Objective Functions

Common to all parameterization methods, there are 9 algebraic objective functions and one statistical function already implemented in the code. Moreover, the code sort the models by their rank and therefore, ranks can be added and sorted again, making the possibility to use a Multiple Objective Genetic Algorithm.

2.6.1 Algebraic Objective Functions

Here are the formulas to calculate the error between multiple data sets and simulations files.

- **Squared Difference of Means (SDM; formerly Mean Square Error, MSE):**

$$\left(\frac{1}{m}\sum_{i=1}^{m}\exp_i - \frac{1}{n}\sum_{j=1}^{n}\mathrm{sim}_j\right)^2$$

- **Absolute Difference of Means (ADM; formerly Mean Absolute Error, MAE):**

$$\mathrm{abs}\left(\frac{1}{m}\sum_{i=1}^{m}\exp_i - \frac{1}{n}\sum_{j=1}^{n}\mathrm{sim}_j\right)$$

- **Pair-Wise Square Deviation (PWSD):**

$$\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\left(\exp_i - \mathrm{sim}_j\right)^2$$

- **Absolute Pair-Wise Deviation (APWSD):**

$$\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\mathrm{abs}\left(\exp_i - \mathrm{sim}_j\right)$$

- **Normalized Pair-Wise Square Deviation (NPWSD):**

$$\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\left(\frac{\exp_i - \mathrm{sim}_j}{\exp_i}\right)^2$$

- **Absolute Normalized Pair-Wise Deviation (ANPWSD):**

$$\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\mathrm{abs}\left(\frac{\exp_i - \mathrm{sim}_j}{\exp_i}\right)$$

- **Sum of SQuares (SSQ):**

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\left(\exp_i - \mathrm{sim}_j\right)^2$$

- **Chi-Square (CHISQ):**

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\left(\frac{\exp_i - \mathrm{sim}_j}{\sigma_{\exp}}\right)^2$$

- **Mean Normalized Square Error (MNSE):**

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\left(\frac{\exp_i - \mathrm{sim}_j}{\overline{\exp}}\right)^2$$

Note: Need a different Objective Function? The code that calculates the error is separated from the main Genetic Algorithm. This make useful to encode other Objective Functions if the already implemented does not apply to your necessities. You could contact us to add your function to the pleione package.

2.6.2 Statistical Objective Functions

We implemented the Mann-Whitney U-test (MWUT) to calculate the error between experimental data and simulations. The U-test is a non-parametric statistical test that, within a confidence level, determine if a stochastic repeated measurements is identical or not to another repeated measurements.

1. We count how many times experimental data (exp_i) are larger than simulated values (sim_j):

for i in range(len(exp)):
    for j in range(len(sim)):
        if $exp_i > sim_j$:
            $U_{exp} \leftarrow U_{exp} + 1.0$

else if $exp_i < sim_j$:
$U_{sim} \leftarrow U_{sim} + 1.0$
else:
$U_{exp} \leftarrow U_{exp} + 0.5$
$U_{sim} \leftarrow U_{sim} + 0.5$

2. We determine if U_{exp} is statistically significant:

for i in range(len(exp)):
if $\text{len}(exp) \times \text{len}(sim) - \min(U_{exp}, U_{sim}) \leq U_{\text{critic}}$:
null hypothesis, H_0, is rejected
$U_{\text{model}} \leftarrow U_{\text{model}} + 1.0$

Note: The U-test is the only fitness function that has known limits: For a *perfect* model, the U-test is zero. A complete wrong model will have a U_{model} equal to the number of Observables times the number of experimental time points. For instance, the example model we use to compare with BioNetFit has 2 Observables and 7 experimental time points, then a max U_{model} equal to 14.

2.6.3 Multiple Objective Functions

A Multiple Objective Function is build from two or more fitness functions. Firstly, a fitness is calculated and all models ranked. Then, the next fitness. Finally, the sum of ranks is use to rank agains the models.

Algoritmically:

$$rank_1 = \text{sort models following function 1}$$
$$\vdots$$
$$rank_n = \text{sort models following function n}$$

$$rank_{MO} = \text{sort models following } (rank_1 + \ldots + rank_n)$$

Note: We currently don't provide weights to rank the models. Be aware that, if you use multiple algebraic functions and the statistical fitness function, the importance of the statistical function is diluted.

2.6.4 Add a fitness function to Pleione

Each simulator are provided with two scripts that calculate errors. They are located at the same path as the main scripts that calibrate. Inside each, there is a template intended with instructions:

```
# Fitness Calculation Template:
if set(args.error).issuperset(set(['the-acronysm'])):
        func = 0
        func = an algebraic expression combining the data average (data_
↪avrg), data variance (data_stdv), simulation average (sims_stdv),
        single experimental files (data.loc[i]) and/or simulation files␣
↪(sims.loc[i]).
        # Please consider this variables are DataFrames, meaning that␣
↪division is a method (pandas.DataFrame.division)
        # Please calculate average or standard deviation values from data.
↪loc[i] and sims.loc[i] if they are needed from them (as in SDM)
```

(continues on next page)

(continued from previous page)

```
        error['acronysm'] = '{:.6e}'.format(func.dropna(axis = 0, how = 'all
↪').dropna(axis = 1, how = 'all').sum().sum())
        # drop NaN values (from experimental data without simulation point␣
↪or vice-versa), sum the two dimensions, and return a 6 float points␣
↪scientific notation number
```

To use:

1) Define an acronysm for your fitness function and replace "the-acronysm"
2) Define func as an operation of DataFrames: data_avrg, data_stdv, sims_avrg, sims_stdv, data.loc[i], and sims.loc[i]

Note: simulator-doerror.py scripts calculates one single fitness function at the time. The Square Difference of Means has code to calculate the average from data and simulations.

Note: Need a different Objective Function? The code that calculates the error is separated from the main Genetic Algorithm. This make useful to encode other Objective Functions if the already implemented does not apply to your necessities. You could contact us to add your function to the pleione package.

Note: Installation instructions: Instructions to install KaSim, BioNetGen, NFsim, and PISKaS are available in their source code webpages. Nonetheless, here you will find basic information to clone using git or download the software and install it.

To install SLURM, you should have admin access to your infrastructure and an UNIX-based OS. Detailed instructions are provided here: *Installing SLURM in your machine*

CHAPTER

THREE

COMPILING PYTHON3 FROM SOURCE

If you don't have admin access to the cluster configuration, you could compile and install python3 from source following these instructions:

```
wget https://www.python.org/ftp/python/3.6.5/Python-3.6.5.tgz \
-O ~/opt/ubuntu-software/Python-3.6.5.tgz
if [ -d ~/opt/Python-3.6.5 ]; then rm -rf ~/opt/Python-3.6.5; fi
tar xvzf ~/opt/ubuntu-software/Python-3.6.5.tgz -C ~/opt
cd ~/opt/Python-3.6.5
if [ -f Makefile ]; then make clean; fi
if [ -d $(HOME)/opt/python-3.6.5 ]; then rm -rf $(HOME)/opt/python-3.6.5; fi
./configure --prefix=$(HOME)/opt/python-3.6.5
make
make install
```

Note: Don't copy an installation folder from another machine since there may be libraries incompatibilities. Instead, the code will download, configure, compile, and install. To make accesible from anywhere, you could add an alias into `~/.bashrc` or a symbolic in your `$HOME/bin` folder for `$HOME/opt/python-3.6.5/bin/python3` and `pip3`.

To install numpy and pandas use the following instructions, in order since some pandas dependencies has also dependencies:

```
wget https://files.pythonhosted.org/packages/71/90/
↪ca61e203e0080a8cef7ac21eca199829fa8d997f7c4da3e985b49d0a107d/numpy-1.14.3-cp36-
↪cp36m-manylinux1_x86_64.whl
wget https://files.pythonhosted.org/packages/dc/83/
↪15f7833b70d3e067ca91467ca245bae0f6fe56ddc7451aa0dc5606b120f2/pytz-2018.4-py2.py3-
↪none-any.whl
wget https://files.pythonhosted.org/packages/67/4b/
↪141a581104b1f6397bfa78ac9d43d8ad29a7ca43ea90a2d863fe3056e86a/six-1.11.0-py2.py3-
↪none-any.whl
wget https://files.pythonhosted.org/packages/cf/f5/
↪af2b09c957ace60dcfac112b669c45c8c97e32f94aa8b56da4c6d1682825/python_dateutil-2.7.3-
↪py2.py3-none-any.whl
wget https://files.pythonhosted.org/packages/69/ec/
↪8ff0800b8594691759b78a42ccd616f81e7099ee47b167eb9bbd502c02b9/pandas-0.23.0-cp36-
↪cp36m-manylinux1_x86_64.whl

pip3 install numpy-1.14.3-cp36-cp36m-manylinux1_x86_64.whl
pip3 install pytz-2018.4-py2.py3-none-any.whl
pip3 install six-1.11.0-py2.py3-none-any.whl
```

(continues on next page)

(continued from previous page)

```
pip3 install python_dateutil-2.7.3-py2.py3-none-any.whl
pip3 install pandas-0.23.0-cp36-cp36m-manylinux1_x86_64.whl
```

If you have admin access (and willing to compile python3 from source) you could install the following dependencies:

```
apt-get install libssl-dev zlib1g-dev libncurses5-dev \
libncursesw5-dev libreadline-dev libsqlite3-dev libgdbm-dev \
libdb5.3-dev libbz2-dev libexpat1-dev liblzma-dev tk-dev
```

Compiling python3 with all dependencies would make installation of packages easier. Just follow the instructions:

```
pip3 install pandas
```

Note: Installing pandas with pip will install numpy as its dependency.

Note: Be sure you are calling pip3 after creating an alias or a symbolic link. Without admin credentials, pip3 would fail to install pandas.

CHAPTER

FOUR

INSTALLING SLURM IN YOUR MACHINE

To install SLURM you need admin access to the machine. Please follow this instructions to start up running the workload manager, in the controller as well in the controlled machines.

```
sudo apt-get -y install slurm-wlm
sudo nano /etc/slurm-llnl/slurm.conf

sudo chown -R slurm:slurm /var/run/slurm-llnl/
sudo chown -R slurm:slurm /var/lib/slurm-llnl/
sudo chown -R slurm:slurm /var/log/slurm-llnl/
sudo mkdir /var/spool/slurmd
sudo chown -R slurm:slurm /var/spool/slurmd

sudo systemctl start slurmd
```

Replace `$HOST_NAME` with your machine name that is going to act as the controller. If you have multiple machines, this configuration file must be identical and in all machines in the queue.

```
### slurm.conf - Slurm config file.

#ClusterName=$HOST_NAME
ControlMachine=$HOST_NAME
SlurmUser=slurm
AuthType=auth/munge

SlurmctldPidFile=/var/run/slurm-llnl/slurmctld.pid
SlurmdPidFile=/var/run/slurm-llnl/slurmd.pid
SlurmdSpoolDir=/var/lib/slurm-llnl/slurmd
StateSaveLocation=/var/lib/slurm-llnl/slurmctld

SwitchType=switch/none
ProctrackType=proctrack/pgid
TaskPlugin=task/none

MpiDefault=none
MaxJobCount=100000
MaxArraySize=64000

# TIMERS
SlurmdTimeout=300
InactiveLimit=0
MinJobAge=300
KillWait=30
Waittime=0
```

(continues on next page)

(continued from previous page)

```
# SCHEDULING
SchedulerType=sched/backfill
SelectType=select/cons_res
SelectTypeParameters=CR_Core
FastSchedule=1

# LOGGING
SlurmctldDebug=3
SlurmctldLogFile=/var/log/slurm-llnl/slurmctld.log
SlurmdDebug=3
SlurmdLogFile=/var/log/slurm-llnl/slurmd.log

# COMPUTE NODES

# Here you add the machine hardware configurations
NodeName=$HOST_NAME Procs=8 Boards=1 SocketsPerBoard=1 CoresPerSocket=4␣
↪ThreadsPerCore=2 State=idle

# Here you add the machine(s) to a Partition
PartitionName=MyCluster Nodes=$HOST_NAME Default=yes MaxTime=INFINITE State=up
```

Note: Please refer to SLURM for advance configuration like limiting time, CPUs and RAM for users or groups, to balance load in your cluster.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search