pyPESTO Documentation

Release 0.0.13

The pyPESTO developers

May 03, 2020

User's guide

1 Install and upgrade **3**
- 1.1 Requirements . . . 3
- 1.2 Install from PIP . . . 3
- 1.3 Install from GIT . . . 3
- 1.4 Upgrade . . . 4
- 1.5 Install optional packages . . . 4

2 Examples **5**
- 2.1 Rosenbrock banana . . . 5
- 2.2 Conversion reaction . . . 16
- 2.3 Fixed parameters . . . 19
- 2.4 AMICI Python example "Boehm" . . . 22
- 2.5 Model import using the Petab format . . . 29
- 2.6 Save and load results as HDF5 files . . . 35
- 2.7 A sampler study . . . 53
- 2.8 Download the examples as notebooks . . . 65

3 Contribute **67**
- 3.1 Contribute documentation . . . 67
- 3.2 Contribute tests . . . 67
- 3.3 Contribute code . . . 68

4 Deploy **69**
- 4.1 Versioning scheme . . . 69
- 4.2 Creating a new release . . . 69

5 Objective **71**

6 Problem **105**

7 PEtab **117**

8 Optimize **121**

9 Profile **135**

10 Sampling **143**

11 Visualize **159**

12 Result **171**

13 Engines **181**

14 Startpoint **189**

15 Logging **191**

16 Release notes **193**
16.1 0.0 series 193

17 Authors **197**

18 Contact **199**

19 License **201**

20 Logo **203**

21 Indices and tables **207**

Python Module Index **209**

Index **211**

Version: 0.0.13
Source code: https://github.com/icb-dcm/pypesto

CHAPTER 1

Install and upgrade

1.1 Requirements

This package requires Python 3.6 or later. It is tested on Linux using Travis continuous integration.

1.1.1 I cannot use my system's Python distribution, what now?

Several Python distributions can co-exist on a single system. If you don't have access to a recent Python version via your system's package manager (this might be the case for old operating systems), it is recommended to install the latest version of the Anaconda Python 3 distribution.

Also, there is the possibility to use multiple virtual environments via:

```
python3 -m virtualenv ENV_NAME
source ENV_NAME/bin/activate
```

where ENV_NAME denotes an individual environment name, if you do not want to mess up the system environment.

1.2 Install from PIP

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

```
pip3 install pypesto
```

1.3 Install from GIT

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

```
pip3 install git+https://github.com/icb-dcm/pypesto.git
```

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

```
git clone https://github.com/icb-dcm/pypesto.git
```

and then install from the local repository via:

```
cd pypesto
pip3 install .
```

1.4 Upgrade

If you want to upgrade from an existing previous version, replace `install` by `install --upgrade` in the above commands.

1.5 Install optional packages

- This package includes multiple comfort methods simplyfing its use for parameter estimation for models generated using the toolbox amici. To use AMICI, install it via pip:

  ```
  pip3 install amici
  ```

- This package inherently supports optimization using the dlib toolbox. To use it, install dlib via:

  ```
  pip3 install dlib
  ```

CHAPTER 2

Examples

The following examples cover typical use cases and should help get a better idea of how to use this package:

2.1 Rosenbrock banana

Here, we perform optimization for the Rosenbrock banana function, which does not require an AMICI model. In particular, we try several ways of specifying derivative information.

[1]:
```
import pypesto
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

%matplotlib inline
```

2.1.1 Define the objective and problem

[2]:
```
# first type of objective
objective1 = pypesto.Objective(fun=sp.optimize.rosen,
                               grad=sp.optimize.rosen_der,
                               hess=sp.optimize.rosen_hess)

# second type of objective
def rosen2(x):
    return sp.optimize.rosen(x), sp.optimize.rosen_der(x), sp.optimize.rosen_hess(x)
objective2 = pypesto.Objective(fun=rosen2, grad=True, hess=True)

dim_full = 10
lb = -5 * np.ones((dim_full, 1))
ub = 5 * np.ones((dim_full, 1))
```

(continues on next page)

(continued from previous page)

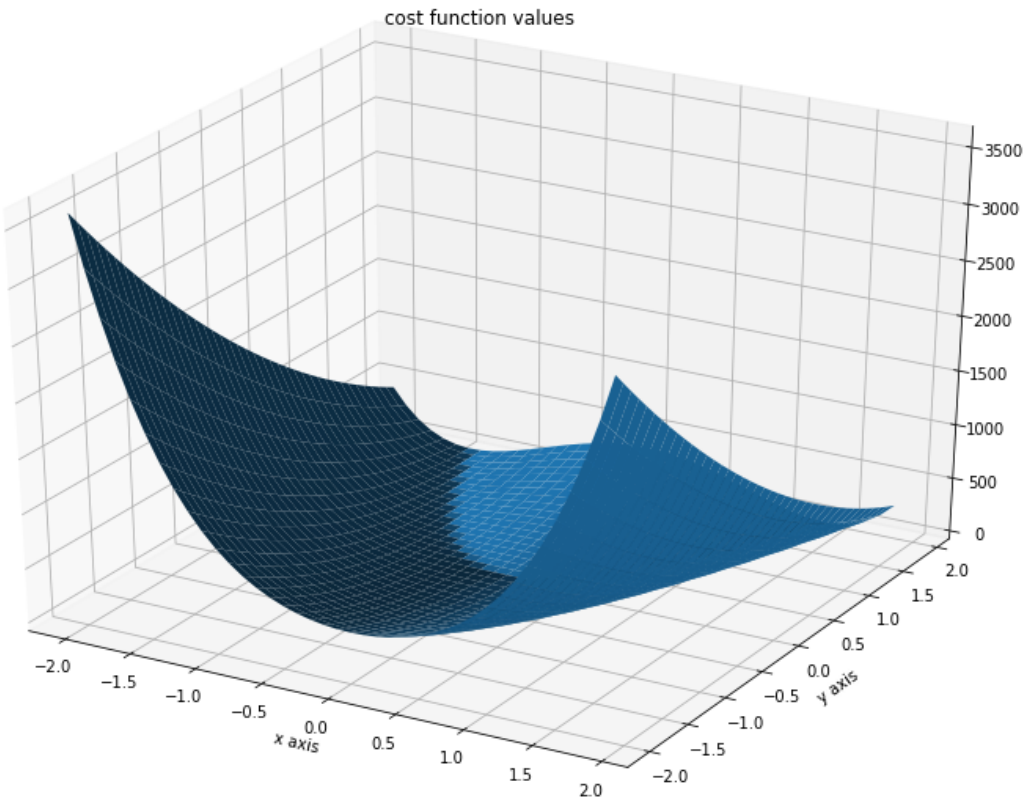
```
problem1 = pypesto.Problem(objective=objective1, lb=lb, ub=ub)
problem2 = pypesto.Problem(objective=objective2, lb=lb, ub=ub)
```

2.1.2 Illustration

[3]:
```
x = np.arange(-2, 2, 0.1)
y = np.arange(-2, 2, 0.1)
x, y = np.meshgrid(x, y)
z = np.zeros_like(x)
for j in range(0, x.shape[0]):
    for k in range(0, x.shape[1]):
        z[j,k] = objective1([x[j,k], y[j,k]], (0,))
```

[4]:
```
fig = plt.figure()
fig.set_size_inches(*(14,10))
ax = plt.axes(projection='3d')
ax.plot_surface(X=x, Y=y, Z=z)
plt.xlabel('x axis')
plt.ylabel('y axis')
ax.set_title('cost function values')
```

[4]:
```
Text(0.5, 0.92, 'cost function values')
```



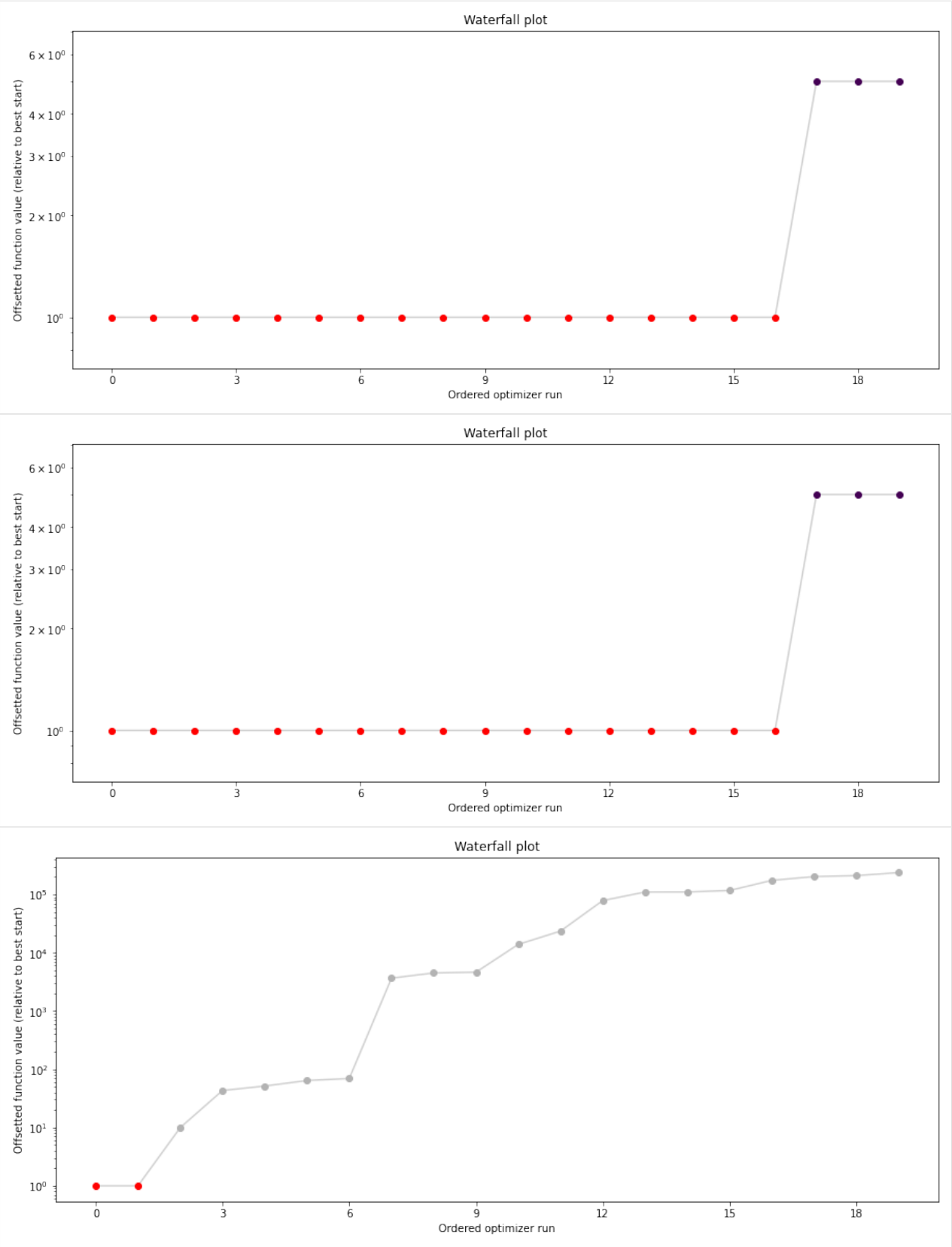
2.1.3 Run optimization

[5]:
```
# create different optimizers
optimizer_bfgs = pypesto.ScipyOptimizer(method='l-bfgs-b')
optimizer_tnc = pypesto.ScipyOptimizer(method='TNC')
optimizer_dogleg = pypesto.ScipyOptimizer(method='dogleg')

# set number of starts
n_starts = 20

# save optimizer trace
history_options = pypesto.HistoryOptions(trace_record=True)

# Run optimizaitons for different optimzers
result1_bfgs = pypesto.minimize(
    problem=problem1, optimizer=optimizer_bfgs,
    n_starts=n_starts, history_options=history_options)
result1_tnc = pypesto.minimize(
    problem=problem1, optimizer=optimizer_tnc,
    n_starts=n_starts, history_options=history_options)
result1_dogleg = pypesto.minimize(
    problem=problem1, optimizer=optimizer_dogleg,
    n_starts=n_starts, history_options=history_options)

# Optimize second type of objective
result2 = pypesto.minimize(problem=problem2, optimizer=optimizer_tnc, n_starts=n_
↪starts)
```

2.1.4 Visualize and compare optimization results

[6]:
```
import pypesto.visualize

# plot separated waterfalls
pypesto.visualize.waterfall(result1_bfgs, size=(15,6))
pypesto.visualize.waterfall(result1_tnc, size=(15,6))
pypesto.visualize.waterfall(result1_dogleg, size=(15,6))
```

[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9397beb90>
```

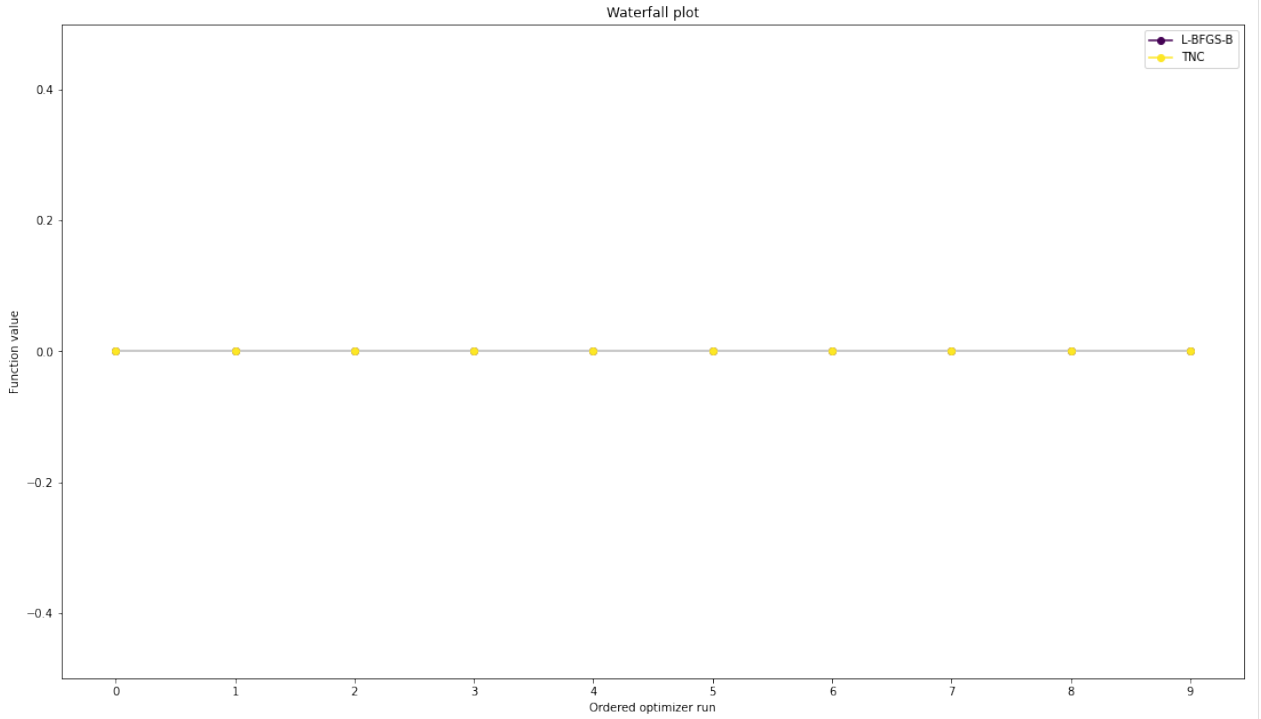


We can now have a closer look, which method perfomred better: Let's first compare bfgs and TNC, since both methods

gave good results. How does the fine convergence look like?

[7]:
```
# plot one list of waterfalls
pypesto.visualize.waterfall([result1_bfgs, result1_tnc],
                            legends=['L-BFGS-B', 'TNC'],
                            start_indices=10,
                            scale_y='lin')
```

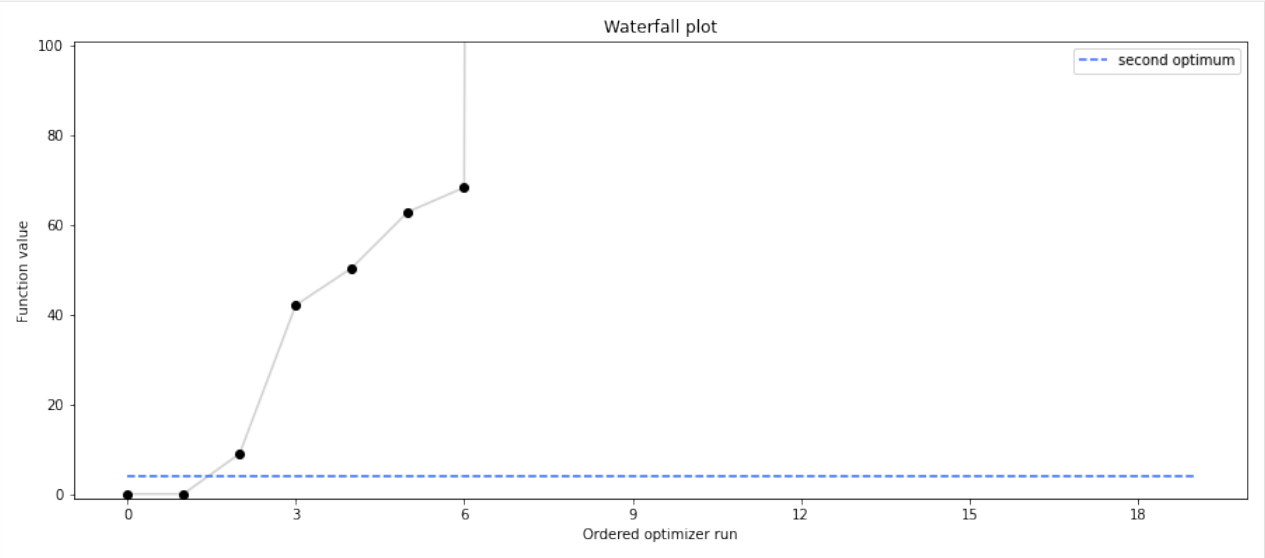
[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd93936a410>
```



[8]:
```
# retrieve second optimum
all_x = result1_bfgs.optimize_result.get_for_key('x')
all_fval = result1_bfgs.optimize_result.get_for_key('fval')
x = all_x[19]
fval = all_fval[19]
print('Second optimum at: ' + str(fval))

# create a reference point from it
ref = {'x': x, 'fval': fval, 'color': [
    0.2, 0.4, 1., 1.], 'legend': 'second optimum'}
ref = pypesto.visualize.create_references(ref)

# new waterfall plot with reference point for second optimum
pypesto.visualize.waterfall(result1_dogleg, size=(15,6),
                            scale_y='lin', y_limits=[-1, 101],
                            reference=ref, colors=[0., 0., 0., 1.])
```

```
Second optimum at: 3.9865791124344874
```

[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9393120d0>
```

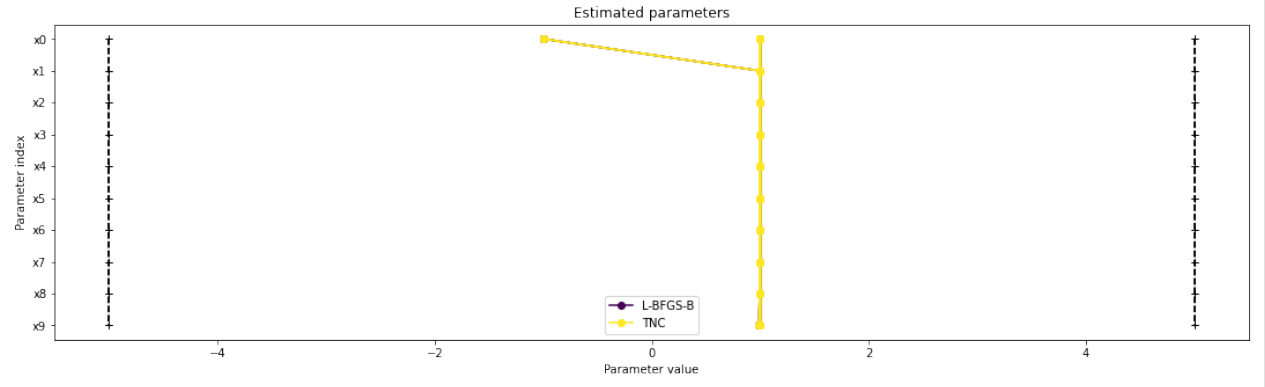


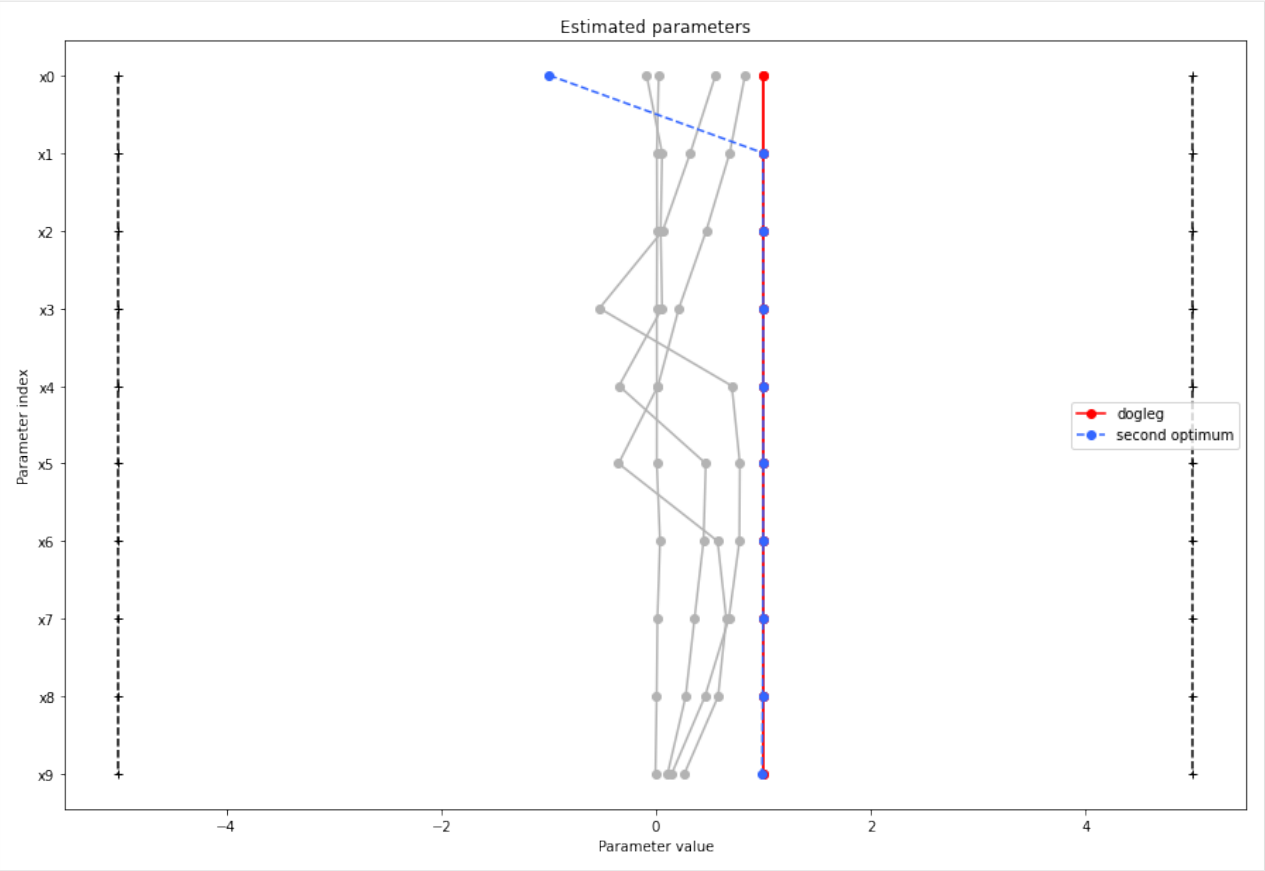
2.1.5 Visualize parameters

There seems to be a second local optimum. We want to see whether it was also found by the dogleg method

[9]:
```
pypesto.visualize.parameters([result1_bfgs, result1_tnc],
                             legends=['L-BFGS-B', 'TNC'],
                             balance_alpha=False)
pypesto.visualize.parameters(result1_dogleg,
                              legends='dogleg',
                              reference=ref,
                              size=(15,10),
                              start_indices=[0, 1, 2, 3, 4, 5],
                              balance_alpha=False)
```

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9394ccf10>
```



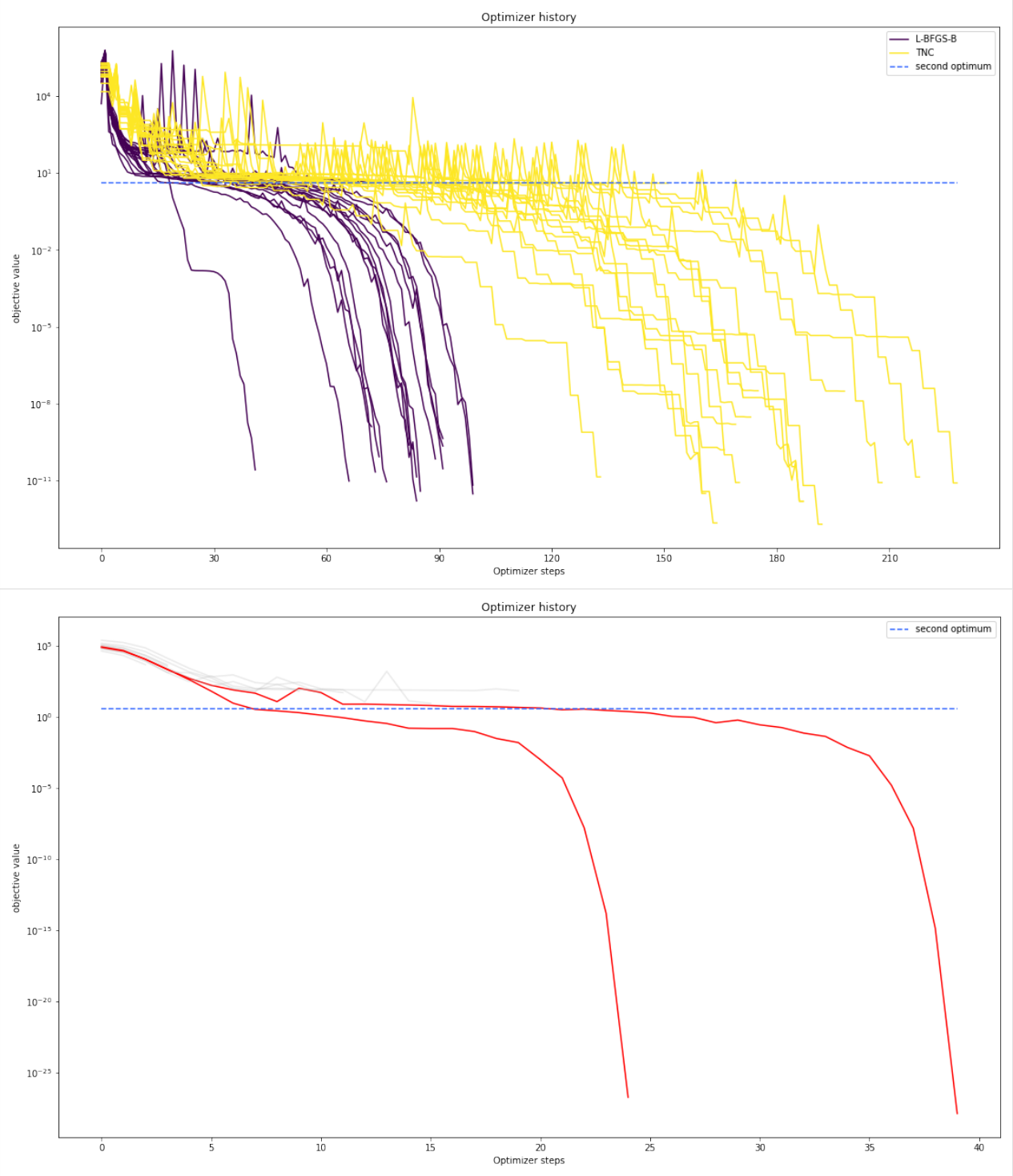


2.1.6 Optimizer history

Let's compare optimzer progress over time.

[10]:
```
# plot one list of waterfalls
pypesto.visualize.optimizer_history([result1_bfgs, result1_tnc],
                                    legends=['L-BFGS-B', 'TNC'],
                                    reference=ref)
# plot one list of waterfalls
pypesto.visualize.optimizer_history(result1_dogleg,
                                    reference=ref)
```

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd93983b750>
```



We can also visualize this usign other scalings or offsets. . .

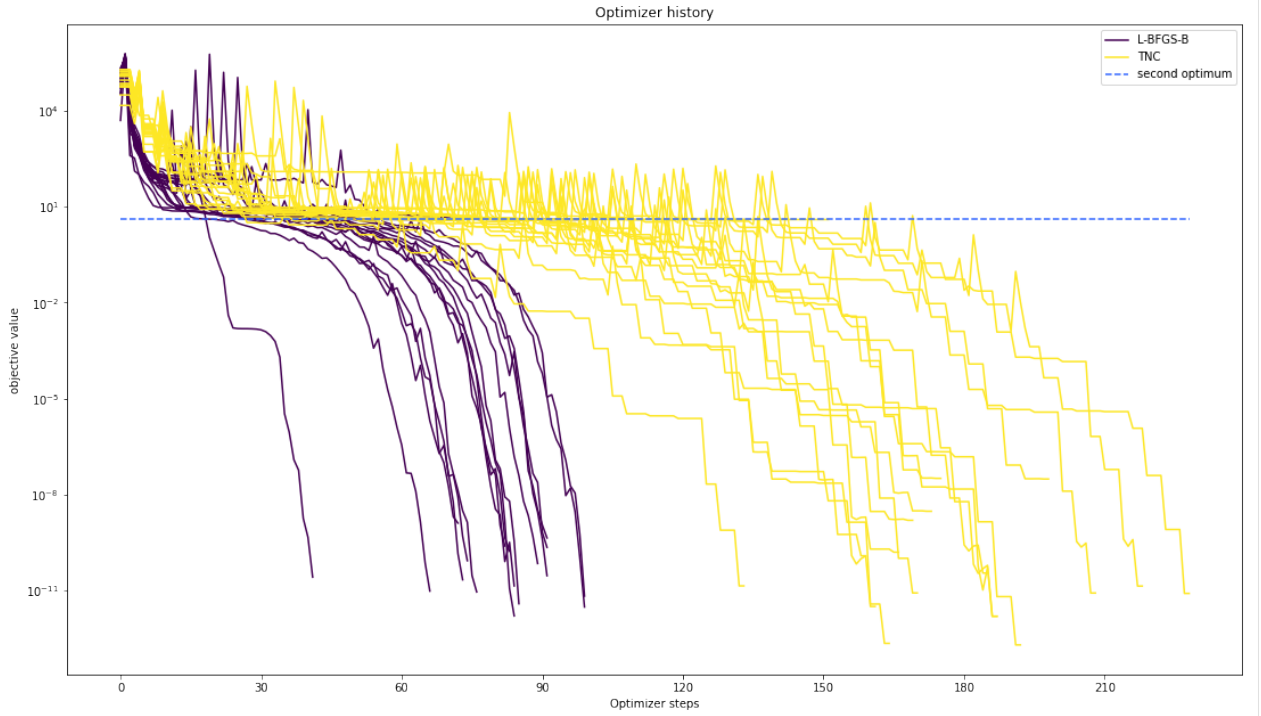
[11]:
```
# plot one list of waterfalls
pypesto.visualize.optimizer_history([result1_bfgs, result1_tnc],
                                    legends=['L-BFGS-B', 'TNC'],
                                    reference=ref,
```

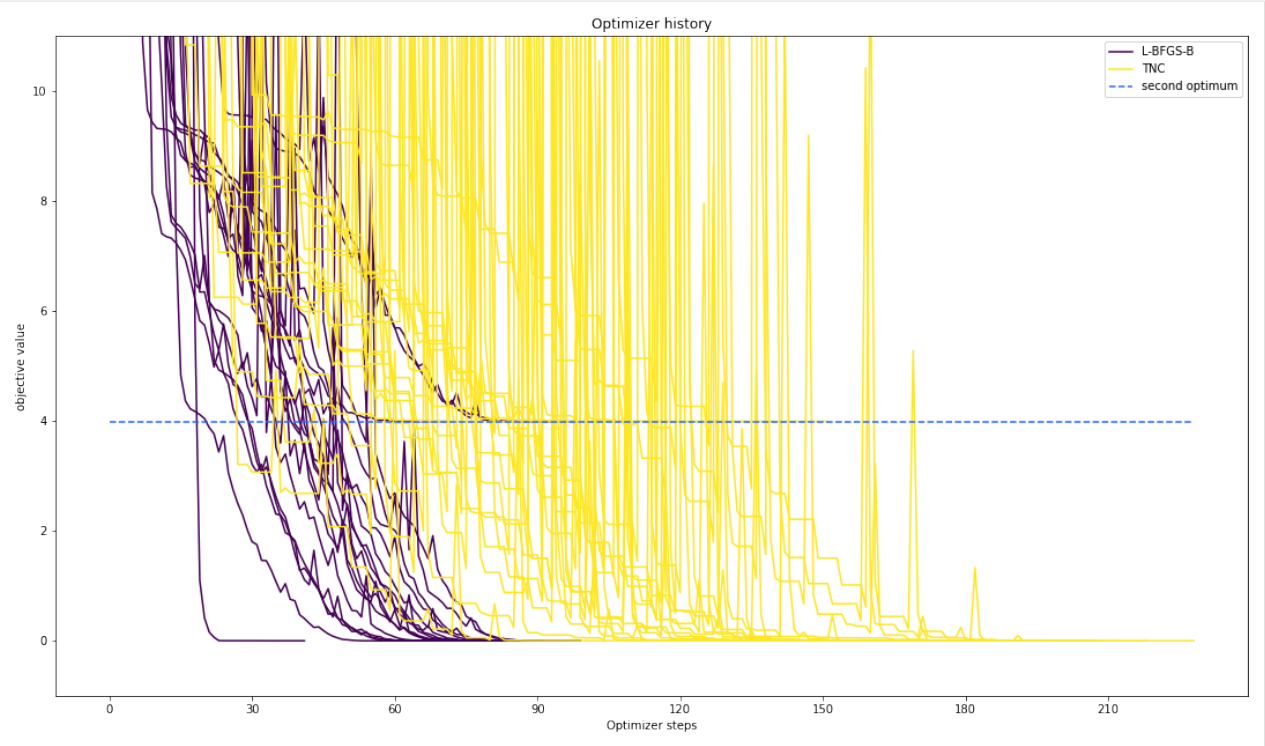
(continues on next page)

(continued from previous page)

```
                                    offset_y=0.)

# plot one list of waterfalls
pypesto.visualize.optimizer_history([result1_bfgs, result1_tnc],
                                    legends=['L-BFGS-B', 'TNC'],
                                    reference=ref,
                                    scale_y='lin',
                                    y_limits=[-1., 11.])
```

[11]: `<matplotlib.axes._subplots.AxesSubplot at 0x7fd9390d9690>`





2.1.7 Compute profiles

The profiling routine needs a problem, a results object and an optimizer.

Moreover it accepts an index of integer (profile_index), whether or not a profile should be computed.

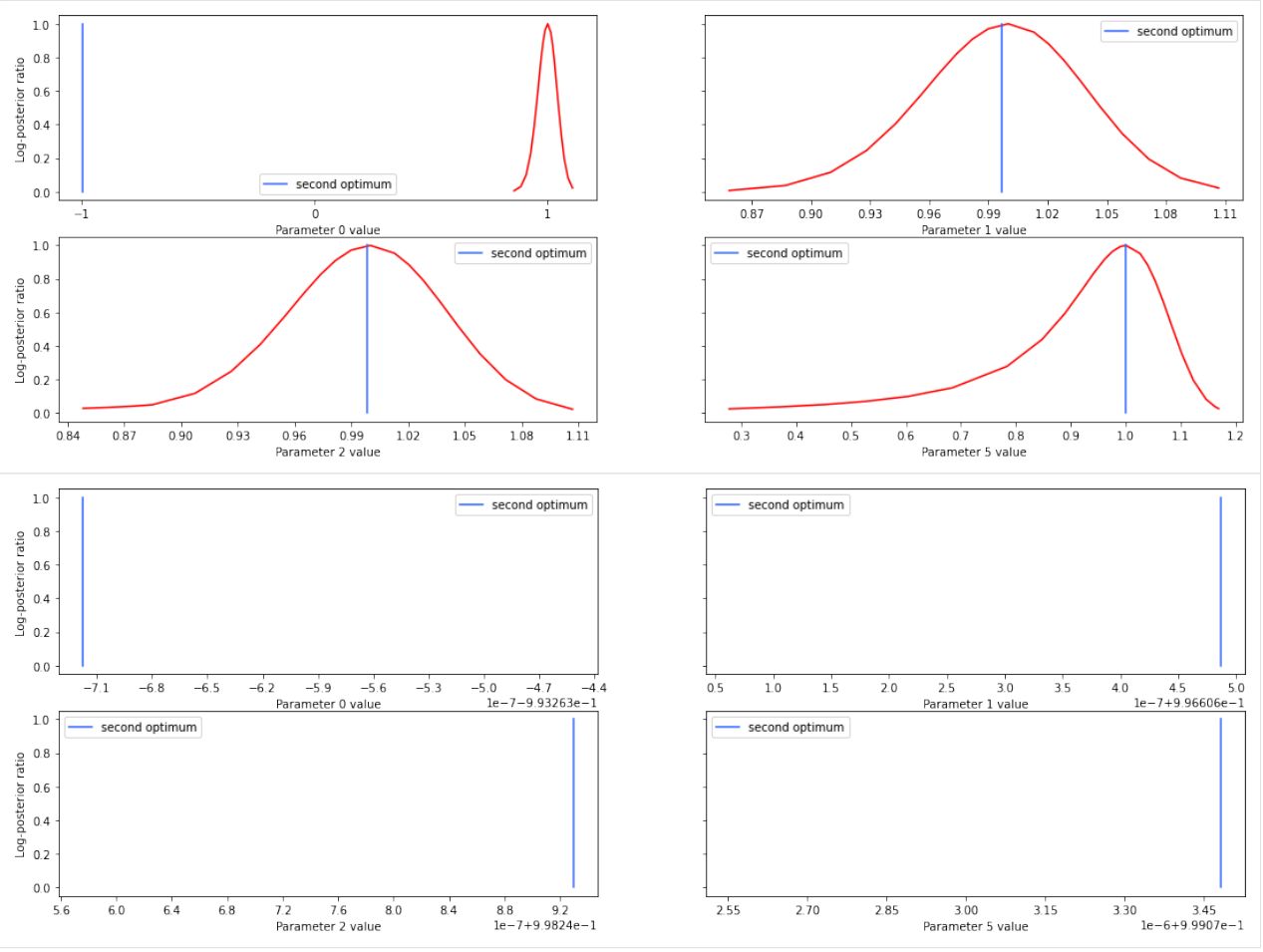
Finally, an integer (result_index) can be passed, in order to specify the local optimum, from which profiling should be started.

[12]:
```
# compute profiles
profile_options = pypesto.ProfileOptions(min_step_size=0.0005,
    delta_ratio_max=0.05,
    default_step_size=0.005,
    ratio_min=0.03)

result1_tnc = pypesto.parameter_profile(
    problem=problem1,
    result=result1_tnc,
    optimizer=optimizer_tnc,
    profile_index=np.array([1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0]),
    result_index=0,
    profile_options=profile_options)

# compute profiles from second optimum
result1_tnc = pypesto.parameter_profile(
    problem=problem1,
    result=result1_tnc,
    optimizer=optimizer_tnc,
    profile_index=np.array([1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0]),
    result_index=19,
    profile_options=profile_options)
```

2.1.8 Visualize and analyze results

pypesto offers easy-to-use visualization routines:

[13]:
```
# specify the parameters, for which profiles should be computed
ax = pypesto.visualize.profiles(result1_tnc, profile_indices = [0,1,2,5],
                                reference=ref, profile_list_id=0)
# plot profiles again, now from second optimum
ax = pypesto.visualize.profiles(result1_tnc, profile_indices = [0,1,2,5],
                                reference=ref, profile_list_id=1)
```



If the result needs to be examined in more detail, it can easily be exported as a pandas.DataFrame:

[14]:
```
result1_tnc.optimize_result.as_dataframe(['fval', 'n_fval', 'n_grad',
                                          'n_hess', 'n_res', 'n_sres', 'time'])
```

[14]:

	fval	n_fval	n_grad	n_hess	n_res	n_sres	time
0	1.968227e-13	193	193	0	0	0	0.018201
1	2.202262e-13	165	165	0	0	0	0.039069
2	1.550811e-12	188	188	0	0	0	0.017980
3	1.553846e-12	188	188	0	0	0	0.017905
4	3.138476e-12	162	162	0	0	0	0.015469
5	8.042668e-12	229	229	0	0	0	0.021637
6	8.268731e-12	209	209	0	0	0	0.049976
7	8.310174e-12	171	171	0	0	0	0.016296
8	1.364149e-11	219	219	0	0	0	0.021103

(continues on next page)

(continued from previous page)

```
9   1.382298e-11     134     134       0       0       0  0.013395
10  3.487863e-11     186     186       0       0       0  0.017843
11  1.211274e-10     160     160       0       0       0  0.042303
12  1.568336e-10     167     167       0       0       0  0.016380
13  1.557791e-09     170     170       0       0       0  0.016406
14  2.989273e-09     174     174       0       0       0  0.017172
15  3.045916e-08     199     199       0       0       0  0.026230
16  3.194012e-08     176     176       0       0       0  0.033760
17  3.986579e+00     134     134       0       0       0  0.013941
18  3.986579e+00     152     152       0       0       0  0.017177
19  3.986579e+00     103     103       0       0       0  0.009830
```

2.2 Conversion reaction

[1]:
```
import importlib
import os
import sys
import numpy as np
import amici
import amici.plotting
import pypesto

# sbml file we want to import
sbml_file = 'conversion_reaction/model_conversion_reaction.xml'
# name of the model that will also be the name of the python module
model_name = 'model_conversion_reaction'
# directory to which the generated model code is written
model_output_dir = 'tmp/' + model_name
```

2.2.1 Compile AMICI model

[2]:
```
# import sbml model, compile and generate amici module
sbml_importer = amici.SbmlImporter(sbml_file)
sbml_importer.sbml2amici(model_name,
                         model_output_dir,
                         verbose=False)
```

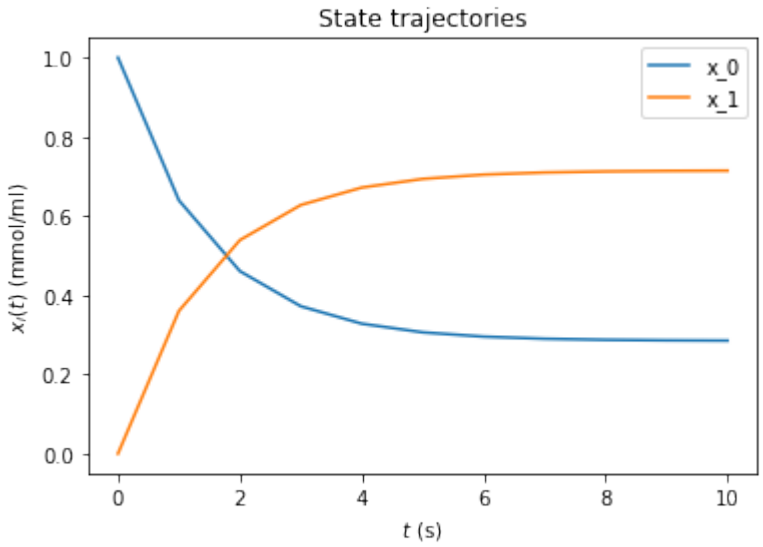
2.2.2 Load AMICI model

[3]:
```
# load amici module (the usual starting point later for the analysis)
sys.path.insert(0, os.path.abspath(model_output_dir))
model_module = importlib.import_module(model_name)
model = model_module.getModel()
model.requireSensitivitiesForAllParameters()
model.setTimepoints(amici.DoubleVector(np.linspace(0, 10, 11)))
model.setParameterScale(amici.ParameterScaling_log10)
model.setParameters(amici.DoubleVector([-0.3,-0.7]))
solver = model.getSolver()
solver.setSensitivityMethod(amici.SensitivityMethod_forward)
solver.setSensitivityOrder(amici.SensitivityOrder_first)
```

(continues on next page)

(continued from previous page)

```
# how to run amici now:
rdata = amici.runAmiciSimulation(model, solver, None)
amici.plotting.plotStateTrajectories(rdata)
edata = amici.ExpData(rdata, 0.2, 0.0)
```



2.2.3 Optimize

[4]:
```
# create objective function from amici model
# pesto.AmiciObjective is derived from pesto.Objective,
# the general pesto objective function class
objective = pypesto.AmiciObjective(model, solver, [edata], 1)

# create optimizer object which contains all information for doing the optimization
optimizer = pypesto.ScipyOptimizer(method='ls_trf')

#optimizer.solver = 'bfgs|meigo'
# if select meigo -> also set default values in solver_options
#optimizer.options = {'maxiter': 1000, 'disp': True} # = pesto.default_options_meigo()
#optimizer.startpoints = []
#optimizer.startpoint_method = 'lhs|uniform|something|function'
#optimizer.n_starts = 100

# see PestoOptions.m for more required options here
# returns OptimizationResult, see parameters.MS for what to return
# list of final optim results foreach multistart, times, hess, grad,
# flags, meta information (which optimizer -> optimizer.get_repr())

# create problem object containing all information on the problem to be solved
problem = pypesto.Problem(objective=objective,
                          lb=[-2,-2], ub=[2,2])

# maybe lb, ub = inf
# other constraints: kwargs, class pesto.Constraints
# constraints on pams, states, esp. pesto.AmiciConstraints (e.g. pam1 + pam2<= const)
```

(continues on next page)

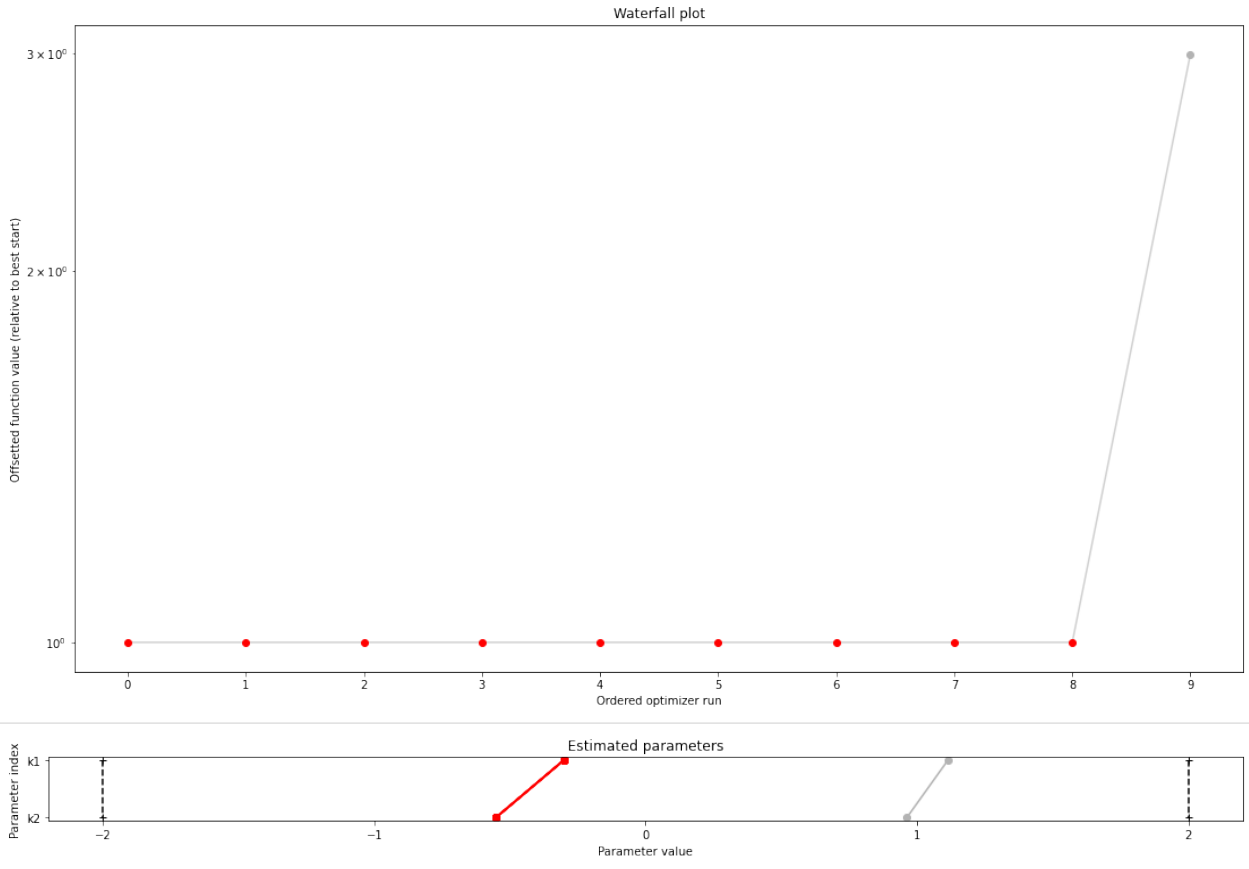
(continued from previous page)

```
# if optimizer cannot handle -> error
# maybe also scaling / transformation of parameters encoded here

# do the optimization
result = pypesto.minimize(problem=problem,
                          optimizer=optimizer,
                          n_starts=10)
# optimize is a function since it does not need an internal memory,
# just takes input and returns output in the form of a Result object
# 'result' parameter: e.g. some results from somewhere -> pick best start points
```

2.2.4 Visualize

[5]:
```
# waterfall, parameter space, scatter plots, fits to data
# different functions for different plotting types
import pypesto.visualize

pypesto.visualize.waterfall(result)
pypesto.visualize.parameters(result)
```

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f48849e1150>
```



2.2.5 Data storage

[6]:
```
# result = pypesto.storage.load('db_file.db')
```

2.2.6 Profiles

[7]:
```
# there are three main parts: optimize, profile, sample. the overall structure of␣
↪profiles and sampling
# will be similar to optimizer like above.
# we intend to only have just one result object which can be reused everywhere, but␣
↪the problem of how to
# not have one huge class but
# maybe simplified views on it for optimization, profiles and sampling is still to be␣
↪solved

# profiler = pypesto.Profiler()

# result = pypesto.profile(problem, profiler, result=None)
# possibly pass result object from optimization to get good parameter guesses
```

2.2.7 Sampling

[8]:
```
# sampler = pypesto.Sampler()

# result = pypesto.sample(problem, sampler, result=None)
```

[9]:
```
# open: how to parallelize. the idea is to use methods similar to those in pyabc for␣
↪working on clusters.
# one way would be to specify an additional 'engine' object passed to optimize(),␣
↪profile(), sample(),
# which in the default setting just does a for loop, but can also be customized.
```

2.3 Fixed parameters

In this notebook we will show how to use fixed parameters. Therefore, we employ our Rosenbrock example. We define two problems, where for the first problem all parameters are optimized, and for the second we fix some of them to specified values.

2.3.1 Define problem

[1]:
```
import pypesto
import pypesto.visualize
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

%matplotlib inline
```

[2]:
```
objective = pypesto.Objective(fun=sp.optimize.rosen,
                              grad=sp.optimize.rosen_der,
                              hess=sp.optimize.rosen_hess)

dim_full = 5
lb = -2 * np.ones((dim_full,1))
ub = 2 * np.ones((dim_full,1))

problem1 = pypesto.Problem(objective=objective, lb=lb, ub=ub)

x_fixed_indices = [1, 3]
x_fixed_vals = [1, 1]
problem2 = pypesto.Problem(objective=objective, lb=lb, ub=ub,
                           x_fixed_indices=x_fixed_indices,
                           x_fixed_vals=x_fixed_vals)
```

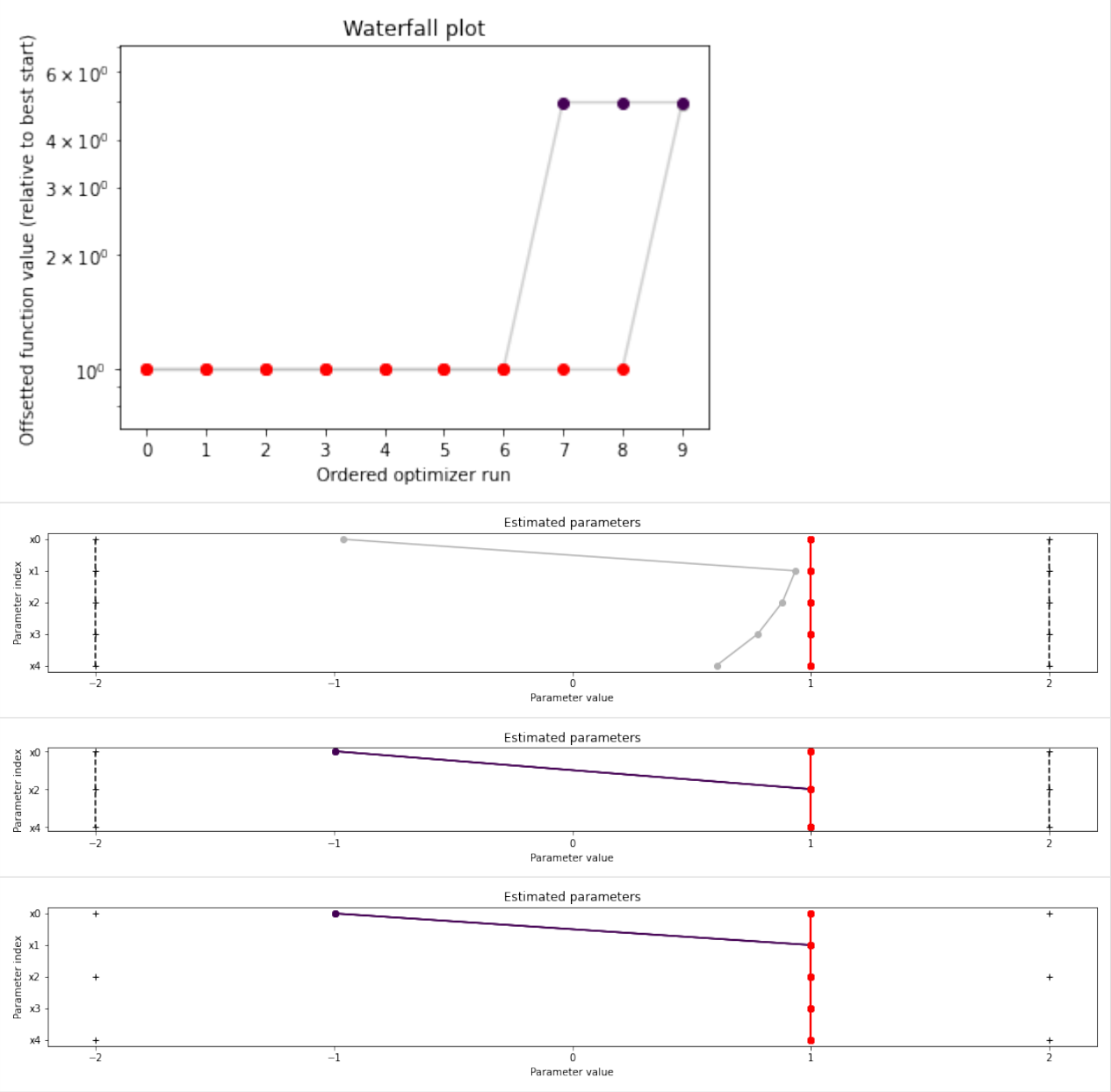
2.3.2 Optimize

[3]:
```
optimizer = pypesto.ScipyOptimizer()
n_starts = 10

result1 = pypesto.minimize(problem=problem1, optimizer=optimizer,
                           n_starts=n_starts)
result2 = pypesto.minimize(problem=problem2, optimizer=optimizer,
                           n_starts=n_starts)
```

2.3.3 Visualize

[4]:
```
fig, ax = plt.subplots()
pypesto.visualize.waterfall(result1, ax)
pypesto.visualize.waterfall(result2, ax)
pypesto.visualize.parameters(result1)
pypesto.visualize.parameters(result2)
pypesto.visualize.parameters(result2, free_indices_only=False)
```

[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe98fe12650>
```



[5]:
```
result1.optimize_result.as_dataframe(['fval', 'x', 'grad'])
```

[5]:

	fval	x	\
0	4.689854e-14	[0.9999999815635604, 0.9999999601303594, 0.999...	
1	5.590260e-14	[0.9999999788673749, 0.9999999710531022, 0.999...	
2	3.799999e-13	[0.9999999894531796, 0.9999999867856568, 1.000...	
3	4.110300e-13	[1.0000000452200102, 1.0000000894442573, 1.000...	
4	6.110801e-13	[0.9999999640563018, 0.9999999933489752, 0.999...	
5	1.231774e-12	[1.0000000581023145, 1.000000113974549, 1.0000...	
6	6.918491e-12	[1.0000000005661727, 1.0000001076896814, 1.000...	
7	2.552430e-11	[0.9999999192005172, 1.0000000369812814, 1.000...	
8	2.855055e-11	[0.9999995268339968, 0.9999992696810054, 0.999...	
9	3.930839e+00	[-0.9620508371994185, 0.9357391790840979, 0.88...	

(continues on next page)

(continued from previous page)

```
                                                grad
0  [1.1618317897690609e-06, 1.2817350405303142e-0...
1  [-5.3696059283362676e-06, -3.17862832097996e-0...
2  [-3.1728126266944613e-06, -1.6951548081005115e...
3  [4.887460487692825e-07, 3.7999227735179252e-06...
4  [-2.6166434580847274e-05, 8.102944813282283e-0...
5  [1.0082380613547457e-06, 1.8071346123806702e-0...
6  [-4.262180210516511e-05, 7.700439519900304e-05...
7  [-7.959368869249685e-05, -8.749066771695052e-0...
8  [-8.735140622961871e-05, 4.373181014008344e-05...
9  [5.2019939965397555e-05, 2.44790688288532e-05,...
```

[6]:
```
result2.optimize_result.as_dataframe(['fval', 'x', 'grad'])
```

[6]:
```
           fval                                                  x  \
0  3.757636e-21  [0.9999999999987625, 1.0, 1.0000000000008613, ...
1  3.403187e-16  [1.000000000296693, 1.0, 0.9999999993226122, 1...
2  1.078998e-15  [0.9999999994275696, 1.0, 0.9999999986532816, ...
3  1.986582e-15  [0.9999999981059375, 1.0, 0.9999999990468542, ...
4  1.777511e-14  [1.0000000013724608, 1.0, 0.9999999967022452, ...
5  2.344376e-14  [0.9999999959141576, 1.0, 0.9999999996553852, ...
6  2.096868e-13  [0.9999999984695723, 1.0, 1.0000000066163943, ...
7  3.989975e+00  [-0.9949747468749881, 1.0, 0.9999999997811084,...
8  3.989975e+00  [-0.9949747444243423, 1.0, 1.000000008470282, ...
9  3.989975e+00  [-0.9949747299382655, 1.0, 1.0000000336260297,...

                                                grad
0  [-9.92438575763029e-10, nan, 8.630336445497983...
1  [2.3794769983646036e-07, nan, -6.7874258776674...
2  [-4.5908920971880386e-07, nan, -1.349411838817...
3  [-1.5190380919959983e-06, nan, -9.550520753635...
4  [1.100713572463121e-06, nan, -3.30435034796972...
5  [-3.2768456252559055e-06, nan, -3.453040106231...
6  [-1.2274029922568326e-06, nan, 6.6296271449766...
7  [-3.507845658390352e-08, nan, -2.1932935463983...
8  [1.900857428349667e-06, nan, 8.487222652784575...
9  [1.334441878420023e-05, nan, 3.369328314657676...
```

2.4 AMICI Python example “Boehm”

This is an example using the model [boehm_ProteomeRes2014.xml] model to demonstrate and test SBML import and AMICI Python interface.

[1]:
```
import libsbml
import importlib
import amici
import pypesto
import os
import sys
import numpy as np
import matplotlib.pyplot as plt

# temporarily add the simulate file
sys.path.insert(0, 'boehm_JProteomeRes2014')
```

(continues on next page)

(continued from previous page)

```
from benchmark_import import DataProvider

# sbml file
sbml_file = 'boehm_JProteomeRes2014/boehm_JProteomeRes2014.xml'

# name of the model that will also be the name of the python module
model_name = 'boehm_JProteomeRes2014'

# output directory
model_output_dir = 'tmp/' + model_name
```

2.4.1 The example model

Here we use `libsbml` to show the reactions and species described by the model (this is independent of AMICI).

[2]:
```
sbml_reader = libsbml.SBMLReader()
sbml_doc = sbml_reader.readSBML(os.path.abspath(sbml_file))
sbml_model = sbml_doc.getModel()
dir(sbml_doc)
print(os.path.abspath(sbml_file))
print('Species: ', [s.getId() for s in sbml_model.getListOfSpecies()])


print('\nReactions:')
for reaction in sbml_model.getListOfReactions():
    reactants = ' + '.join(['%s %s'%(int(r.getStoichiometry()) if r.
↪getStoichiometry() > 1 else '', r.getSpecies()) for r in reaction.
↪getListOfReactants()])
    products  = ' + '.join(['%s %s'%(int(r.getStoichiometry()) if r.
↪getStoichiometry() > 1 else '', r.getSpecies()) for r in reaction.
↪getListOfProducts()])
    reversible = '<' if reaction.getReversible() else ''
    print('%3s: %10s %1s->%10s\t\t[%s]' % (reaction.getId(),
                        reactants,
                        reversible,
                         products,
                        libsbml.formulaToL3String(reaction.getKineticLaw().
↪getMath())))
```

```
/home/yannik/pypesto/doc/example/boehm_JProteomeRes2014/boehm_JProteomeRes2014.xml
Species:  ['STAT5A', 'STAT5B', 'pApB', 'pApA', 'pBpB', 'nucpApA', 'nucpApB', 'nucpBpB
↪']

Reactions:
v1_v_0:   2 STAT5A  ->      pApA              [cyt * BaF3_Epo * STAT5A^2 * k_phos]
v2_v_1:  STAT5A +  STAT5B  ->      pApB               [cyt * BaF3_Epo * STAT5A *␣
↪STAT5B * k_phos]
v3_v_2:   2 STAT5B  ->      pBpB              [cyt * BaF3_Epo * STAT5B^2 * k_phos]
v4_v_3:       pApA  ->   nucpApA              [cyt * k_imp_homo * pApA]
v5_v_4:       pApB  ->   nucpApB              [cyt * k_imp_hetero * pApB]
v6_v_5:       pBpB  ->   nucpBpB              [cyt * k_imp_homo * pBpB]
v7_v_6:    nucpApA  ->  2 STAT5A              [nuc * k_exp_homo * nucpApA]
v8_v_7:    nucpApB  -> STAT5A +  STAT5B               [nuc * k_exp_hetero * nucpApB]
v9_v_8:    nucpBpB  ->  2 STAT5B              [nuc * k_exp_homo * nucpBpB]
```

2.4.2 Importing an SBML model, compiling and generating an AMICI module

Before we can use AMICI to simulate our model, the SBML model needs to be translated to C++ code. This is done by `amici.SbmlImporter`.

[3]:
```
# Create an SbmlImporter instance for our SBML model
sbml_importer = amici.SbmlImporter(sbml_file)
```

In this example, we want to specify fixed parameters, observables and a σ parameter. Unfortunately, the latter two are not part of the SBML standard. However, they can be provided to `amici.SbmlImporter.sbml2amici` as demonstrated in the following.

Constant parameters

Constant parameters, i.e. parameters with respect to which no sensitivities are to be computed (these are often parameters specifying a certain experimental condition) are provided as a list of parameter names.

[4]:
```
constantParameters = {'ratio', 'specC17'}
```

Observables

We used SBML's `AssignmentRule <http://sbml.org/Software/libSBML/5.13.0/docs//python-api/classlibsbml_1_1_rule.html>`__ as a non-standard way to specify *Model outputs* within the SBML file. These rules need to be removed prior to the model import (AMICI does at this time not support these Rules). This can be easily done using `amici.assignmentRules2observables()`.

In this example, we introduced parameters named `observable_*` as targets of the observable AssignmentRules. Where applicable we have `observable_*_sigma` parameters for σ parameters (see below).

[5]:
```
# Retrieve model output names and formulae from AssignmentRules and remove the␣
↪respective rules
observables = amici.assignmentRules2observables(
        sbml_importer.sbml, # the libsbml model object
        filter_function=lambda variable: variable.getId().startswith('observable_')␣
↪and not variable.getId().endswith('_sigma')
    )
print('Observables:', observables)
```

```
Observables: {'observable_pSTAT5A_rel': {'name': '', 'formula': '(100 * pApB + 200 *␣
↪pApA * specC17) / (pApB + STAT5A * specC17 + 2 * pApA * specC17)'}, 'observable_
↪pSTAT5B_rel': {'name': '', 'formula': '-(100 * pApB - 200 * pBpB * (specC17 - 1)) /␣
↪(STAT5B * (specC17 - 1) - pApB + 2 * pBpB * (specC17 - 1))'}, 'observable_rSTAT5A_
↪rel': {'name': '', 'formula': '(100 * pApB + 100 * STAT5A * specC17 + 200 * pApA *␣
↪specC17) / (2 * pApB + STAT5A * specC17 + 2 * pApA * specC17 - STAT5B * (specC17 -␣
↪1) - 2 * pBpB * (specC17 - 1))'}}
```

σ parameters

To specify measurement noise as a parameter, we simply provide a dictionary with (preexisting) parameter names as keys and a list of observable names as values to indicate which sigma parameter is to be used for which observable.

[6]:
```
sigma_vals = ['sd_pSTAT5A_rel', 'sd_pSTAT5B_rel', 'sd_rSTAT5A_rel']
observable_names = observables.keys()
```

(continues on next page)

(continued from previous page)

```
sigmas = dict(zip(list(observable_names), sigma_vals))
print(sigmas)
```

```
{'observable_pSTAT5A_rel': 'sd_pSTAT5A_rel', 'observable_pSTAT5B_rel': 'sd_pSTAT5B_rel
↪', 'observable_rSTAT5A_rel': 'sd_rSTAT5A_rel'}
```

Generating the module

Now we can generate the python module for our model. `amici.SbmlImporter.sbml2amici` will symbolically derive the sensitivity equations, generate C++ code for model simulation, and assemble the python module.

[7]:
```
sbml_importer.sbml2amici(model_name,
                         model_output_dir,
                         verbose=False,
                         observables=observables,
                         constantParameters=constantParameters,
                         sigmas=sigmas
  )
```

Importing the module and loading the model

If everything went well, we need to add the previously selected model output directory to our PYTHON_PATH and are then ready to load newly generated model:

[8]:
```
sys.path.insert(0, os.path.abspath(model_output_dir))
model_module = importlib.import_module(model_name)
```

And get an instance of our model from which we can retrieve information such as parameter names:

[9]:
```
model = model_module.getModel()

print("Model parameters:", list(model.getParameterIds()))
print("Model outputs:   ", list(model.getObservableIds()))
print("Model states:    ", list(model.getStateIds()))
```

```
Model parameters: ['Epo_degradation_BaF3', 'k_exp_hetero', 'k_exp_homo', 'k_imp_hetero
↪', 'k_imp_homo', 'k_phos', 'sd_pSTAT5A_rel', 'sd_pSTAT5B_rel', 'sd_rSTAT5A_rel']
Model outputs:    ['observable_pSTAT5A_rel', 'observable_pSTAT5B_rel', 'observable_
↪rSTAT5A_rel']
Model states:     ['STAT5A', 'STAT5B', 'pApB', 'pApA', 'pBpB', 'nucpApA', 'nucpApB',
↪'nucpBpB']
```

2.4.3 Running simulations and analyzing results

After importing the model, we can run simulations using `amici.runAmiciSimulation`. This requires a `Model` instance and a `Solver` instance. Optionally you can provide measurements inside an `ExpData` instance, as shown later in this notebook.

[10]:
```
h5_file = 'boehm_JProteomeRes2014/data_boehm_JProteomeRes2014.h5'
dp = DataProvider(h5_file)
```

[11]:
```
# set timepoints for which we want to simulate the model
timepoints = amici.DoubleVector(dp.get_timepoints())
model.setTimepoints(timepoints)

# set fixed parameters for which we want to simulate the model
model.setFixedParameters(amici.DoubleVector(np.array([0.693, 0.107])))

# set parameters to optimal values found in the benchmark collection
model.setParameterScale(2)
model.setParameters(amici.DoubleVector(np.array([-1.568917588,
-4.999704894,
-2.209698782,
-1.786006548,
4.990114009,
4.197735488,
0.585755271,
0.818982819,
0.498684404
])))

# Create solver instance
solver = model.getSolver()

# Run simulation using model parameters from the benchmark collection and default␣
↪solver options
rdata = amici.runAmiciSimulation(model, solver)
```

[12]:
```
# Create edata
edata = amici.ExpData(rdata, 1.0, 0)

# set observed data
edata.setObservedData(amici.DoubleVector(dp.get_measurements()[0][:, 0]), 0)
edata.setObservedData(amici.DoubleVector(dp.get_measurements()[0][:, 1]), 1)
edata.setObservedData(amici.DoubleVector(dp.get_measurements()[0][:, 2]), 2)

# set standard deviations to optimal values found in the benchmark collection
edata.setObservedDataStdDev(amici.DoubleVector(np.array(16*[10**0.585755271])), 0)
edata.setObservedDataStdDev(amici.DoubleVector(np.array(16*[10**0.818982819])), 1)
edata.setObservedDataStdDev(amici.DoubleVector(np.array(16*[10**0.498684404])), 2)
```

[13]:
```
rdata = amici.runAmiciSimulation(model, solver, edata)

print('Chi2 value reported in benchmark collection: 47.9765479')
print('chi2 value using AMICI:')
print(rdata['chi2'])
```

```
Chi2 value reported in benchmark collection: 47.9765479
chi2 value using AMICI:
47.97654266893465
```

2.4.4 Run optimization using pyPESTO

[14]:
```
# create objective function from amici model
# pesto.AmiciObjective is derived from pesto.Objective,
# the general pesto objective function class
```

(continues on next page)

(continued from previous page)

```
model.requireSensitivitiesForAllParameters()


solver.setSensitivityMethod(amici.SensitivityMethod_forward)
solver.setSensitivityOrder(amici.SensitivityOrder_first)


objective = pypesto.AmiciObjective(model, solver, [edata], 1)
```

[15]:
```
# create optimizer object which contains all information for doing the optimization
optimizer = pypesto.ScipyOptimizer()

optimizer.solver = 'bfgs'
```

[16]:
```
# create problem object containing all information on the problem to be solved
x_names = ['x' + str(j) for j in range(0, 9)]
problem = pypesto.Problem(objective=objective,
                          lb=-5*np.ones((9)), ub=5*np.ones((9)),
                          x_names=x_names)
```

[17]:
```
# do the optimization
result = pypesto.minimize(problem=problem,
                          optimizer=optimizer,
                          n_starts=10)  # 200
```

```
[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 221.821 and h = 3.00478e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 221.821149:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 221.821 and h = 3.00478e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 221.821149:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 221.821 and h = 3.00478e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 221.821149:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 147.199 and h = 2.90261e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 147.198629:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 147.199 and h = 2.90261e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 147.198629:
AMICI failed to integrate the forward problem
```

(continues on next page)

(continued from previous page)

```
[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 147.199 and h = 2.90261e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 147.198629:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 198 and h = 2.97875e-05, the error test failed repeatedly or with␣
↪|h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 197.999609:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 197.697 and h = 2.98464e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 197.696730:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 197.697 and h = 2.98464e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 197.696730:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 197.697 and h = 2.98464e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 197.696730:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 66.4603 and h = 6.88533e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 66.460272:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 66.3735 and h = 8.78908e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 66.373478:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 85.8974 and h = 2.05376e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 85.897359:
AMICI failed to integrate the forward problem
```

2.4.5 Visualization

Create waterfall and parameter plot

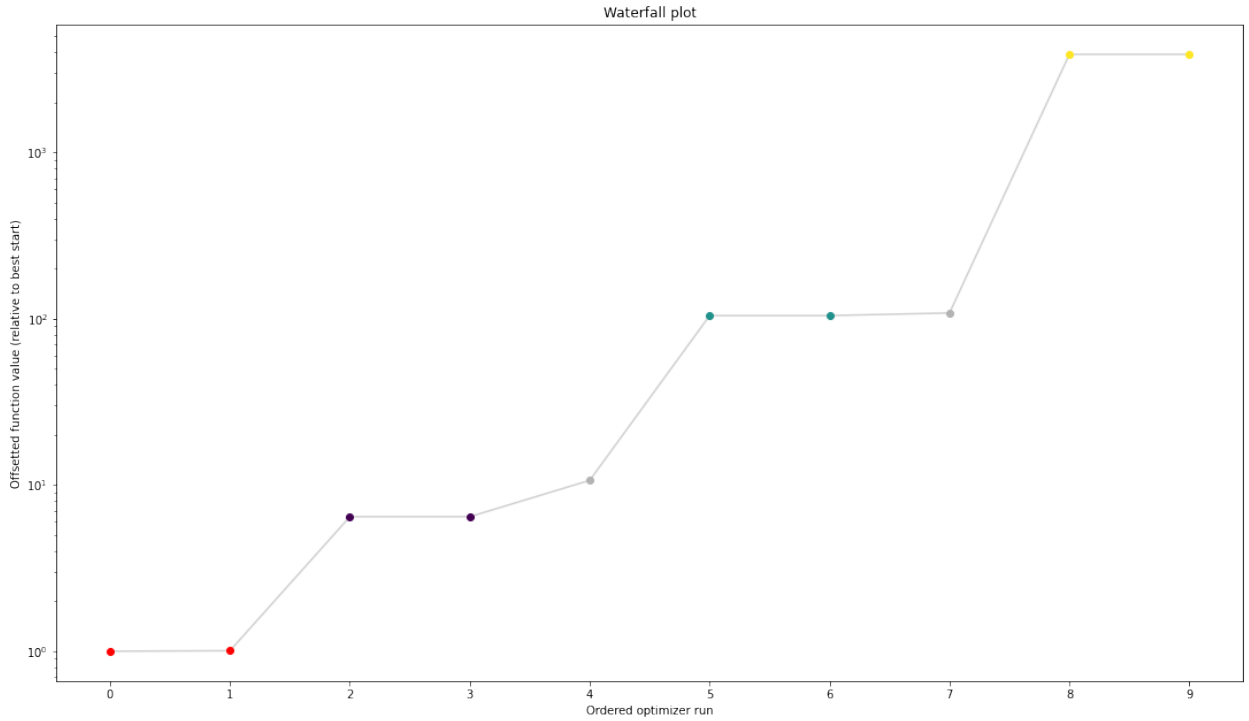
[18]:
```
# waterfall, parameter space,
import pypesto.visualize
```

(continues on next page)

(continued from previous page)

```
pypesto.visualize.waterfall(result)
pypesto.visualize.parameters(result)
```

[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3bc5881c10>
```



2.5 Model import using the Petab format

In this notebook, we illustrate how to use pyPESTO together with PEtab and AMICI. We employ models from the benchmark collection, which we first download:

[1]:
```
import pypesto
import amici
import petab

import os
import numpy as np
```

(continues on next page)

(continued from previous page)

```
import matplotlib.pyplot as plt

%matplotlib inline

!git clone --depth 1 https://github.com/Benchmarking-Initiative/Benchmark-Models-
↪PEtab.git tmp/benchmark-models || (cd tmp/benchmark-models && git pull)

folder_base = "tmp/benchmark-models/Benchmark-Models/"
```

```
fatal: destination path 'tmp/benchmark-models' already exists and is not an empty␣
↪directory.
Already up to date.
```

2.5.1 Import

Manage PEtab model

A PEtab problem comprises all the information on the model, the data and the parameters to perform parameter estimation. We import a model as a `petab.Problem`.

[2]:
```
# a collection of models that can be simulated

#model_name = "Zheng_PNAS2012"
model_name = "Boehm_JProteomeRes2014"
#model_name = "Fujita_SciSignal2010"
#model_name = "Sneyd_PNAS2002"
#model_name = "Borghans_BiophysChem1997"
#model_name = "Elowitz_Nature2000"
#model_name = "Crauste_CellSystems2017"
#model_name = "Lucarelli_CellSystems2018"
#model_name = "Schwen_PONE2014"
#model_name = "Blasi_CellSystems2016"

# the yaml configuration file links to all needed files
yaml_config = os.path.join(folder_base, model_name, model_name + '.yaml')

# create a petab problem
petab_problem = petab.Problem.from_yaml(yaml_config)
```

Import model to AMICI

The model must be imported to pyPESTO and AMICI. Therefore, we create a `pypesto.PetabImporter` from the problem, and create an AMICI model.

[3]:
```
importer = pypesto.PetabImporter(petab_problem)

model = importer.create_model()

# some model properties
print("Model parameters:", list(model.getParameterIds()), '\n')
print("Model const parameters:", list(model.getFixedParameterIds()), '\n')
print("Model outputs:    ", list(model.getObservableIds()), '\n')
print("Model states:     ", list(model.getStateIds()), '\n')
```

```
Model parameters: ['Epo_degradation_BaF3', 'k_exp_hetero', 'k_exp_homo', 'k_imp_hetero
↪', 'k_imp_homo', 'k_phos', 'ratio', 'specC17', 'noiseParameter1_pSTAT5A_rel',
↪'noiseParameter1_pSTAT5B_rel', 'noiseParameter1_rSTAT5A_rel']

Model const parameters: []

Model outputs:    ['pSTAT5A_rel', 'pSTAT5B_rel', 'rSTAT5A_rel']

Model states:     ['STAT5A', 'STAT5B', 'pApB', 'pApA', 'pBpB', 'nucpApA', 'nucpApB',
↪'nucpBpB']
```

Create objective function

To perform parameter estimation, we need to define an objective function, which integrates the model, data, and noise model defined in the PEtab problem.

[4]:
```
import libsbml
converter_config = libsbml.SBMLLocalParameterConverter()\
    .getDefaultProperties()
petab_problem.sbml_document.convert(converter_config)

obj = importer.create_objective()

# for some models, hyperparamters need to be adjusted
#obj.amici_solver.setMaxSteps(10000)
#obj.amici_solver.setRelativeTolerance(1e-7)
#obj.amici_solver.setAbsoluteTolerance(1e-7)
```

We can request variable derivatives via `sensi_orders`, or function values or residuals as specified via `mode`. Passing `return_dict`, we obtain the direct result of the AMICI simulation.

[5]:
```
ret = obj(petab_problem.x_nominal_scaled, mode='mode_fun', sensi_orders=(0,1), return_
↪dict=True)
print(ret)
```

```
{'fval': 138.22199677513575, 'grad': array([ 2.20386015e-02,  5.53227506e-02,  5.
↪78886452e-03,  5.40656415e-03,
       -4.51595809e-05,  7.91163446e-03,  0.00000000e+00,  1.07840959e-02,
        2.40378735e-02,  1.91919657e-02,  0.00000000e+00]), 'hess': array([[ 2.
↪11105595e+03,  5.89390039e-01,  1.07159910e+02,
         2.81393973e+03,  8.94333861e-06, -7.86055092e+02,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00],
       [ 5.89390039e-01,  1.91513744e-03, -1.72774945e-01,
         7.12558479e-01, -3.69774927e-08, -3.20531692e-01,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00],
       [ 1.07159910e+02, -1.72774945e-01,  6.99839693e+01,
         1.61497679e+02,  7.16323554e-06, -8.83572656e+01,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00],
       [ 2.81393973e+03,  7.12558479e-01,  1.61497679e+02,
         3.76058352e+03,  8.40044683e-06, -1.04136909e+03,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00],
```

(continues on next page)

(continued from previous page)

```
        [ 8.94333861e-06, -3.69774927e-08,  7.16323554e-06,
          8.40044683e-06,  2.86438192e-10, -2.24927732e-04,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00],
        [-7.86055092e+02, -3.20531692e-01, -8.83572656e+01,
         -1.04136909e+03, -2.24927732e-04,  9.29902113e+02,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          0.00000000e+00,  0.00000000e+00]]), 'res': array([], dtype=float64), 'sres':␣
↪array([], shape=(0, 11), dtype=float64), 'rdatas': [<amici.numpy.ReturnDataView␣
↪object at 0x7f7802f86610>]}
```

The problem defined in PEtab also defines the fixing of parameters, and parameter bounds. This information is contained in a `pypesto.Problem`.

[6]:
```
problem = importer.create_problem(obj)
```

In particular, the problem accounts for the fixing of parametes.

[7]:
```
print(problem.x_fixed_indices, problem.x_free_indices)
```

```
[6, 10] [0, 1, 2, 3, 4, 5, 7, 8, 9]
```

The problem creates a copy of he objective function that takes into account the fixed parameters. The objective function is able to calculate function values and derivatives. A finite difference check whether the computed gradient is accurate:

[8]:
```
objective = problem.objective
ret = objective(petab_problem.x_nominal_free_scaled, sensi_orders=(0,1))
print(ret)
```

```
(138.22199677513575, array([ 2.20386015e-02,  5.53227506e-02,  5.78886452e-03,  5.
↪40656415e-03,
       -4.51595809e-05,  7.91163446e-03,  1.07840959e-02,  2.40378735e-02,
        1.91919657e-02]))
```

[9]:
```
eps = 1e-4
```

(continues on next page)

(continued from previous page)

```
def fd(x):
    grad = np.zeros_like(x)
    j = 0
    for i, xi in enumerate(x):
        mask = np.zeros_like(x)
        mask[i] += eps
        valinc, _ = objective(x+mask, sensi_orders=(0,1))
        valdec, _ = objective(x-mask, sensi_orders=(0,1))
        grad[j] = (valinc - valdec) / (2*eps)
        j += 1
    return grad

fdval = fd(petab_problem.x_nominal_free_scaled)
print("fd: ", fdval)
print("l2 difference: ", np.linalg.norm(ret[1] - fdval))
```

```
fd:  [0.02493368 0.05309659 0.00530587 0.01291083 0.00587754 0.01473653
 0.01078279 0.02403657 0.01919066]
l2 difference:  0.012310244824532144
```

In short

All of the previous steps can be shortened by directly creating an importer object and then a problem:

[10]:
```
importer = pypesto.PetabImporter.from_yaml(yaml_config)
problem = importer.create_problem()
```

2.5.2 Run optimization

Given the problem, we can perform optimization. We can specify an optimizer to use, and a parallelization engine to speed things up.

[11]:
```
optimizer = pypesto.ScipyOptimizer()

# engine = pypesto.SingleCoreEngine()
engine = pypesto.MultiProcessEngine()

# do the optimization
result = pypesto.minimize(problem=problem, optimizer=optimizer,
                          n_starts=10, engine=engine)
```

```
Engine set up to use up to 4 processes in total. The number was automatically␣
↪determined and might not be appropriate on some systems.
[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 38.1195 and h = 5.55541e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 38.119511:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 88.9211 and h = 2.14177e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 88.921131:
AMICI failed to integrate the forward problem
```

(continues on next page)

(continued from previous page)

```
[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 88.9211 and h = 2.14177e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 88.921131:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 88.9211 and h = 2.14177e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 88.921131:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 145.551 and h = 1.32433e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 145.550813:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 145.551 and h = 1.32433e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 145.550813:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 145.551 and h = 1.32433e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 145.550813:
AMICI failed to integrate the forward problem
```

2.5.3 Visualize

The results are contained in a `pypesto.Result` object. It contains e.g. the optimal function values.

[12]:
```
result.optimize_result.get_for_key('fval')
```

[12]:
```
[138.2219740350346,
 138.22404611978106,
 145.7594099868979,
 147.54397516143254,
 149.58782926326572,
 151.16644923400784,
 154.73312826411254,
 205.61953652493594,
 249.27713115708494,
 249.7459974433355]
```

We can use the standard pyPESTO plotting routines to visualize and analyze the results.

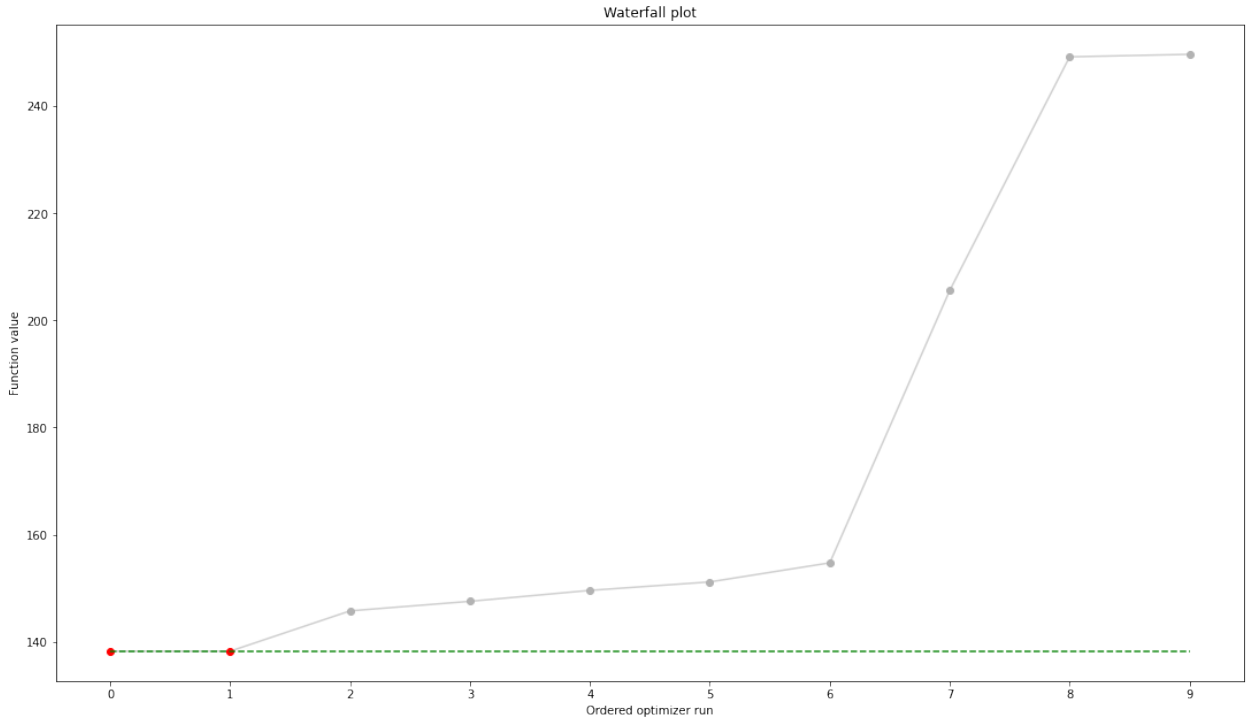
[13]:
```
import pypesto.visualize

ref = pypesto.visualize.create_references(x=petab_problem.x_nominal_scaled,␣
↪fval=obj(petab_problem.x_nominal_scaled))
```

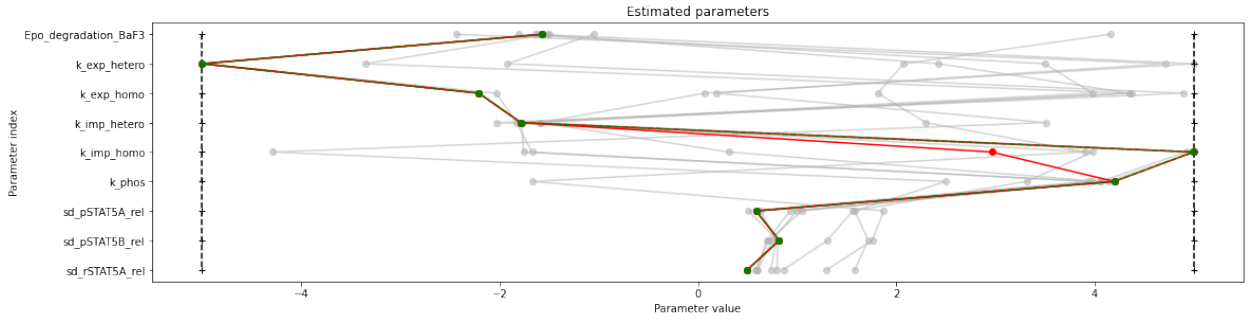
(continues on next page)

(continued from previous page)

```
pypesto.visualize.waterfall(result, reference=ref, scale_y='lin')
pypesto.visualize.parameters(result, reference=ref)
```

[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7802e43510>
```





2.6 Save and load results as HDF5 files

[1]:
```
import pypesto
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
from pypesto.storage import (save_to_hdf5, read_from_hdf5)
import tempfile

%matplotlib inline
```

2.6.1 Define the objective and problem

[2]:
```
objective = pypesto.Objective(fun=sp.optimize.rosen,
                              grad=sp.optimize.rosen_der,
                              hess=sp.optimize.rosen_hess)

dim_full = 10
lb = -5 * np.ones((dim_full, 1))
ub = 5 * np.ones((dim_full, 1))

problem = pypesto.Problem(objective=objective, lb=lb, ub=ub)
```

2.6.2 Run optimization

[3]:
```
# create optimizers
optimizer = pypesto.ScipyOptimizer(method='l-bfgs-b')

# set number of starts
n_starts = 20

# Run optimizaitons
result = pypesto.minimize(
    problem=problem, optimizer=optimizer,
    n_starts=n_starts)
```

[4]:
```
result.optimize_result.list
```

[4]:
```
[{'id': '8',
  'x': array([1.00000001, 0.99999999, 0.99999997, 0.99999998, 0.99999998,
         0.99999999, 0.99999996, 0.99999987, 0.99999971, 0.9999995 ]),
  'fval': 1.4448869867069234e-12,
  'grad': array([ 9.91613312e-06, -2.33793663e-07, -1.84487477e-05, -1.24826804e-06,
         -7.03416051e-06,  1.12040576e-05,  1.88713028e-05, -4.68014961e-07,
         -3.65179645e-05,  1.53152743e-05]),
  'hess': None,
  'res': None,
  'sres': None,
  'n_fval': 73,
  'n_grad': 73,
  'n_hess': 0,
  'n_res': 0,
  'n_sres': 0,
  'x0': array([1.00000001, 0.99999999, 0.99999997, 0.99999998, 0.99999998,
         0.99999999, 0.99999996, 0.99999987, 0.99999971, 0.9999995 ]),
  'fval0': 116315.16334351365,
  'history': <pypesto.objective.history.History at 0x7fefee65a750>,
  'exitflag': 0,
  'time': 0.010613441467285156,
  'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
 {'id': '16',
  'x': array([0.99999994, 0.99999998, 1.        , 1.00000002, 1.00000004,
         1.00000001, 1.        , 1.00000012, 1.0000003 , 1.00000051]),
  'fval': 4.371307796809753e-12,
  'grad': array([-4.26293148e-05,  9.13631144e-06, -1.31339486e-06,  2.51280250e-06,
          2.59501842e-05,  4.21294205e-07, -5.58158396e-05,  7.08567852e-07,
```

(continues on next page)

(continued from previous page)

```
         4.41611237e-05, -1.57413407e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 79,
 'n_grad': 79,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([0.99999994, 0.99999998, 1.        , 1.00000002, 1.00000004,
       1.00000001, 1.        , 1.00000012, 1.0000003 , 1.00000051]),
 'fval0': 127542.57197202934,
 'history': <pypesto.objective.history.History at 0x7fefee65a910>,
 'exitflag': 0,
 'time': 0.011748075485229492,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '4',
 'x': array([1.00000007, 0.99999999, 0.99999998, 1.        , 1.00000008,
       1.00000021, 1.00000015, 1.00000006, 1.00000004, 1.00000013]),
 'fval': 1.7134261938953258e-11,
 'grad': array([ 5.79464879e-05, -3.23661397e-05, -1.13616716e-05, -2.69343079e-05,
       -1.67474293e-06,  1.20454131e-04,  3.83436764e-05, -1.71072644e-05,
       -3.54339727e-05,  1.03840629e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 88,
 'n_grad': 88,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000007, 0.99999999, 0.99999998, 1.        , 1.00000008,
       1.00000021, 1.00000015, 1.00000006, 1.00000004, 1.00000013]),
 'fval0': 169143.33089007522,
 'history': <pypesto.objective.history.History at 0x7fefee9008d0>,
 'exitflag': 0,
 'time': 0.016152620315551758,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '11',
 'x': array([0.99999983, 0.99999997, 0.99999989, 0.9999998 , 1.00000006,
       0.99999986, 0.99999963, 0.99999927, 0.99999861, 0.99999709]),
 'fval': 4.473573948366185e-11,
 'grad': array([-1.25925183e-04,  8.34342658e-05, -1.58946249e-05, -1.85224905e-04,
        2.00742516e-04, -1.80384056e-05, -1.83734314e-05, -2.93938826e-05,
        5.98755497e-05, -2.43744695e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 78,
 'n_grad': 78,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([0.99999983, 0.99999997, 0.99999989, 0.9999998 , 1.00000006,
       0.99999986, 0.99999963, 0.99999927, 0.99999861, 0.99999709]),
 'fval0': 111440.55513257613,
 'history': <pypesto.objective.history.History at 0x7fefee65a7d0>,
```

(continues on next page)

(continued from previous page)

```
 'exitflag': 0,
 'time': 0.009877920150756836,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '0',
 'x': array([1.00000016, 1.00000029, 1.00000013, 1.00000007, 0.99999984,
       0.99999968, 0.99999955, 0.99999932, 0.99999871, 0.99999745]),
 'fval': 4.512690733773355e-11,
 'grad': array([ 6.88017381e-06,  1.82437618e-04, -1.71219792e-05,  7.83029016e-05,
       -5.64629619e-05, -7.75613657e-05, -5.33124129e-05,  2.00358870e-05,
       -3.38990540e-06,  6.36234430e-06]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 80,
 'n_grad': 80,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000016, 1.00000029, 1.00000013, 1.00000007, 0.99999984,
       0.99999968, 0.99999955, 0.99999932, 0.99999871, 0.99999745]),
 'fval0': 179787.03971937217,
 'history': <pypesto.objective.history.History at 0x7fefee6d1e50>,
 'exitflag': 0,
 'time': 0.05712604522705078,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '18',
 'x': array([0.9999999 , 0.99999988, 1.00000002, 1.00000017, 1.00000034,
       1.00000045, 1.00000052, 1.00000076, 1.00000146, 1.00000315]),
 'fval': 4.530338040953872e-11,
 'grad': array([-3.46616032e-05, -8.55052094e-05,  4.32179353e-06,  2.27795791e-05,
        8.77325561e-05,  1.11150847e-04,  4.14626291e-05, -3.72317820e-05,
       -9.65698380e-05,  4.47893709e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 85,
 'n_grad': 85,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([0.9999999 , 0.99999988, 1.00000002, 1.00000017, 1.00000034,
       1.00000045, 1.00000052, 1.00000076, 1.00000146, 1.00000315]),
 'fval0': 84246.35907849146,
 'history': <pypesto.objective.history.History at 0x7fefee65a990>,
 'exitflag': 0,
 'time': 0.01163339614868164,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '17',
 'x': array([1.00000014, 1.00000006, 1.00000012, 1.00000002, 0.99999972,
       0.99999976, 0.99999924, 0.9999986 , 0.99999728, 0.99999457]),
 'fval': 5.187501773111393e-11,
 'grad': array([ 8.98076519e-05, -4.45109249e-05,  8.61160519e-05,  8.83761172e-05,
       -1.98032428e-04,  1.80982671e-04, -1.05227326e-04, -1.64856814e-05,
        1.06897803e-05,  1.09849767e-06]),
 'hess': None,
 'res': None,
 'sres': None,
```

(continues on next page)

(continued from previous page)

```
 'n_fval': 99,
 'n_grad': 99,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000014, 1.00000006, 1.00000012, 1.00000002, 0.99999972,
        0.99999976, 0.99999924, 0.9999986 , 0.99999728, 0.99999457]),
 'fval0': 164257.74387447865,
 'history': <pypesto.objective.history.History at 0x7fefee65a950>,
 'exitflag': 0,
 'time': 0.013841629028320312,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '13',
 'x': array([1.00000009, 1.00000004, 1.00000001, 1.0000001 , 1.00000019,
        1.00000029, 1.00000006, 1.00000059, 1.00000094, 1.00000194]),
 'fval': 6.113882079371062e-11,
 'grad': array([ 5.38201966e-05,  1.12643005e-06, -4.79180197e-05,  1.70876435e-05,
         3.76896795e-05,  1.90462489e-04, -2.94335966e-04,  1.92128181e-04,
        -7.02675278e-05,  1.21112443e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 84,
 'n_grad': 84,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000009, 1.00000004, 1.00000001, 1.0000001 , 1.00000019,
        1.00000029, 1.00000006, 1.00000059, 1.00000094, 1.00000194]),
 'fval0': 128057.90608516608,
 'history': <pypesto.objective.history.History at 0x7fefee65a850>,
 'exitflag': 0,
 'time': 0.011581659317016602,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '9',
 'x': array([1.0000003 , 1.00000024, 0.99999999, 0.99999981, 0.99999975,
        0.99999984, 1.00000003, 1.00000007, 1.00000014, 1.00000029]),
 'fval': 6.603657918190325e-11,
 'grad': array([ 1.45585786e-04,  1.26131400e-04, -3.15965052e-05, -8.20696700e-05,
        -1.17487544e-04, -6.42577094e-05,  5.98749705e-05,  4.78675947e-06,
        -7.72296480e-06,  3.75853482e-06]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 83,
 'n_grad': 83,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.0000003 , 1.00000024, 0.99999999, 0.99999981, 0.99999975,
        0.99999984, 1.00000003, 1.00000007, 1.00000014, 1.00000029]),
 'fval0': 66001.21516931924,
 'history': <pypesto.objective.history.History at 0x7ff00035a3d0>,
 'exitflag': 0,
 'time': 0.012688159942626953,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '19',
```

(continues on next page)

(continued from previous page)

```
 'x': array([1.00000001, 1.        , 0.99999996, 0.99999988, 1.00000026,
       1.00000032, 1.00000033, 1.00000085, 1.00000187, 1.00000333]),
 'fval': 6.709274183733498e-11,
 'grad': array([ 8.70631729e-06,  9.70587008e-06,  1.05693559e-05, -2.12546166e-04,
        1.84978271e-04,  8.28659608e-05, -1.39703429e-04, -2.30820323e-05,
        1.96299302e-04, -8.02670983e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 67,
 'n_grad': 67,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000001, 1.        , 0.99999996, 0.99999988, 1.00000026,
       1.00000032, 1.00000033, 1.00000085, 1.00000187, 1.00000333]),
 'fval0': 80442.06208067665,
 'history': <pypesto.objective.history.History at 0x7fefee65a9d0>,
 'exitflag': 0,
 'time': 0.008251428604125977,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '10',
 'x': array([0.9999998 , 1.00000012, 0.99999997, 1.00000017, 1.00000028,
       1.00000037, 1.0000004 , 1.0000005 , 1.00000069, 1.00000195]),
 'fval': 1.065068443513822e-10,
 'grad': array([-2.07797214e-04,  2.08384202e-04, -1.40258436e-04,  6.72135025e-05,
        6.29120089e-05,  9.92424271e-05,  5.28220148e-05,  6.30692991e-05,
       -2.90590939e-04,  1.15498187e-04]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 135,
 'n_grad': 135,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([0.9999998 , 1.00000012, 0.99999997, 1.00000017, 1.00000028,
       1.00000037, 1.0000004 , 1.0000005 , 1.00000069, 1.00000195]),
 'fval0': 218642.53588542074,
 'history': <pypesto.objective.history.History at 0x7fefee65a510>,
 'exitflag': 0,
 'time': 0.017060041427612305,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '6',
 'x': array([1.00000009, 0.9999998 , 1.00000026, 1.00000009, 1.00000031,
       1.00000016, 1.00000029, 1.00000052, 1.00000131, 1.00000213]),
 'fval': 1.3507352118480165e-10,
 'grad': array([ 1.50715573e-04, -3.42587714e-04,  3.08461493e-04, -1.41826830e-04,
        2.10332130e-04, -7.64162802e-05,  1.68271418e-05, -1.20534088e-04,
        2.55005501e-04, -9.87312659e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 86,
 'n_grad': 86,
 'n_hess': 0,
 'n_res': 0,
```

(continues on next page)

(continued from previous page)

```
 'n_sres': 0,
 'x0': array([1.00000009, 0.9999998 , 1.00000026, 1.00000009, 1.00000031,
        1.00000016, 1.00000029, 1.00000052, 1.00000131, 1.00000213]),
 'fval0': 49550.65276671963,
 'history': <pypesto.objective.history.History at 0x7fefef785510>,
 'exitflag': 0,
 'time': 0.012197017669677734,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '14',
 'x': array([1.00000021, 0.99999999, 0.99999968, 0.99999939, 0.99999938,
        0.99999947, 0.9999996 , 0.99999955, 0.99999928, 0.99999839]),
 'fval': 1.76508815611245e-10,
 'grad': array([ 1.69974542e-04,  3.68276755e-05, -7.46786006e-05, -2.36880611e-04,
        -1.60596472e-04, -1.26090555e-04, -5.76633224e-06, -1.01269226e-06,
         9.59426070e-05, -3.14643190e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 94,
 'n_grad': 94,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000021, 0.99999999, 0.99999968, 0.99999939, 0.99999938,
        0.99999947, 0.9999996 , 0.99999955, 0.99999928, 0.99999839]),
 'fval0': 113189.5063880412,
 'history': <pypesto.objective.history.History at 0x7fefee65a890>,
 'exitflag': 0,
 'time': 0.010685205459594727,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '7',
 'x': array([1.00000001, 1.00000008, 1.00000032, 0.9999997 , 0.99999988,
        0.9999994 , 0.99999945, 0.99999874, 0.9999976 , 0.99999541]),
 'fval': 1.8629501787028135e-10,
 'grad': array([-1.89690853e-05, -5.75176850e-05,  4.08684636e-04, -3.79119426e-04,
         2.41840701e-04, -3.35502015e-04,  1.94400780e-04, -8.41625352e-05,
        -6.36788959e-05,  4.17276302e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 86,
 'n_grad': 86,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000001, 1.00000008, 1.00000032, 0.9999997 , 0.99999988,
        0.9999994 , 0.99999945, 0.99999874, 0.9999976 , 0.99999541]),
 'fval0': 275340.0482345366,
 'history': <pypesto.objective.history.History at 0x7fefee65a550>,
 'exitflag': 0,
 'time': 0.011858463287353516,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '5',
 'x': array([1.00000063, 1.00000079, 1.00000065, 1.00000046, 1.00000018,
        0.99999966, 0.99999884, 0.99999747, 0.99999501, 0.99999004]),
 'fval': 3.4686626972583124e-10,
 'grad': array([ 1.88621968e-04,  2.82853573e-04,  1.47613974e-04,  1.32427091e-04,
```

(continues on next page)

(continued from previous page)

```
         1.34368505e-04,  4.58042858e-05, -1.05802446e-05, -7.23486331e-05,
        -8.94165854e-06,  5.57700990e-06]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 98,
 'n_grad': 98,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000063, 1.00000079, 1.00000065, 1.00000046, 1.00000018,
       0.99999966, 0.99999884, 0.99999747, 0.99999501, 0.99999004]),
 'fval0': 95086.57486034792,
 'history': <pypesto.objective.history.History at 0x7fefee65a650>,
 'exitflag': 0,
 'time': 0.013252247573852539,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '12',
 'x': array([0.99999993, 1.00000061, 1.00000043, 0.9999998 , 1.00000026,
       1.00000048, 1.00000068, 1.00000084, 1.00000215, 1.00000338]),
 'fval': 4.2760857236608074e-10,
 'grad': array([-0.0003034 ,  0.00046938,  0.00026867, -0.00047909,  0.00014639,
        0.00010591,  0.00015162, -0.00028974,  0.00047086, -0.00018617]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 77,
 'n_grad': 77,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([0.99999993, 1.00000061, 1.00000043, 0.9999998 , 1.00000026,
       1.00000048, 1.00000068, 1.00000084, 1.00000215, 1.00000338]),
 'fval0': 278737.0766282746,
 'history': <pypesto.objective.history.History at 0x7fefee65a710>,
 'exitflag': 0,
 'time': 0.010965824127197266,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '1',
 'x': array([0.99999964, 1.00000006, 0.99999975, 1.00000012, 0.99999977,
       0.99999967, 0.99999847, 0.99999623, 0.9999936 , 0.99998718]),
 'fval': 4.4824491354324123e-10,
 'grad': array([-3.15440907e-04,  3.04897788e-04, -3.24323195e-04,  3.10014361e-04,
       -1.40439929e-04,  3.69237832e-04,  1.05035392e-04, -6.01846648e-04,
        2.19479487e-04, -2.71116767e-06]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 81,
 'n_grad': 81,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([0.99999964, 1.00000006, 0.99999975, 1.00000012, 0.99999977,
       0.99999967, 0.99999847, 0.99999623, 0.9999936 , 0.99998718]),
 'fval0': 231983.4016462493,
 'history': <pypesto.objective.history.History at 0x7ff030770110>,
```

(continues on next page)

(continued from previous page)

```
 'exitflag': 0,
 'time': 0.022760868072509766,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '15',
 'x': array([-0.99326331,  0.99660594,  0.99824067,  0.9989884 ,  0.99922624,
        0.99907383,  0.99845405,  0.99705586,  0.9941786 ,  0.98839115]),
 'fval': 3.9865791123861647,
 'grad': array([ 1.37836191e-05, -9.56428278e-05,  1.15714471e-04, -9.46304780e-05,
        2.76772792e-05,  1.99653191e-04, -4.63338544e-05, -2.78221136e-05,
       -2.26556385e-05,  1.07817815e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 72,
 'n_grad': 72,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([-0.99326331,  0.99660594,  0.99824067,  0.9989884 ,  0.99922624,
        0.99907383,  0.99845405,  0.99705586,  0.9941786 ,  0.98839115]),
 'fval0': 117406.38350731946,
 'history': <pypesto.objective.history.History at 0x7fefee65a8d0>,
 'exitflag': 0,
 'time': 0.01103520393371582,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '2',
 'x': array([-0.99326315,  0.99660609,  0.99824021,  0.99898815,  0.99922624,
        0.99907353,  0.99845395,  0.99705588,  0.99417909,  0.98839179]),
 'fval': 3.986579112503477,
 'grad': array([ 1.91428162e-04,  2.94649756e-04, -3.04516493e-04, -1.62074006e-04,
        2.52224941e-04, -6.28043726e-05, -3.29243223e-05, -1.67555012e-04,
        1.96809844e-04, -5.29321104e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 78,
 'n_grad': 78,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([-0.99326315,  0.99660609,  0.99824021,  0.99898815,  0.99922624,
        0.99907353,  0.99845395,  0.99705588,  0.99417909,  0.98839179]),
 'fval0': 90060.0282425554,
 'history': <pypesto.objective.history.History at 0x7fefee6d19d0>,
 'exitflag': 0,
 'time': 0.016544103622436523,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '3',
 'x': array([-0.99326274,  0.99660589,  0.99824031,  0.99898835,  0.99922654,
        0.99907324,  0.99845446,  0.99705616,  0.99417916,  0.98839222]),
 'fval': 3.9865791128374686,
 'grad': array([ 4.43724847e-04,  2.30363468e-04, -2.12480723e-04, -1.17788491e-04,
        5.86174712e-04, -6.78812087e-04,  4.85626746e-04, -1.17917644e-04,
       -1.48260191e-05,  3.53218137e-06]),
 'hess': None,
 'res': None,
 'sres': None,
```

(continues on next page)

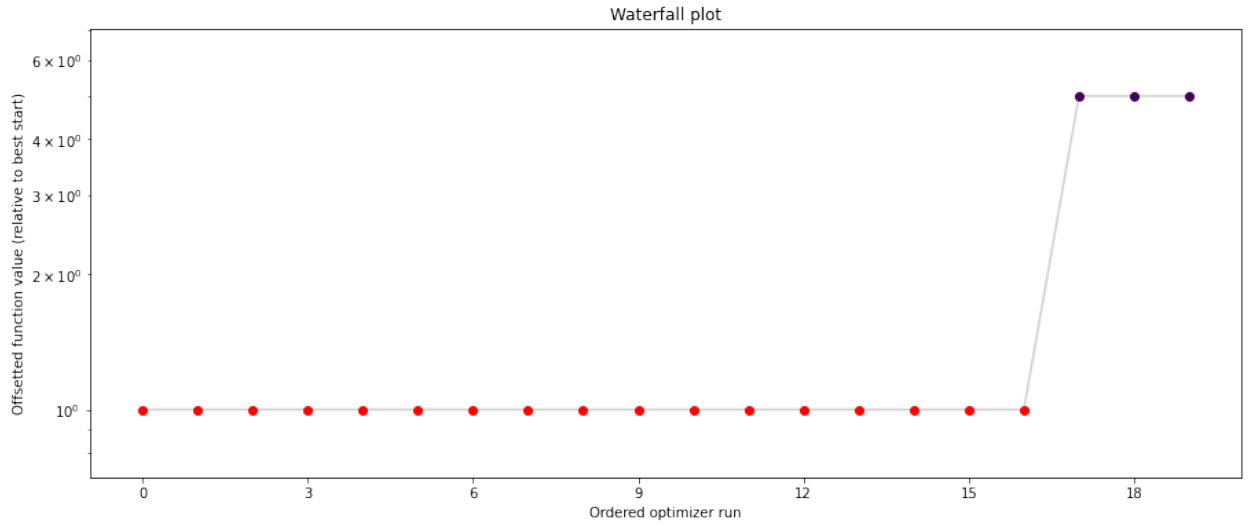
(continued from previous page)

```
  'n_fval': 68,
  'n_grad': 68,
  'n_hess': 0,
  'n_res': 0,
  'n_sres': 0,
  'x0': array([-0.99326274,  0.99660589,  0.99824031,  0.99898835,  0.99922654,
          0.99907324,  0.99845446,  0.99705616,  0.99417916,  0.98839222]),
  'fval0': 58537.15752301021,
  'history': <pypesto.objective.history.History at 0x7ff0003b3c90>,
  'exitflag': 0,
  'time': 0.007930994033813477,
  'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'}]
```

2.6.3 Plot results

[5]:
```
import pypesto.visualize

# plot waterfalls
pypesto.visualize.waterfall(result, size=(15,6))
```

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fefef14a310>
```



2.6.4 Save optimization result as HDF5 file

[6]:
```
fn = tempfile.mktemp(".hdf5")

# Write result
hdf5_writer = save_to_hdf5.OptimizationResultHDF5Writer(fn)
hdf5_writer.write(result)

# Write problem
hdf5_writer = save_to_hdf5.ProblemHDF5Writer(fn)
hdf5_writer.write(problem)
```

[7]:
```
# Read result and problem
hdf5_reader = read_from_hdf5.OptimizationResultHDF5Reader(fn)
result = hdf5_reader.read()
```

[8]:
```
result.optimize_result.list
```

[8]:
```
[{'id': '8',
  'x': array([1.00000001, 0.99999999, 0.99999997, 0.99999998, 0.99999998,
         0.99999999, 0.99999996, 0.99999987, 0.99999971, 0.9999995 ]),
  'fval': 1.4448869867069234e-12,
  'grad': array([ 9.91613312e-06, -2.33793663e-07, -1.84487477e-05, -1.24826804e-06,
         -7.03416051e-06,  1.12040576e-05,  1.88713028e-05, -4.68014961e-07,
         -3.65179645e-05,  1.53152743e-05]),
  'hess': None,
  'res': None,
  'sres': None,
  'n_fval': 73,
  'n_grad': 73,
  'n_hess': 0,
  'n_res': 0,
  'n_sres': 0,
  'x0': array([1.00000001, 0.99999999, 0.99999997, 0.99999998, 0.99999998,
         0.99999999, 0.99999996, 0.99999987, 0.99999971, 0.9999995 ]),
  'fval0': 116315.16334351365,
  'history': None,
  'exitflag': 0,
  'time': 0.010613441467285156,
  'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
 {'id': '16',
  'x': array([0.99999994, 0.99999998, 1.        , 1.00000002, 1.00000004,
         1.00000001, 1.        , 1.00000012, 1.0000003 , 1.00000051]),
  'fval': 4.371307796809753e-12,
  'grad': array([-4.26293148e-05,  9.13631144e-06, -1.31339486e-06,  2.51280250e-06,
          2.59501842e-05,  4.21294205e-07, -5.58158396e-05,  7.08567852e-07,
          4.41611237e-05, -1.57413407e-05]),
  'hess': None,
  'res': None,
  'sres': None,
  'n_fval': 79,
  'n_grad': 79,
  'n_hess': 0,
  'n_res': 0,
  'n_sres': 0,
  'x0': array([0.99999994, 0.99999998, 1.        , 1.00000002, 1.00000004,
         1.00000001, 1.        , 1.00000012, 1.0000003 , 1.00000051]),
  'fval0': 127542.57197202934,
  'history': None,
  'exitflag': 0,
  'time': 0.011748075485229492,
  'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
 {'id': '4',
  'x': array([1.00000007, 0.99999999, 0.99999998, 1.        , 1.00000008,
         1.00000021, 1.00000015, 1.00000006, 1.00000004, 1.00000013]),
  'fval': 1.7134261938953258e-11,
  'grad': array([ 5.79464879e-05, -3.23661397e-05, -1.13616716e-05, -2.69343079e-05,
         -1.67474293e-06,  1.20454131e-04,  3.83436764e-05, -1.71072644e-05,
         -3.54339727e-05,  1.03840629e-05]),
```

(continues on next page)

(continued from previous page)

```
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 88,
 'n_grad': 88,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000007, 0.99999999, 0.99999998, 1.        , 1.00000008,
        1.00000021, 1.00000015, 1.00000006, 1.00000004, 1.00000013]),
 'fval0': 169143.33089007522,
 'history': None,
 'exitflag': 0,
 'time': 0.016152620315551758,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '11',
 'x': array([0.99999983, 0.99999997, 0.99999989, 0.9999998 , 1.00000006,
        0.99999986, 0.99999963, 0.99999927, 0.99999861, 0.99999709]),
 'fval': 4.473573948366185e-11,
 'grad': array([-1.25925183e-04,  8.34342658e-05, -1.58946249e-05, -1.85224905e-04,
         2.00742516e-04, -1.80384056e-05, -1.83734314e-05, -2.93938826e-05,
         5.98755497e-05, -2.43744695e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 78,
 'n_grad': 78,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([0.99999983, 0.99999997, 0.99999989, 0.9999998 , 1.00000006,
        0.99999986, 0.99999963, 0.99999927, 0.99999861, 0.99999709]),
 'fval0': 111440.55513257613,
 'history': None,
 'exitflag': 0,
 'time': 0.009877920150756836,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '0',
 'x': array([1.00000016, 1.00000029, 1.00000013, 1.00000007, 0.99999984,
        0.99999968, 0.99999955, 0.99999932, 0.99999871, 0.99999745]),
 'fval': 4.512690733773355e-11,
 'grad': array([ 6.88017381e-06,  1.82437618e-04, -1.71219792e-05,  7.83029016e-05,
        -5.64629619e-05, -7.75613657e-05, -5.33124129e-05,  2.00358870e-05,
        -3.38990540e-06,  6.36234430e-06]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 80,
 'n_grad': 80,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000016, 1.00000029, 1.00000013, 1.00000007, 0.99999984,
        0.99999968, 0.99999955, 0.99999932, 0.99999871, 0.99999745]),
 'fval0': 179787.03971937217,
 'history': None,
 'exitflag': 0,
```

(continues on next page)

(continued from previous page)

```
 'time': 0.05712604522705078,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '18',
 'x': array([0.9999999 , 0.99999988, 1.00000002, 1.00000017, 1.00000034,
       1.00000045, 1.00000052, 1.00000076, 1.00000146, 1.00000315]),
 'fval': 4.530338040953872e-11,
 'grad': array([-3.46616032e-05, -8.55052094e-05,  4.32179353e-06,  2.27795791e-05,
        8.77325561e-05,  1.11150847e-04,  4.14626291e-05, -3.72317820e-05,
       -9.65698380e-05,  4.47893709e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 85,
 'n_grad': 85,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([0.9999999 , 0.99999988, 1.00000002, 1.00000017, 1.00000034,
       1.00000045, 1.00000052, 1.00000076, 1.00000146, 1.00000315]),
 'fval0': 84246.35907849146,
 'history': None,
 'exitflag': 0,
 'time': 0.01163339614868164,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '17',
 'x': array([1.00000014, 1.00000006, 1.00000012, 1.00000002, 0.99999972,
       0.99999976, 0.99999924, 0.9999986 , 0.99999728, 0.99999457]),
 'fval': 5.187501773111393e-11,
 'grad': array([ 8.98076519e-05, -4.45109249e-05,  8.61160519e-05,  8.83761172e-05,
       -1.98032428e-04,  1.80982671e-04, -1.05227326e-04, -1.64856814e-05,
        1.06897803e-05,  1.09849767e-06]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 99,
 'n_grad': 99,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000014, 1.00000006, 1.00000012, 1.00000002, 0.99999972,
       0.99999976, 0.99999924, 0.9999986 , 0.99999728, 0.99999457]),
 'fval0': 164257.74387447865,
 'history': None,
 'exitflag': 0,
 'time': 0.013841629028320312,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '13',
 'x': array([1.00000009, 1.00000004, 1.00000001, 1.0000001 , 1.00000019,
       1.00000029, 1.00000006, 1.00000059, 1.00000094, 1.00000194]),
 'fval': 6.113882079371062e-11,
 'grad': array([ 5.38201966e-05,  1.12643005e-06, -4.79180197e-05,  1.70876435e-05,
        3.76896795e-05,  1.90462489e-04, -2.94335966e-04,  1.92128181e-04,
       -7.02675278e-05,  1.21112443e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 84,
```

(continues on next page)

(continued from previous page)

```
 'n_grad': 84,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000009, 1.00000004, 1.00000001, 1.0000001 , 1.00000019,
        1.00000029, 1.00000006, 1.00000059, 1.00000094, 1.00000194]),
 'fval0': 128057.90608516608,
 'history': None,
 'exitflag': 0,
 'time': 0.011581659317016602,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '9',
 'x': array([1.0000003 , 1.00000024, 0.99999999, 0.99999981, 0.99999975,
        0.99999984, 1.00000003, 1.00000007, 1.00000014, 1.00000029]),
 'fval': 6.603657918190325e-11,
 'grad': array([ 1.45585786e-04,  1.26131400e-04, -3.15965052e-05, -8.20696700e-05,
        -1.17487544e-04, -6.42577094e-05,  5.98749705e-05,  4.78675947e-06,
        -7.72296480e-06,  3.75853482e-06]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 83,
 'n_grad': 83,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.0000003 , 1.00000024, 0.99999999, 0.99999981, 0.99999975,
        0.99999984, 1.00000003, 1.00000007, 1.00000014, 1.00000029]),
 'fval0': 66001.21516931924,
 'history': None,
 'exitflag': 0,
 'time': 0.012688159942626953,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '19',
 'x': array([1.00000001, 1.        , 0.99999996, 0.99999988, 1.00000026,
        1.00000032, 1.00000033, 1.00000085, 1.00000187, 1.00000333]),
 'fval': 6.709274183733498e-11,
 'grad': array([ 8.70631729e-06,  9.70587008e-06,  1.05693559e-05, -2.12546166e-04,
         1.84978271e-04,  8.28659608e-05, -1.39703429e-04, -2.30820323e-05,
         1.96299302e-04, -8.02670983e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 67,
 'n_grad': 67,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000001, 1.        , 0.99999996, 0.99999988, 1.00000026,
        1.00000032, 1.00000033, 1.00000085, 1.00000187, 1.00000333]),
 'fval0': 80442.06208067665,
 'history': None,
 'exitflag': 0,
 'time': 0.008251428604125977,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '10',
 'x': array([0.9999998 , 1.00000012, 0.99999997, 1.00000017, 1.00000028,
```

(continues on next page)

(continued from previous page)

```
        1.00000037, 1.0000004 , 1.0000005 , 1.00000069, 1.00000195]),
 'fval': 1.065068443513822e-10,
 'grad': array([-2.07797214e-04,  2.08384202e-04, -1.40258436e-04,  6.72135025e-05,
         6.29120089e-05,  9.92424271e-05,  5.28220148e-05,  6.30692991e-05,
        -2.90590939e-04,  1.15498187e-04]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 135,
 'n_grad': 135,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([0.9999998 , 1.00000012, 0.99999997, 1.00000017, 1.00000028,
        1.00000037, 1.0000004 , 1.0000005 , 1.00000069, 1.00000195]),
 'fval0': 218642.53588542074,
 'history': None,
 'exitflag': 0,
 'time': 0.017060041427612305,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '6',
 'x': array([1.00000009, 0.9999998 , 1.00000026, 1.00000009, 1.00000031,
        1.00000016, 1.00000029, 1.00000052, 1.00000131, 1.00000213]),
 'fval': 1.3507352118480165e-10,
 'grad': array([ 1.50715573e-04, -3.42587714e-04,  3.08461493e-04, -1.41826830e-04,
         2.10332130e-04, -7.64162802e-05,  1.68271418e-05, -1.20534088e-04,
         2.55005501e-04, -9.87312659e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 86,
 'n_grad': 86,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000009, 0.9999998 , 1.00000026, 1.00000009, 1.00000031,
        1.00000016, 1.00000029, 1.00000052, 1.00000131, 1.00000213]),
 'fval0': 49550.65276671963,
 'history': None,
 'exitflag': 0,
 'time': 0.012197017669677734,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '14',
 'x': array([1.00000021, 0.99999999, 0.99999968, 0.99999939, 0.99999938,
        0.99999947, 0.9999996 , 0.99999955, 0.99999928, 0.99999839]),
 'fval': 1.76508815611245e-10,
 'grad': array([ 1.69974542e-04,  3.68276755e-05, -7.46786006e-05, -2.36880611e-04,
        -1.60596472e-04, -1.26090555e-04, -5.76633224e-06, -1.01269226e-06,
         9.59426070e-05, -3.14643190e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 94,
 'n_grad': 94,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
```

(continues on next page)

(continued from previous page)

```
 'x0': array([1.00000021, 0.99999999, 0.99999968, 0.99999939, 0.99999938,
        0.99999947, 0.9999996 , 0.99999955, 0.99999928, 0.99999839]),
 'fval0': 113189.5063880412,
 'history': None,
 'exitflag': 0,
 'time': 0.010685205459594727,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '7',
 'x': array([1.00000001, 1.00000008, 1.00000032, 0.9999997 , 0.99999988,
        0.9999994 , 0.99999945, 0.99999874, 0.9999976 , 0.99999541]),
 'fval': 1.8629501787028135e-10,
 'grad': array([-1.89690853e-05, -5.75176850e-05,  4.08684636e-04, -3.79119426e-04,
         2.41840701e-04, -3.35502015e-04,  1.94400780e-04, -8.41625352e-05,
        -6.36788959e-05,  4.17276302e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 86,
 'n_grad': 86,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000001, 1.00000008, 1.00000032, 0.9999997 , 0.99999988,
        0.9999994 , 0.99999945, 0.99999874, 0.9999976 , 0.99999541]),
 'fval0': 275340.0482345366,
 'history': None,
 'exitflag': 0,
 'time': 0.011858463287353516,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '5',
 'x': array([1.00000063, 1.00000079, 1.00000065, 1.00000046, 1.00000018,
        0.99999966, 0.99999884, 0.99999747, 0.99999501, 0.99999004]),
 'fval': 3.4686626972583124e-10,
 'grad': array([ 1.88621968e-04,  2.82853573e-04,  1.47613974e-04,  1.32427091e-04,
         1.34368505e-04,  4.58042858e-05, -1.05802446e-05, -7.23486331e-05,
        -8.94165854e-06,  5.57700990e-06]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 98,
 'n_grad': 98,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([1.00000063, 1.00000079, 1.00000065, 1.00000046, 1.00000018,
        0.99999966, 0.99999884, 0.99999747, 0.99999501, 0.99999004]),
 'fval0': 95086.57486034792,
 'history': None,
 'exitflag': 0,
 'time': 0.013525247573852539,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '12',
 'x': array([0.99999993, 1.00000061, 1.00000043, 0.9999998 , 1.00000026,
        1.00000048, 1.00000068, 1.00000084, 1.00000215, 1.00000338]),
 'fval': 4.2760857236608074e-10,
 'grad': array([-0.0003034 ,  0.00046938,  0.00026867, -0.00047909,  0.00014639,
         0.00010591,  0.00015162, -0.00028974,  0.00047086, -0.00018617]),
```

(continues on next page)

(continued from previous page)

```
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 77,
 'n_grad': 77,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([0.99999993, 1.00000061, 1.00000043, 0.9999998 , 1.00000026,
        1.00000048, 1.00000068, 1.00000084, 1.00000215, 1.00000338]),
 'fval0': 278737.0766282746,
 'history': None,
 'exitflag': 0,
 'time': 0.010965824127197266,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '1',
 'x': array([0.99999964, 1.00000006, 0.99999975, 1.00000012, 0.99999977,
        0.99999967, 0.99999847, 0.99999623, 0.9999936 , 0.99998718]),
 'fval': 4.4824491354324123e-10,
 'grad': array([-3.15440907e-04,  3.04897788e-04, -3.24323195e-04,  3.10014361e-04,
        -1.40439929e-04,  3.69237832e-04,  1.05035392e-04, -6.01846648e-04,
         2.19479487e-04, -2.71116767e-06]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 81,
 'n_grad': 81,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([0.99999964, 1.00000006, 0.99999975, 1.00000012, 0.99999977,
        0.99999967, 0.99999847, 0.99999623, 0.9999936 , 0.99998718]),
 'fval0': 231983.4016462493,
 'history': None,
 'exitflag': 0,
 'time': 0.022760868072509766,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
{'id': '15',
 'x': array([-0.99326331,  0.99660594,  0.99824067,  0.9989884 ,  0.99922624,
         0.99907383,  0.99845405,  0.99705586,  0.9941786 ,  0.98839115]),
 'fval': 3.9865791123861647,
 'grad': array([ 1.37836191e-05, -9.56428278e-05,  1.15714471e-04, -9.46304780e-05,
         2.76772792e-05,  1.99653191e-04, -4.63338544e-05, -2.78221136e-05,
        -2.26556385e-05,  1.07817815e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 72,
 'n_grad': 72,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([-0.99326331,  0.99660594,  0.99824067,  0.9989884 ,  0.99922624,
         0.99907383,  0.99845405,  0.99705586,  0.9941786 ,  0.98839115]),
 'fval0': 117406.38350731946,
 'history': None,
 'exitflag': 0,
```

(continues on next page)

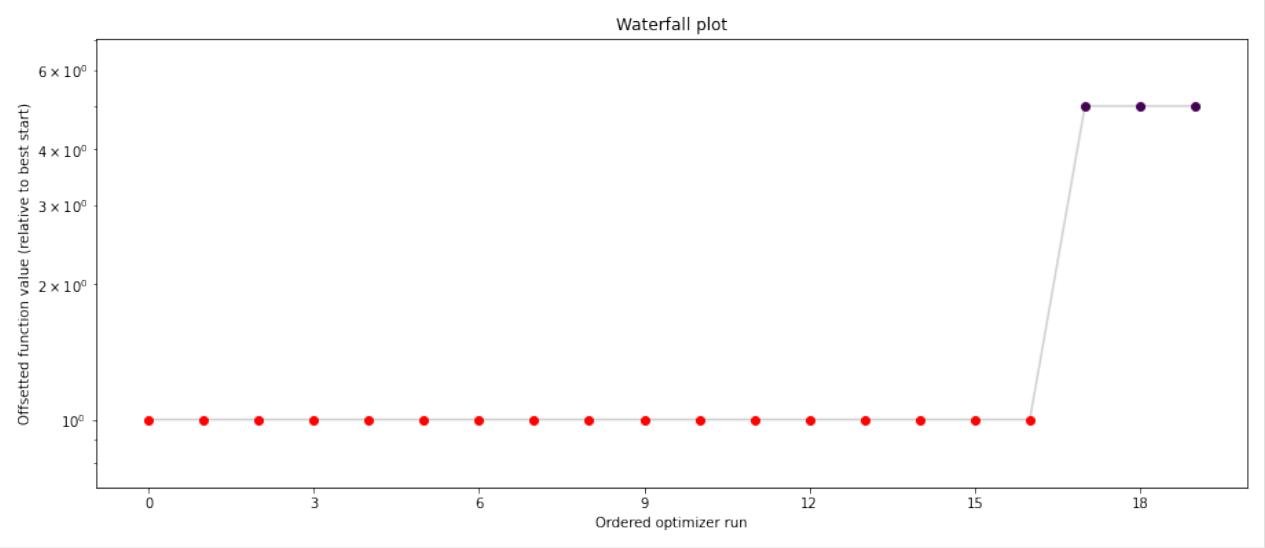
(continued from previous page)

```
  'time': 0.01103520393371582,
  'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
 {'id': '2',
  'x': array([-0.99326315,  0.99660609,  0.99824021,  0.99898815,  0.99922624,
         0.99907353,  0.99845395,  0.99705588,  0.99417909,  0.98839179]),
  'fval': 3.986579112503477,
  'grad': array([ 1.91428162e-04,  2.94649756e-04, -3.04516493e-04, -1.62074006e-04,
         2.52224941e-04, -6.28043726e-05, -3.29243223e-05, -1.67555012e-04,
         1.96809844e-04, -5.29321104e-05]),
  'hess': None,
  'res': None,
  'sres': None,
  'n_fval': 78,
  'n_grad': 78,
  'n_hess': 0,
  'n_res': 0,
  'n_sres': 0,
  'x0': array([-0.99326315,  0.99660609,  0.99824021,  0.99898815,  0.99922624,
         0.99907353,  0.99845395,  0.99705588,  0.99417909,  0.98839179]),
  'fval0': 90060.0282425554,
  'history': None,
  'exitflag': 0,
  'time': 0.016544103622436523,
  'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
 {'id': '3',
  'x': array([-0.99326274,  0.99660589,  0.99824031,  0.99898835,  0.99922654,
         0.99907324,  0.99845446,  0.99705616,  0.99417916,  0.98839222]),
  'fval': 3.9865791128374686,
  'grad': array([ 4.43724847e-04,  2.30363468e-04, -2.12480723e-04, -1.17788491e-04,
         5.86174712e-04, -6.78812087e-04,  4.85626746e-04, -1.17917644e-04,
        -1.48260191e-05,  3.53218137e-06]),
  'hess': None,
  'res': None,
  'sres': None,
  'n_fval': 68,
  'n_grad': 68,
  'n_hess': 0,
  'n_res': 0,
  'n_sres': 0,
  'x0': array([-0.99326274,  0.99660589,  0.99824031,  0.99898835,  0.99922654,
         0.99907324,  0.99845446,  0.99705616,  0.99417916,  0.98839222]),
  'fval0': 58537.15752301021,
  'history': None,
  'exitflag': 0,
  'time': 0.007930994033813477,
  'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'}]
```

2.6.5 Plot results

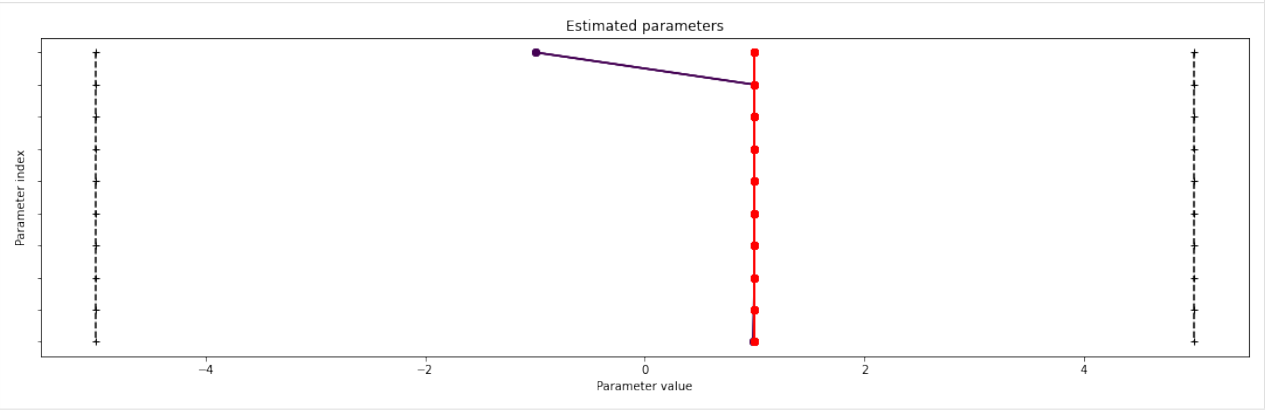
[9]:
```
# plot waterfalls
pypesto.visualize.waterfall(result, size=(15,6))
```

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fefec49cbd0>
```



[10]:
```
pypesto.visualize.parameters(result,
                             balance_alpha=False)
```

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fefebca04d0>
```



[ ]:
```
```

2.7 A sampler study

In this notebook, we perform a short study of how various samplers implemented in pyPESTO perform.

2.7.1 The pipeline

First, we show a typical workflow, fully integrating the samplers with a PEtab problem, using a toy example of a conversion reaction.

[1]:
```
import pypesto
import petab

# import to petab
```

(continues on next page)

(continued from previous page)

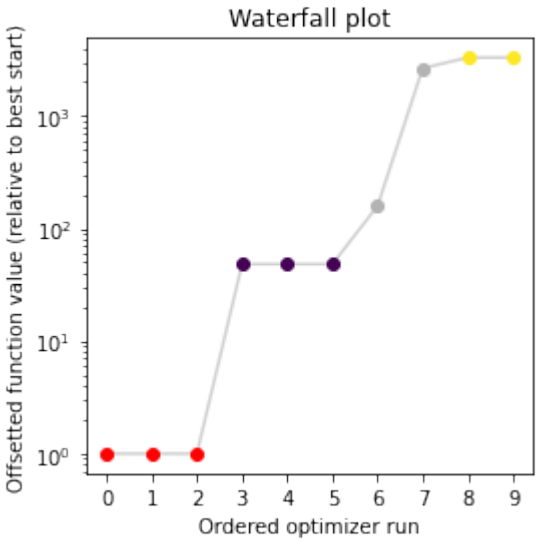
```
petab_problem = petab.Problem.from_yaml(
    "conversion_reaction/conversion_reaction.yaml")
# import to pypesto
importer = pypesto.PetabImporter(petab_problem)
# create problem
problem = importer.create_problem()
```

Commonly, as a first step, optimization is performed, in order to find good parameter point estimates.

[2]:
```
result = pypesto.minimize(problem, n_starts=10)
```

[3]:
```
pypesto.visualize.waterfall(result, size=(4,4))
```

[3]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1ccd9d8090>
```



Next, we perform sampling. Here, we employ a `pypesto.sample.AdaptiveParallelTemperingSampler` sampler, which runs Markov Chain Monte Carlo (MCMC) chains on different temperatures. For each chain, we employ a `pypesto.sample.AdaptiveMetropolisSampler`. For more on the samplers see below or the API documentation.

[4]:
```
sampler = pypesto.AdaptiveParallelTemperingSampler(
    internal_sampler=pypesto.AdaptiveMetropolisSampler(),
    n_chains=3)
```

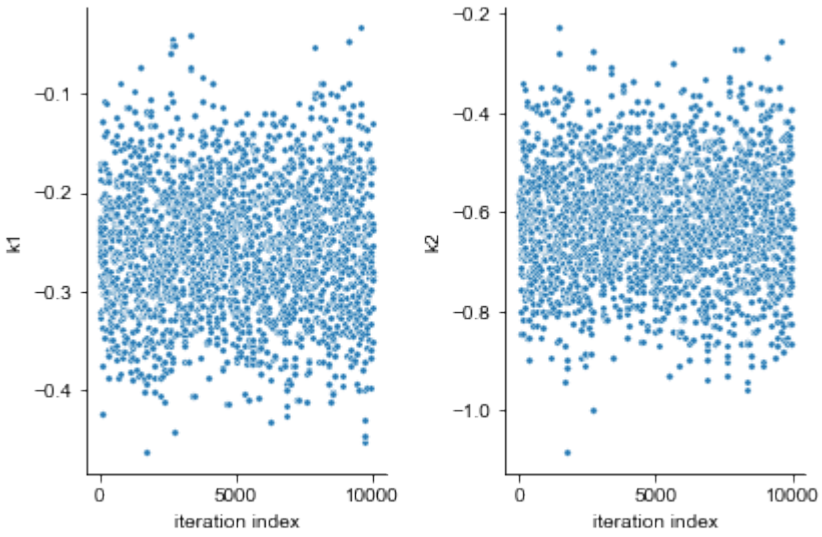
For the actual sampling, we call the `pypesto.sample` function. By passing the result object to the function, the previously found global optimum is used as starting point for the MCMC sampling.

[5]:
```
result = pypesto.sample(problem, n_samples=10000, sampler=sampler, result=result)
```

When the sampling is finished, we can analyse our results. pyPESTO provides functions to analyse both the sampling process as well as the obtained sampling result. Visualizing the traces e.g. allows to detect burn-in phases, or fine-tune hyperparameters. First, the parameter trajectories can be visualized:

[6]:
```
pypesto.visualize.sampling_parameters_trace(result, use_problem_bounds=False)
```

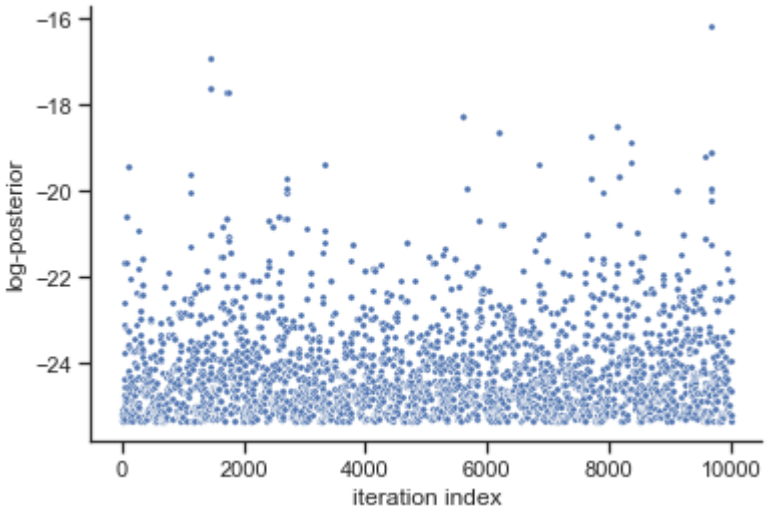
[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1cccf78dd0>
```



Next, also the log posterior trace can be visualized:

[7]:
```
pypesto.visualize.sampling_fval_trace(result)
```

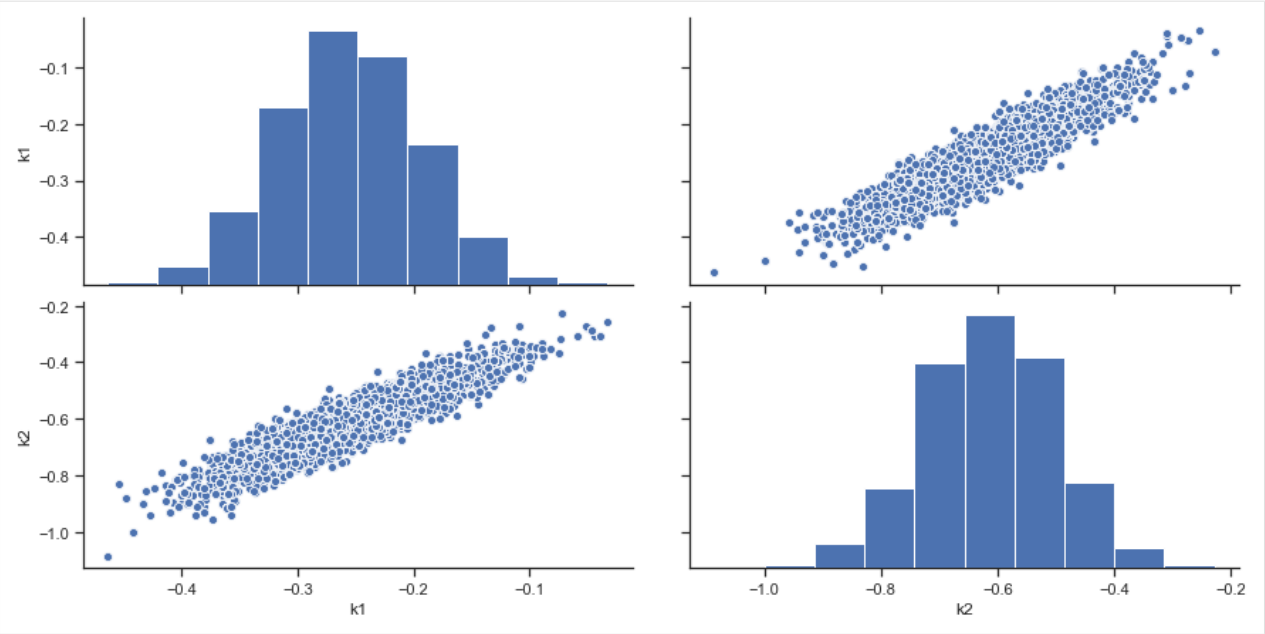
[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1cccf16b90>
```



To visualize the result, there are various options. The scatter plot shows histograms of 1-dim parameter marginals and scatter plots of 2-dimensional parameter combinations:

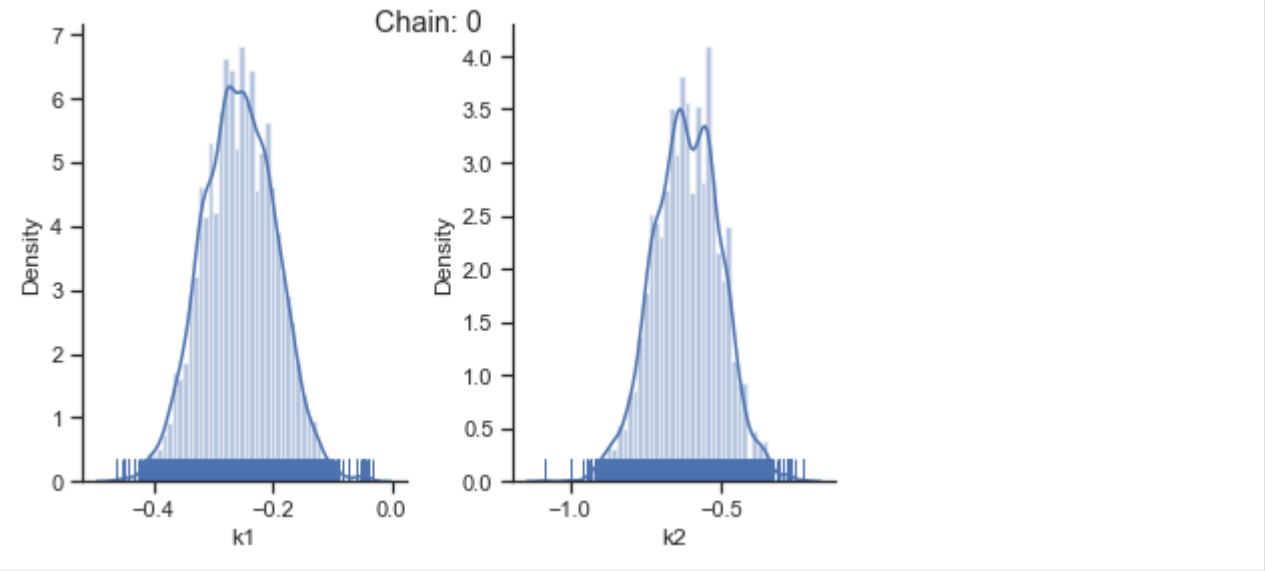
[8]:
```
pypesto.visualize.sampling_scatter(result, size=[13,6])
```

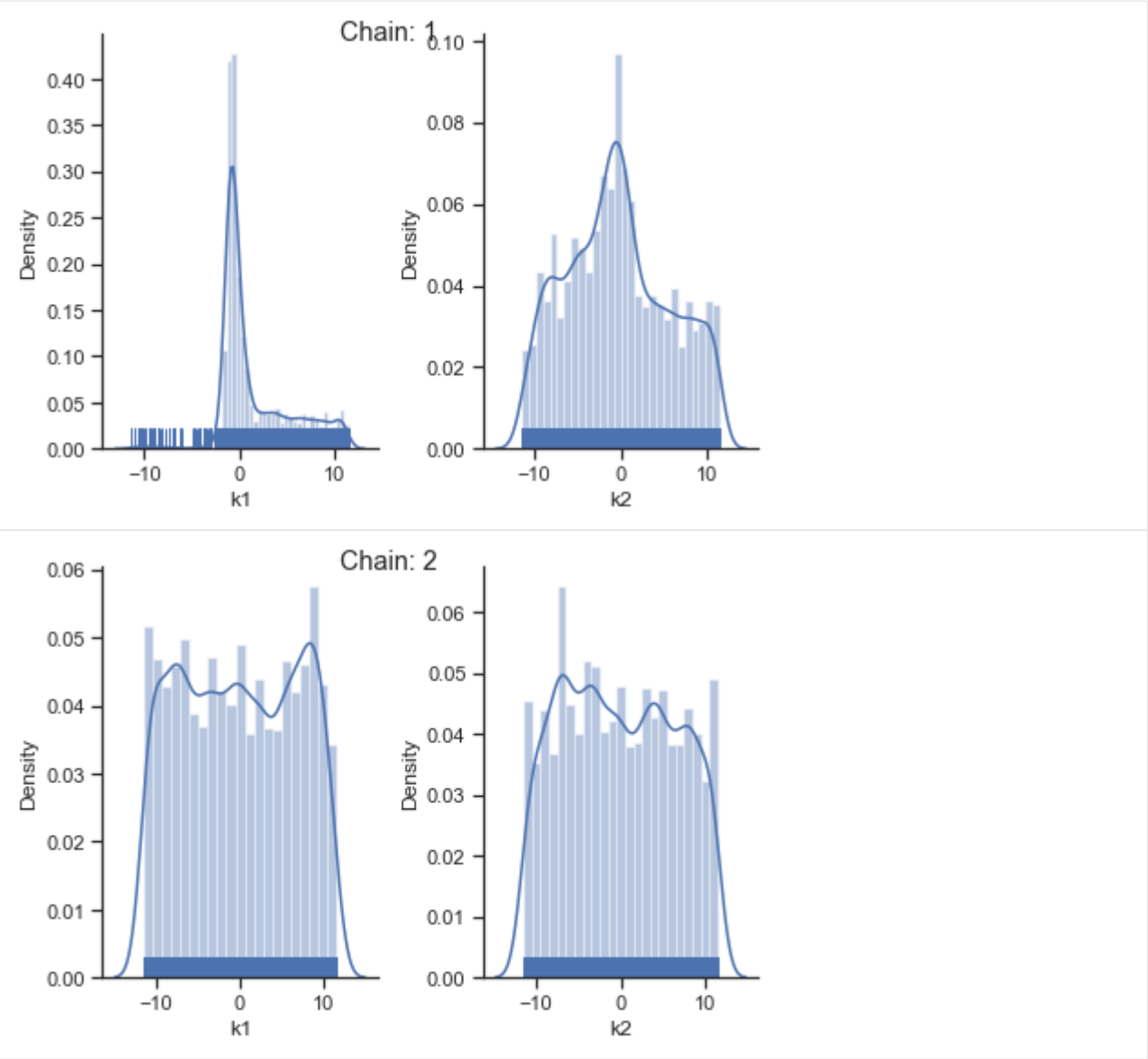
[8]:
```
<seaborn.axisgrid.PairGrid at 0x7f1d12243cd0>
```



`sampling_1d_marginals` allows to plot e.g. kernel density estimates or histograms (internally using seaborn):

[9]:
```
for i_chain in range(len(result.sample_result.betas)):
    pypesto.visualize.sampling_1d_marginals(
        result, i_chain=i_chain, suptitle=f"Chain: {i_chain}")
```





That's it for the moment on using the sampling pipeline.

2.7.2 1-dim test problem

To compare and test the various implemented samplers, we first study a 1-dimensional test problem of a gaussian mixture density, together with a flat prior.

[10]:
```
import numpy as np
from scipy.stats import multivariate_normal
import seaborn as sns
import pypesto

def density(x):
    return 0.3*multivariate_normal.pdf(x, mean=-1.5, cov=0.1) + \
        0.7*multivariate_normal.pdf(x, mean=2.5, cov=0.2)
```

(continues on next page)

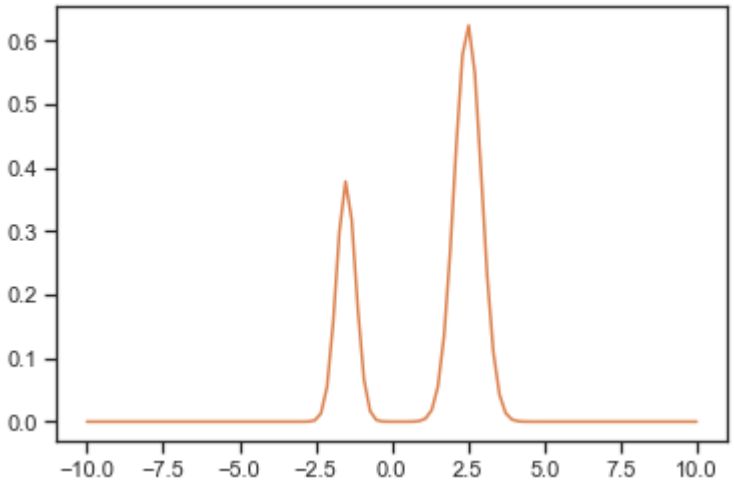
(continued from previous page)

```
def p(x):
    return - np.log(density(x))

objective = pypesto.Objective(fun=p)
problem = pypesto.Problem(
    objective=objective, lb=np.array(-10), ub=np.array(10), x_names=['x'])
```

The likelihood has two separate modes:

[11]:
```
xs = np.linspace(-10, 10, 100)
ys = [density(x) for x in xs]

sns.lineplot(xs, ys, color='C1')
```

[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1cbb116550>
```

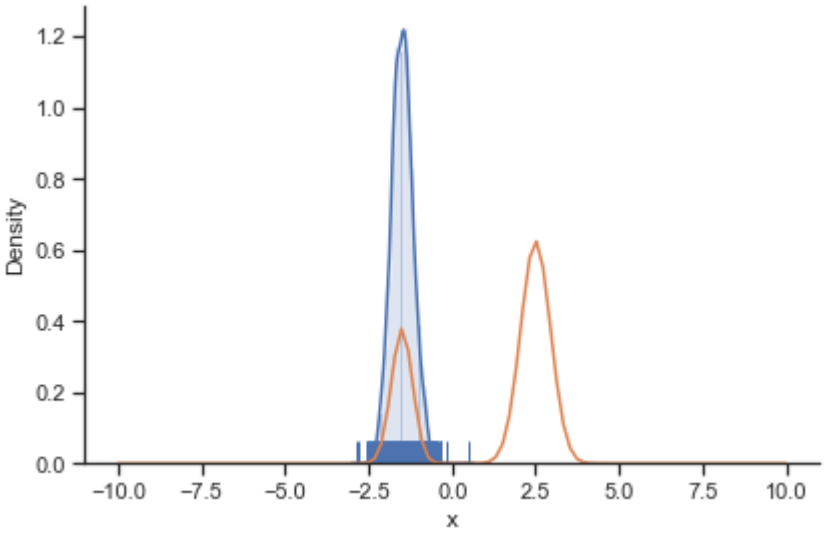


Metropolis sampler

For this problem, let us try out the simplest sampler, the `pypesto.sample.MetropolisSampler`.

[12]:
```
sampler = pypesto.MetropolisSampler({'std': 0.5})
result = pypesto.sample(problem, 1e4, sampler, x0=np.array([0.5]))

ax = pypesto.visualize.sampling_1d_marginals(result)
ax[0][0].plot(xs, ys)
```

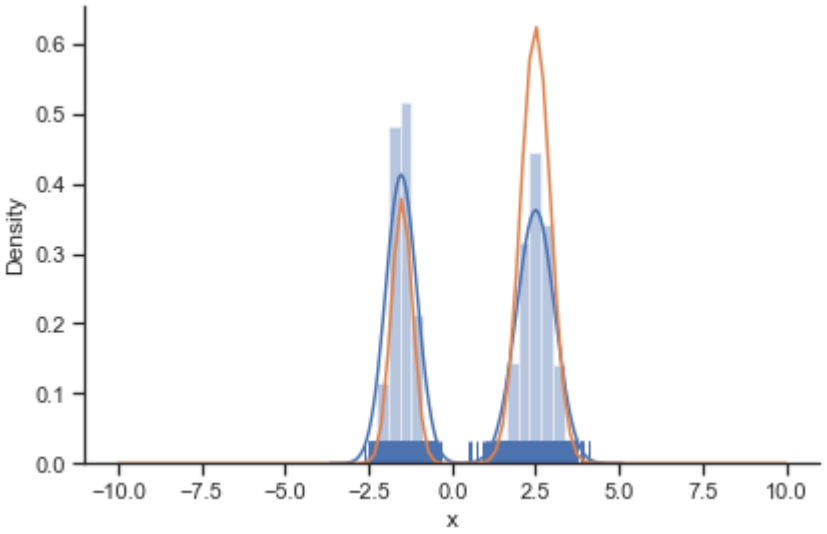
[12]:
```
[<matplotlib.lines.Line2D at 0x7f1cccffaa50>]
```



The obtained posterior does not accurately represent the distribution, often only capturing one mode. This is because it is hard for the Markov chain to jump between the distribution's two modes. This can be fixed by choosing a higher proposal variation `std`:

[13]:
```
sampler = pypesto.MetropolisSampler({'std': 1})
result = pypesto.sample(problem, 1e4, sampler, x0=np.array([0.5]))

ax = pypesto.visualize.sampling_1d_marginals(result)
ax[0][0].plot(xs, ys)
```

[13]:
```
[<matplotlib.lines.Line2D at 0x7f1cba518750>]
```



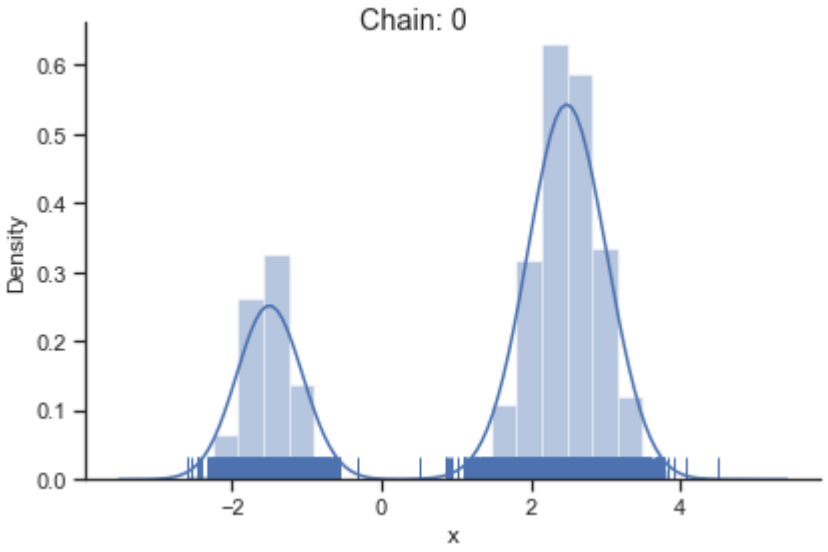
In general, MCMC have difficulties exploring multimodel landscapes. One way to overcome this is to used parallel tempering. There, various chains are run, lifting the densities to different temperatures. At high temperatures, proposed steps are more likely to get accepted and thus jumps between modes more likely.

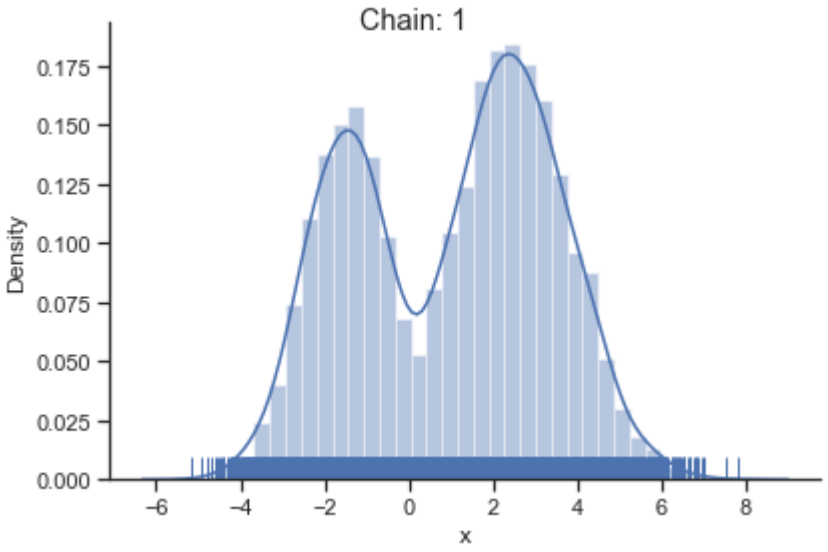
Parallel tempering sampler

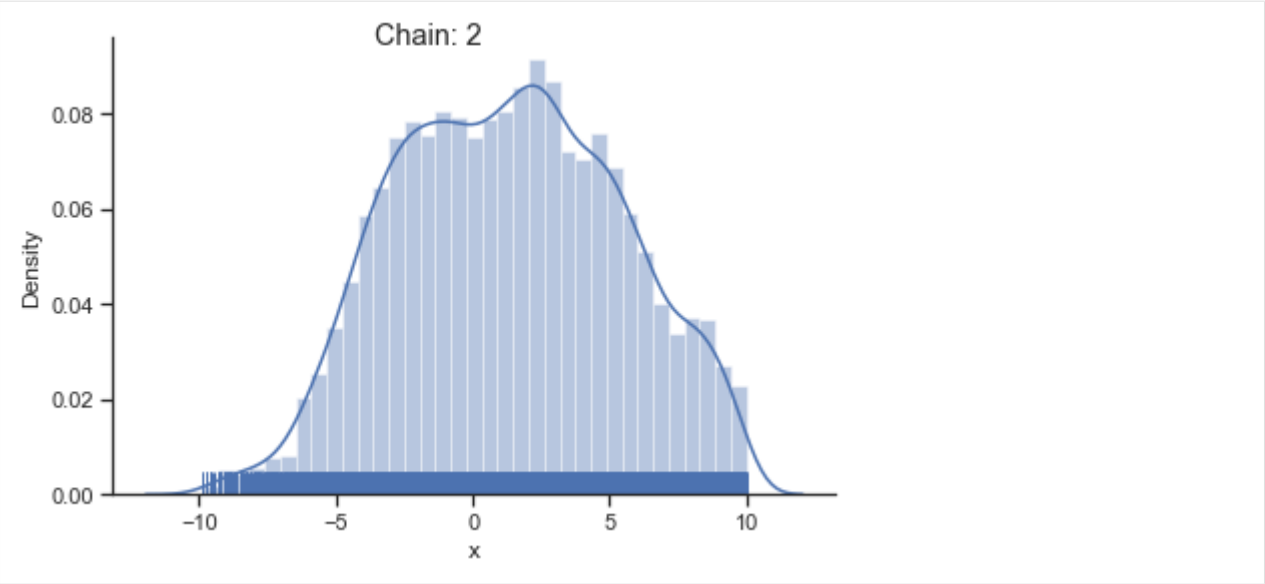
In pyPESTO, the most basic parallel tempering algorithm is the `pypesto.sample.ParallelTemperingSampler`. It takes an `internal_sampler` parameter, to specify what sampler to use for performing sampling the different chains. Further, we can directly specify what inverse temperatures `betas` to use. When not specifying the `betas` explicitly but just the number of chains `n_chains`, an established near-exponential decay scheme is used.

[14]:
```
sampler = pypesto.ParallelTemperingSampler(
    internal_sampler=pypesto.MetropolisSampler(),
    betas=[1, 1e-1, 1e-2])
result = pypesto.sample(problem, 1e4, sampler, x0=np.array([0.5]))
```

[15]:
```
for i_chain in range(len(result.sample_result.betas)):
    pypesto.visualize.sampling_1d_marginals(
        result, i_chain=i_chain, suptitle=f"Chain: {i_chain}")
```







Of interest is here finally the first chain at index `i_chain=0`, which approximates the posterior well.

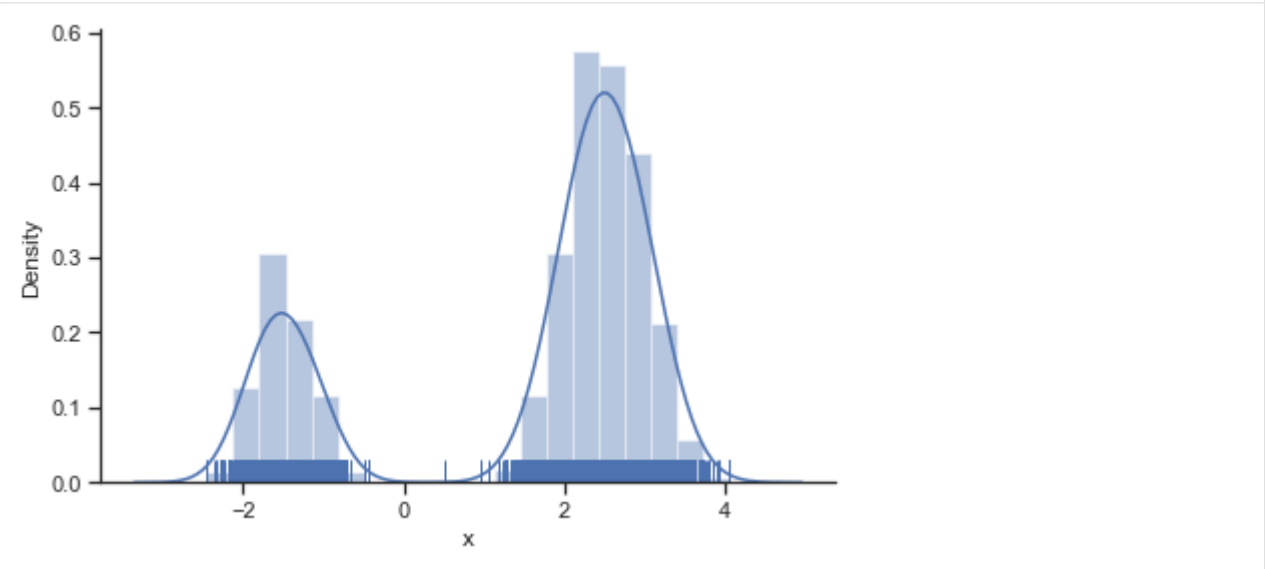
Adaptive Metropolis sampler

The problem of having to specify the proposal step variation manually can be overcome by using the `pypesto.sample.AdaptiveMetropolisSampler`, which iteratively adjusts the proposal steps to the function landscape.

[16]:
```
sampler = pypesto.AdaptiveMetropolisSampler()
result = pypesto.sample(problem, 1e4, sampler, x0=np.array([0.5]))
```

[17]:
```
pypesto.visualize.sampling_1d_marginals(result)
```

[17]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f1cbc3c6290>]],
      dtype=object)
```

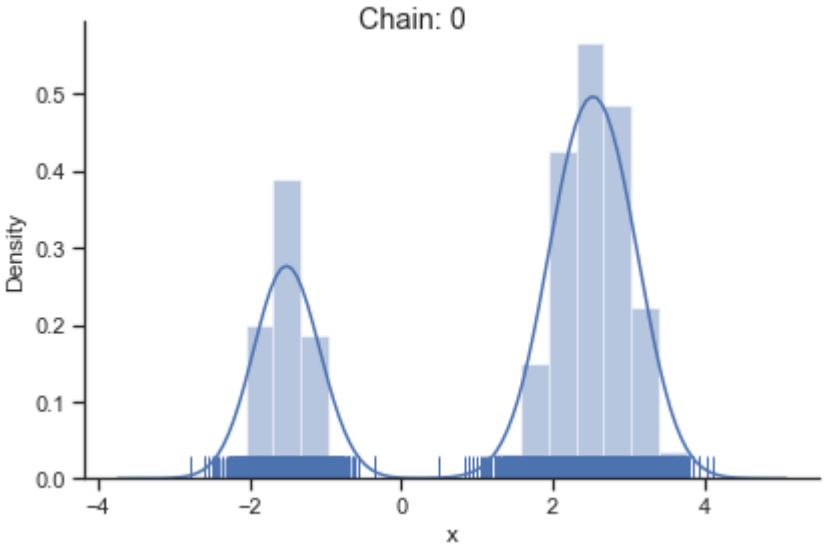


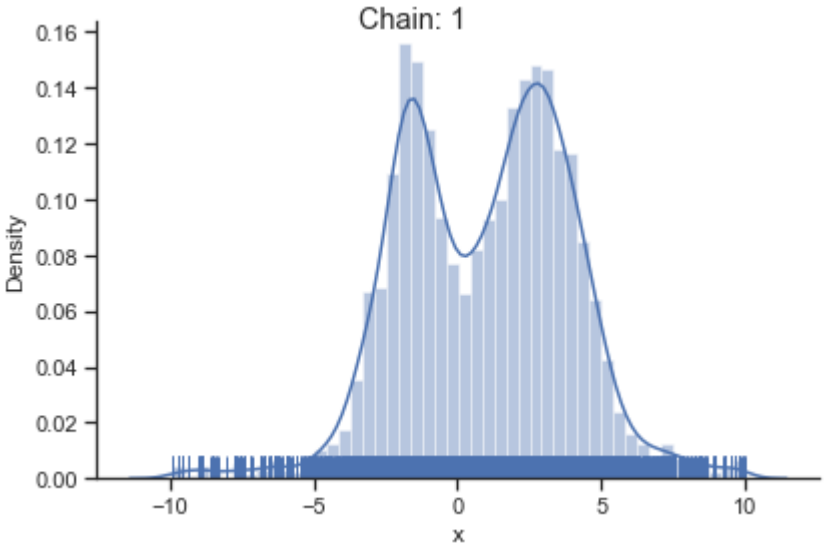
Adaptive parallel tempering sampler

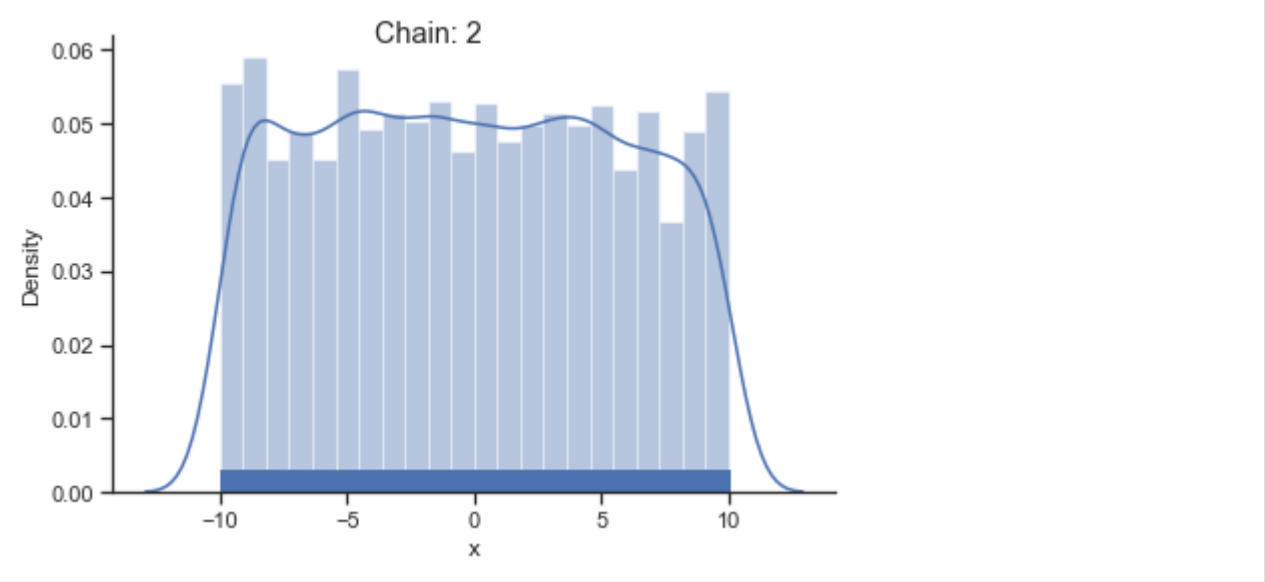
The `pypesto.sample.AdaptiveParallelTemperingSampler` iteratively adjusts the temperatures to obtain good swapping rates between chains.

[18]:
```
sampler = pypesto.AdaptiveParallelTemperingSampler(
    internal_sampler=pypesto.AdaptiveMetropolisSampler(), n_chains=3)
result = pypesto.sample(problem, 1e4, sampler, x0=np.array([0.5]))
```

[19]:
```
for i_chain in range(len(result.sample_result.betas)):
    pypesto.visualize.sampling_1d_marginals(
        result, i_chain=i_chain, suptitle=f"Chain: {i_chain}")
```







[20]:
```
result.sample_result.betas
```

[20]:
```
array([1.00000000e+00, 8.02757714e-02, 2.00000000e-05])
```

2.7.3 2-dim test problem: Rosenbrock banana

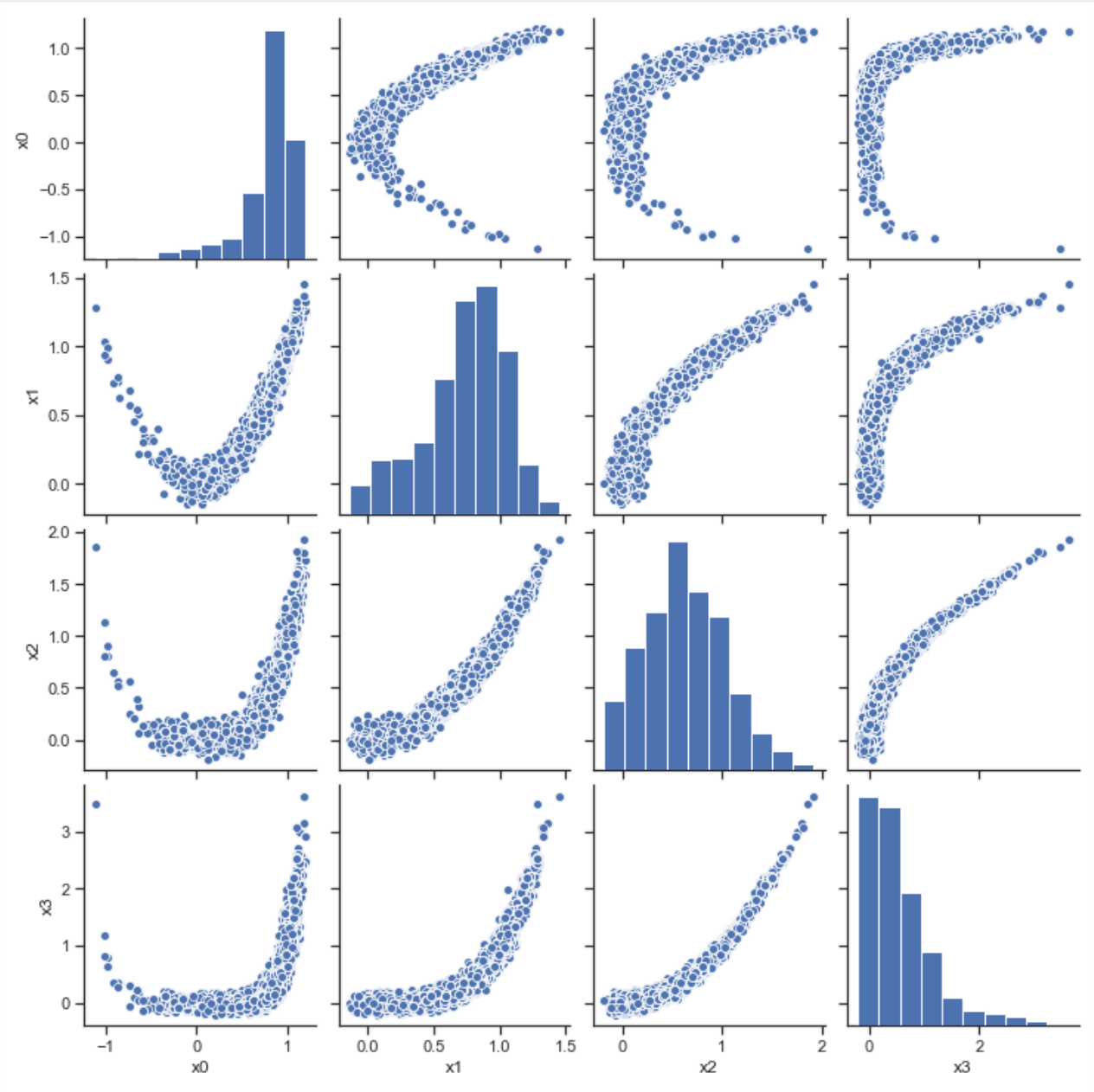
The adaptive parallel tempering sampler with chains running adaptive Metropolis samplers is also able to sample from more challenging posterior distributions. To illustrates this shortly, we use the Rosenbrock function.

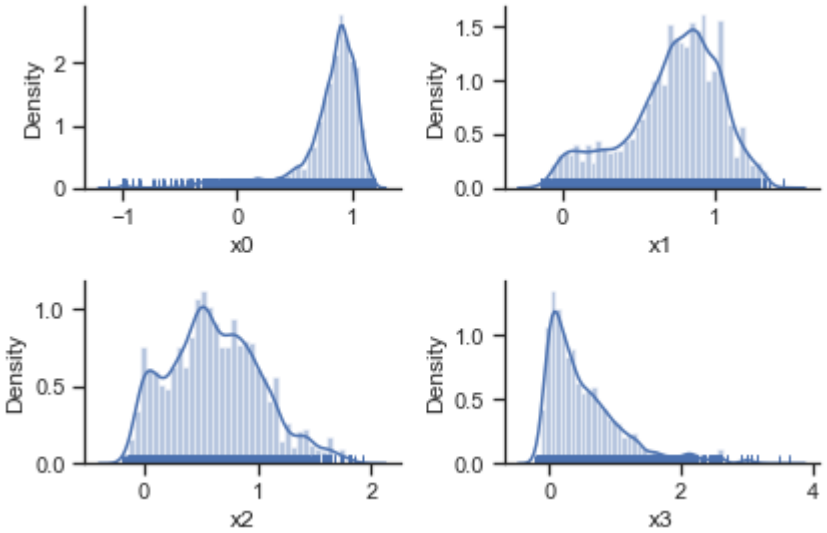
[21]:
```
import scipy.optimize as so
import pypesto

# first type of objective
objective = pypesto.Objective(fun=so.rosen)

dim_full = 4
lb = -5 * np.ones((dim_full, 1))
ub = 5 * np.ones((dim_full, 1))

problem = pypesto.Problem(objective=objective, lb=lb, ub=ub)
```

[23]:
```
sampler = pypesto.AdaptiveParallelTemperingSampler(
    internal_sampler=pypesto.AdaptiveMetropolisSampler(), n_chains=10)
result = pypesto.sample(problem, 1e4, sampler, x0=np.zeros(dim_full))
```

[24]:
```
pypesto.visualize.sampling_scatter(result)
pypesto.visualize.sampling_1d_marginals(result)
```

[24]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f1cbb524390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1cbc726250>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f1cbcb1d310>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1cbcac95d0>]],
      dtype=object)
```


x0
x1
x2
x3
1.0
0.5
0.0
−0.5
−1.0
1.5
2.0
3
2
1
0
−1
0.0
0.5
1.0
1.5
2



[ ]:

2.8 Download the examples as notebooks

- `Rosenbrock`
- `Conversion reaction`
- `Fixed parameters`
- `Boehm model`
- `Petab import`
- `HDF5 storage`
- `Sampler study`

Note: Some of the notebooks have extra dependencies.

CHAPTER 3

Contribute

3.1 Contribute documentation

To make pypesto easily usable, we are committed to documenting extensively. This involves in particular documenting the functionality of methods and classes, the purpose of single lines of code, and giving usage examples. The documentation is hosted on pypesto.readthedocs.io and updated automatically every time the master branch on github.com/icb-dcm/pypesto is updated. To compile the documentation locally, use:

```
cd doc
make html
```

3.2 Contribute tests

Tests are located in the `test` folder. All files starting with `test_` contain tests and are automatically run on Travis CI. To run them manually, type:

```
python3 -m pytest test
```

or alternatively:

```
python3 -m unittest test
```

You can also run specific tests.

Tests can be written with pytest or the unittest module.

3.2.1 PEP8

We try to respect the PEP8 coding standards. We run flake8 as part of the tests. If flake8 complains, the tests won't pass. You can run it via:

```
./run_flake8.sh
```

in Linux from the base directory, or directly from python. More, you can use the tool autopep8 to automatically fix various coding issues.

3.3 Contribute code

If you start working on a new feature or a fix, if not already done, please create an issue on github shortly describing your plans and assign it to yourself.

To get your code merged, please:

1. create a pull request to develop
2. if not already done in a commit message already, use the pull request description to reference and automatically close the respective issue (see https://help.github.com/articles/closing-issues-using-keywords/)
3. check that all tests on travis pass
4. check that the documentation is up-to-date
5. request a code review

General notes:

- Internally, we use `numpy` for arrays. In particular, vectors are represented as arrays of shape (n,).
- Use informative commmit messages.

CHAPTER 4

Deploy

New features and bug fixes are continuously added to the develop branch. On every merge to master, the version number in `pypesto/version.py` should be incremented as described below.

4.1 Versioning scheme

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.

4.2 Creating a new release

After new commits have been added to the develop branch, changes can be merged to master and a new version of pyPESTO can be released. Every merge to master should coincide with an incremented version number and a git tag on the respective merge commit.

4.2.1 Merge into master

1. create a pull request from develop to master
2. check that all tests on travis pass
3. check that the documentation is up-to-date
4. adapt the version number in the file `pesto/version.py` (see above)
5. update the release notes in `doc/releasenotes.rst`
6. request a code review

7. merge into the origin master branch

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

4.2.2 Creating a release on github

After merging into master, create a new release on Github. In the release form:

- specify a tag with the new version as specified in `pesto/version.py`, prefixed with `v` (e.g. `v0.0.1`)
- include the latest additions to `doc/releasenotes.rst` in the release description

Tagging the release commit will automatically trigger deployment of the new version to pypi.

CHAPTER 5

Objective

class pypesto.objective.**AggregatedObjective**(*objectives: List[pypesto.objective.objective.Objective], x_names: List[str] = None*)

Bases: pypesto.objective.objective.Objective

This class aggregates multiple objectives into one objective.

__call__()

Method to obtain arbitrary sensitivities. This is the central method which is always called, also by the get_* methods.

There are different ways in which an optimizer calls the objective function, and in how the objective function provides information (e.g. derivatives via separate functions or along with the function values). The different calling modes increase efficiency in space and time and make the objective flexible.

Parameters

- **x** – The parameters for which to evaluate the objective function.
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.
- **return_dict** – If False (default), the result is a Tuple of the requested values in the requested order. Tuples of length one are flattened. If True, instead a dict is returned which can carry further information.

Returns By default, this is a tuple of the requested function values and derivatives in the requested order (if only 1 value, the tuple is flattened). If *return_dict*, then instead a dict is returned with function values and derivatives indicated by ids.

Return type result

__class__

alias of builtins.type

__deepcopy__(*memodict=None*)

__delattr__

Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.objective.aggregated', '__doc__': '\n '

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*objectives: List[pypesto.objective.objective.Objective], x_names: List[str] = None*)
Constructor.

Parameters **objectives** (*list*) – List of pypesto.objetive instances

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.objective.aggregated'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

aggregate_fun(*x*)

aggregate_fun_sensi_orders(*x*, *sensi_orders*)

aggregate_grad(*x*)

aggregate_hess(*x*)

aggregate_hessp(*x*)

aggregate_res(*x*)

aggregate_res_sensi_orders(*x*, *sensi_orders*)

aggregate_sres(*x*)

check_grad(*x: numpy.ndarray*, *x_indices: List[int] = None*, *eps: float = 1e-05*, *verbosity: int = 1*, *mode: str = 'mode_fun'*) → pandas.core.frame.DataFrame
Compare gradient evaluation: Firstly approximate via finite differences, and secondly use the objective gradient.

Parameters

- **x** – The parameters for which to evaluate the gradient.
- **x_indices** – List of index values for which to compute gradients. Default: all.
- **eps** – Finite differences step size. Default: 1e-5.
- **verbosity** – Level of verbosity for function output. * 0: no output, * 1: summary for all parameters, * 2: summary for individual parameters. Default: 1.
- **mode** – Residual (MODE_RES) or objective function value (MODE_FUN, default) computation mode.

Returns gradient, finite difference approximations and error estimates.

Return type result

check_sensi_orders(*sensi_orders*, *mode*) → None
Check if the objective is able to compute the requested sensitivities. If not, throw an exception.

Raises

- ValueError if the objective function cannot be called as
- `requested.`

get_fval(*x: numpy.ndarray*) → float
Get the function value at x.

get_grad(*x: numpy.ndarray*) → numpy.ndarray
Get the gradient at x.

get_hess(*x: numpy.ndarray*) → numpy.ndarray
Get the Hessian at x.

get_res(*x: numpy.ndarray*) → numpy.ndarray
Get the residuals at x.

get_sres(*x: numpy.ndarray*) → numpy.ndarray
Get the residual sensitivities at x.

has_fun

has_grad

has_hess

has_hessp

has_res

has_sres

initialize()
Initialize the objective function. This function is used at the beginning of an analysis, e.g. optimization, and can e.g. reset the objective memory. By default does nothing.

static output_to_dict()
Convert output tuple to dict.

static output_to_tuple()
Return values as requested by the caller, since usually only a subset is demanded. One output is returned as-is, more than one output are returned as a tuple in order (fval, grad, hess).

reset_steadystate_guesses()
Propagates reset_steadystate_guesses() to child objectives if available (currently only applies for amici_objective)

update_from_problem(*dim_full: int, x_free_indices: List[int], x_fixed_indices: List[int], x_fixed_vals: List[int]*)
Handle fixed parameters. Later, the objective will be given parameter vectors x of dimension dim, which have to be filled up with fixed parameter values to form a vector of dimension dim_full >= dim. This vector is then used to compute function value and derivatives. The derivatives must later be reduced again to dimension dim.

This is so as to make the fixing of parameters transparent to the caller.

The methods preprocess, postprocess are overwritten for the above functionality, respectively.

Parameters

- **dim_full** – Dimension of the full vector including fixed parameters.
- **x_free_indices** – Vector containing the indices (zero-based) of free parameters (complimentary to x_fixed_indices).
- **x_fixed_indices** – Vector containing the indices (zero-based) of parameter components that are not to be optimized.
- **x_fixed_vals** – Vector of the same length as x_fixed_indices, containing the values of the fixed parameters.

class pypesto.objective.**AmiciCalculator**
Bases: object

Class to perform the actual call to AMICI and obtain requested objective function values.

__call__(*x_dct: Dict, sensi_order: int, mode: str, amici_model: Union[amici.Model, amici.ModelPtr], amici_solver: Union[amici.Solver, amici.SolverPtr], edatas: List[amici.ExpData], n_threads: int, x_ids: Sequence[str], parameter_mapping: ParameterMapping*)

Perform the actual AMICI call.

Called within the *AmiciObjective.__call__()* method.

Parameters

- **x_dct** – Parameters for which to compute function value and derivatives.
- **sensi_order** – Maximum sensitivity order.
- **mode** – Call mode (function value or residual based).
- **amici_model** – The AMICI model.
- **amici_solver** – The AMICI solver.
- **edatas** – The experimental data.
- **n_threads** – Number of threads for AMICI call.
- **x_ids** – Ids of optimization parameters.
- **parameter_mapping** – Mapping of optimization to simulation parameters.

__class__
alias of builtins.type

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.objective.amici_calculator', '__doc__'

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.objective.amici_calculator'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

initialize()
Initialize the calculator. Default: Do nothing.

class pypesto.objective.**AmiciObjectBuilder**
Bases: abc.ABC

Allows to build AMICI model, solver, and edatas.

This class is useful for pickling an pypesto.AmiciObjective, which is required in some parallelization schemes. Therefore, this class itself must be picklable.

__abstractmethods__ = frozenset({'create_edatas', 'create_solver', 'create_model'})

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.objective.amici_objective', '__doc__':

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.objective.amici_objective'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

create_edatas(*model: Union[amici.Model, amici.ModelPtr]*) → Sequence[amici.ExpData]
Create AMICI experimental data.

create_model() → Union[amici.Model, amici.ModelPtr]
Create an AMICI model.

create_solver(*model: Union[amici.Model, amici.ModelPtr]*) → Union[amici.Solver, amici.SolverPtr]
Create an AMICI solver.

class pypesto.objective.**AmiciObjective**(*amici_model: Union[amici.Model, amici.ModelPtr], amici_solver: Union[amici.Solver, amici.SolverPtr], edatas: Union[Sequence[amici.ExpData], amici.ExpData], max_sensi_order: int = None, x_ids: Sequence[str] = None, x_names: Sequence[str] = None, parameter_mapping: ParameterMapping = None, guess_steadystate: bool = True, n_threads: int = 1, amici_object_builder: pypesto.objective.amici_objective.AmiciObjectBuilder = None, calculator: pypesto.objective.amici_calculator.AmiciCalculator = None*)
Bases: pypesto.objective.objective.Objective

This class allows to create an objective directly from an amici model.

__call__()
Method to obtain arbitrary sensitivities. This is the central method which is always called, also by the get_* methods.

There are different ways in which an optimizer calls the objective function, and in how the objective function provides information (e.g. derivatives via separate functions or along with the function values). The different calling modes increase efficiency in space and time and make the objective flexible.

Parameters

- **x** – The parameters for which to evaluate the objective function.
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.
- **return_dict** – If False (default), the result is a Tuple of the requested values in the requested order. Tuples of length one are flattened. If True, instead a dict is returned which can carry further information.

Returns By default, this is a tuple of the requested function values and derivatives in the requested order (if only 1 value, the tuple is flattened). If *return_dict*, then instead a dict is returned with function values and derivatives indicated by ids.

Return type result

__class__
alias of builtins.type

__deepcopy__(*memodict: Dict = None*) → pypesto.objective.amici_objective.AmiciObjective

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.objective.amici_objective', '__doc__':

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__getstate__() → Dict

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*amici_model: Union[amici.Model, amici.ModelPtr], amici_solver: Union[amici.Solver, amici.SolverPtr], edatas: Union[Sequence[amici.ExpData], amici.ExpData], max_sensi_order: int = None, x_ids: Sequence[str] = None, x_names: Sequence[str] = None, parameter_mapping: ParameterMapping = None, guess_steadystate: bool = True, n_threads: int = 1, amici_object_builder: pypesto.objective.amici_objective.AmiciObjectBuilder = None, calculator: pypesto.objective.amici_calculator.AmiciCalculator = None*)
Constructor.

Parameters

- **amici_model** – The amici model.
- **amici_solver** – The solver to use for the numeric integration of the model.
- **edatas** – The experimental data. If a list is passed, its entries correspond to multiple experimental conditions.
- **max_sensi_order** – Maximum sensitivity order supported by the model. Defaults to 2 if the model was compiled with o2mode, otherwise 1.
- **x_ids** – Ids of optimization parameters. In the simplest case, this will be the AMICI model parameters (default).
- **x_names** – Names of optimization parameters.
- **parameter_mapping** – Mapping of optimization parameters to model parameters. Format as created by *amici.petab_objective.create_parameter_mapping*. The default is just to assume that optimization and simulation parameters coincide.
- **guess_steadystate** – Whether to guess steadystates based on previous steadystates and respective derivatives. This option may lead to unexpected results for models with conservation laws and should accordingly be deactivated for those models.
- **n_threads** – Number of threads that are used for parallelization over experimental conditions. If amici was not installed with openMP support this option will have no effect.
- **amici_object_builder** – AMICI object builder. Allows recreating the objective for pickling, required in some parallelization schemes.
- **calculator** – Performs the actual calculation of the function values and derivatives.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.objective.amici_objective'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__setstate__(*state: Dict*)

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

apply_steadystate_guess(*condition_ix: int*, *x_dct: Dict*)
Use the stored steadystate as well as the respective sensitivity (if available) and parameter value to approximate the steadystate at the current parameters using a zeroth or first order taylor approximation: x_ss(x') = x_ss(x) [+ dx_ss/dx(x)*(x'-x)]

check_grad(*x: numpy.ndarray*, *x_indices: List[int] = None*, *eps: float = 1e-05*, *verbosity: int = 1*, *mode: str = 'mode_fun'*) → pandas.core.frame.DataFrame
Compare gradient evaluation: Firstly approximate via finite differences, and secondly use the objective gradient.

Parameters

- **x** – The parameters for which to evaluate the gradient.
- **x_indices** – List of index values for which to compute gradients. Default: all.

- **eps** – Finite differences step size. Default: 1e-5.
- **verbosity** – Level of verbosity for function output. * 0: no output, * 1: summary for all parameters, * 2: summary for individual parameters. Default: 1.
- **mode** – Residual (MODE_RES) or objective function value (MODE_FUN, default) computation mode.

Returns gradient, finite difference approximations and error estimates.

Return type result

check_sensi_orders(*sensi_orders*, *mode*) → None
Check if the objective is able to compute the requested sensitivities. If not, throw an exception.

Raises

- ValueError if the objective function cannot be called as
- `requested.`

get_bound_fun()
Generate a fun function that calls _call_amici with MODE_FUN. Defining a non-class function that references self as a local variable will bind the function to a copy of the current self object and will accordingly not take future changes to self into account.

get_bound_res()
Generate a res function that calls _call_amici with MODE_RES. Defining a non-class function that references self as a local variable will bind the function to a copy of the current self object and will accordingly not take future changes to self into account.

get_fval(*x: numpy.ndarray*) → float
Get the function value at x.

get_grad(*x: numpy.ndarray*) → numpy.ndarray
Get the gradient at x.

get_hess(*x: numpy.ndarray*) → numpy.ndarray
Get the Hessian at x.

get_res(*x: numpy.ndarray*) → numpy.ndarray
Get the residuals at x.

get_sres(*x: numpy.ndarray*) → numpy.ndarray
Get the residual sensitivities at x.

has_fun

has_grad

has_hess

has_hessp

has_res

has_sres

initialize()
Initialize the objective function. This function is used at the beginning of an analysis, e.g. optimization, and can e.g. reset the objective memory. By default does nothing.

static output_to_dict()
Convert output tuple to dict.

static output_to_tuple()
Return values as requested by the caller, since usually only a subset is demanded. One output is returned as-is, more than one output are returned as a tuple in order (fval, grad, hess).

par_arr_to_dct(*x: Sequence[float]*) → Dict[str, float]
Create dict from parameter vector.

rebind_fun()
Replace the current fun function with one that is bound to the current instance

rebind_res()
Replace the current res function with one that is bound to the current instance

reset_steadystate_guesses()
Resets all steadystate guess data

store_steadystate_guess(*condition_ix: int*, *x_dct: Dict*, *rdata: amici.ReturnData*)
Store condition parameter, steadystate and steadystate sensitivity in steadystate_guesses if steadystate guesses are enabled for this condition

update_from_problem(*dim_full: int, x_free_indices: List[int], x_fixed_indices: List[int], x_fixed_vals: List[int]*)
Handle fixed parameters. Later, the objective will be given parameter vectors x of dimension dim, which have to be filled up with fixed parameter values to form a vector of dimension dim_full >= dim. This vector is then used to compute function value and derivatives. The derivatives must later be reduced again to dimension dim.

This is so as to make the fixing of parameters transparent to the caller.

The methods preprocess, postprocess are overwritten for the above functionality, respectively.

Parameters

- **dim_full** – Dimension of the full vector including fixed parameters.
- **x_free_indices** – Vector containing the indices (zero-based) of free parameters (complimentary to x_fixed_indices).
- **x_fixed_indices** – Vector containing the indices (zero-based) of parameter components that are not to be optimized.
- **x_fixed_vals** – Vector of the same length as x_fixed_indices, containing the values of the fixed parameters.

class pypesto.objective.**CsvHistory**(*file: str*, *x_names: Iterable[str] = None*, *options: Dict = None*)
Bases: pypesto.objective.history.History

Stores a representation of the history in a CSV file.

Parameters

- **file** – CSV file name.
- **x_names** – Parameter names.
- **options** – History options.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.objective.history', '__doc__': 'Stores

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*file: str*, *x_names: Iterable[str] = None*, *options: Dict = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.objective.history'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

finalize()
Finalize history. Called after a run.

get_chi2_trace() → Sequence[numpy.ndarray]
Chi2 value trace.

get_fval_trace() → pandas.core.series.Series
Function value trace.

get_grad_trace() → Sequence[numpy.ndarray]
Gradient trace.

get_hess_trace() → Sequence[numpy.ndarray]
Hessian trace.

get_res_trace() → Sequence[numpy.ndarray]
Residual trace.

get_schi2_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Chi2 value sensitivity trace.

get_sres_trace() → Sequence[numpy.ndarray]
Residual sensitivity trace.

get_time_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Execution time trace.

get_x_trace() → Sequence[numpy.ndarray]
Parameter trace.

n_fval
Number of function evaluations.

n_grad
Number of gradient evaluations.

n_hess
Number of Hessian evaluations.

n_res
Number of residual evaluations.

n_sres
Number or residual sensitivity evaluations.

start_time
Start time.

update(*x: numpy.ndarray, sensi_orders: Tuple[int, ...], mode: str, result: Dict[str, Union[float, numpy.ndarray]]*) → None
Update history after a function evaluation.

Parameters

- **x** – The parameter vector.

- **sensi_orders** – The sensitivity orders computed.
- **mode** – The objective function mode computed (function value or residuals).
- **result** – The objective function values for parameters *x*, sensitivities *sensi_orders* and mode *mode*.

class pypesto.objective.**Hdf5History**(*id: str*, *file: str*, *options: Dict = None*)

Bases: pypesto.objective.history.History

Stores a representation of the history in an HDF5 file.

Parameters

- **id** – Id of the history
- **file** – HDF5 file name.
- **options** – History options.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.objective.history', '__doc__': 'Stores

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*id: str*, *file: str*, *options: Dict = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.objective.history'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

finalize()
Finalize history. Called after a run.

get_chi2_trace() → Sequence[numpy.ndarray]
Chi2 value trace.

get_fval_trace() → Sequence[float]
Function value trace.

get_grad_trace() → Sequence[numpy.ndarray]
Gradient trace.

get_hess_trace() → Sequence[numpy.ndarray]
Hessian trace.

get_res_trace() → Sequence[numpy.ndarray]
Residual trace.

get_schi2_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Chi2 value sensitivity trace.

get_sres_trace() → Sequence[numpy.ndarray]
Residual sensitivity trace.

get_time_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Execution time trace.

get_x_trace() → Sequence[numpy.ndarray]
Parameter trace.

n_fval
Number of function evaluations.

n_grad
Number of gradient evaluations.

n_hess
Number of Hessian evaluations.

n_res
Number of residual evaluations.

n_sres
Number or residual sensitivity evaluations.

start_time
Start time.

update(*x: numpy.ndarray, sensi_orders: Tuple[int, ...], mode: str, result: Dict[str, Union[float, numpy.ndarray]]*) → None
Update history after a function evaluation.

Parameters

- **x** – The parameter vector.
- **sensi_orders** – The sensitivity orders computed.
- **mode** – The objective function mode computed (function value or residuals).
- **result** – The objective function values for parameters *x*, sensitivities *sensi_orders* and mode *mode*.

class pypesto.objective.**History**(*options: Dict = None*)
Bases: pypesto.objective.history.HistoryBase

Tracks numbers of function evaluations only, no trace.

Parameters **options** – History options.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.objective.history', '__doc__': 'Tracks

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*options: Dict = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.objective.history'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

finalize()
Finalize history. Called after a run.

get_chi2_trace() → Sequence[numpy.ndarray]
Chi2 value trace.

get_fval_trace() → Sequence[float]
Function value trace.

get_grad_trace() → Sequence[numpy.ndarray]
Gradient trace.

get_hess_trace() → Sequence[numpy.ndarray]
Hessian trace.

get_res_trace() → Sequence[numpy.ndarray]
Residual trace.

get_schi2_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Chi2 value sensitivity trace.

get_sres_trace() → Sequence[numpy.ndarray]
Residual sensitivity trace.

get_time_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Execution time trace.

get_x_trace() → Sequence[numpy.ndarray]
Parameter trace.

n_fval
Number of function evaluations.

n_grad
Number of gradient evaluations.

n_hess
Number of Hessian evaluations.

n_res
Number of residual evaluations.

n_sres
Number or residual sensitivity evaluations.

start_time
Start time.

update(*x: numpy.ndarray, sensi_orders: Tuple[int, ...], mode: str, result: Dict[str, Union[float, numpy.ndarray]]*) → None
Update history after a function evaluation.

Parameters

- **x** – The parameter vector.
- **sensi_orders** – The sensitivity orders computed.
- **mode** – The objective function mode computed (function value or residuals).
- **result** – The objective function values for parameters *x*, sensitivities *sensi_orders* and mode *mode*.

class pypesto.objective.**HistoryBase**
Bases: abc.ABC

Abstract base class for history objects.

Can be used as a dummy history, but does not implement any history functionality.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.objective.history', '__doc__': 'Abstra

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.objective.history'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

finalize()
Finalize history. Called after a run.

get_chi2_trace() → Sequence[numpy.ndarray]
Chi2 value trace.

get_fval_trace() → Sequence[float]
Function value trace.

get_grad_trace() → Sequence[numpy.ndarray]
Gradient trace.

get_hess_trace() → Sequence[numpy.ndarray]
Hessian trace.

get_res_trace() → Sequence[numpy.ndarray]
Residual trace.

get_schi2_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Chi2 value sensitivity trace.

get_sres_trace() → Sequence[numpy.ndarray]
Residual sensitivity trace.

get_time_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Execution time trace.

get_x_trace() → Sequence[numpy.ndarray]
Parameter trace.

n_fval
Number of function evaluations.

n_grad
Number of gradient evaluations.

n_hess
Number of Hessian evaluations.

n_res
Number of residual evaluations.

n_sres
Number or residual sensitivity evaluations.

start_time
Start time.

update(*x: numpy.ndarray, sensi_orders: Tuple[int, ...], mode: str, result: Dict[str, Union[float, numpy.ndarray]]*) → None
Update history after a function evaluation.

Parameters

- **x** – The parameter vector.

- **sensi_orders** – The sensitivity orders computed.
- **mode** – The objective function mode computed (function value or residuals).
- **result** – The objective function values for parameters *x*, sensitivities *sensi_orders* and mode *mode*.

class pypesto.objective.**HistoryOptions**(*trace_record: bool = False*, *trace_record_grad: bool = True*, *trace_record_hess: bool = True*, *trace_record_res: bool = True*, *trace_record_sres: bool = True*, *trace_record_chi2: bool = True*, *trace_record_schi2: bool = True*, *trace_save_iter: int = 10*, *storage_file: str = None*)

Bases: dict

Options for the objective that are used in optimization, profiles and sampling.

In addition implements a factory pattern to generate history objects.

Parameters

- **trace_record** – Flag indicating whether to record the trace of function calls. The trace_record_* flags only become effective if trace_record is True. Default: False.
- **trace_record_grad** – Flag indicating whether to record the gradient in the trace. Default: True.
- **trace_record_hess** – Flag indicating whether to record the Hessian in the trace. Default: False.
- **trace_record_res** – Flag indicating whether to record the residual in the trace. Default: False.
- **trace_record_sres** – Flag indicating whether to record the residual sensitivities in the trace. Default: False.
- **trace_record_chi2** – Flag indicating whether to record the chi2 in the trace. Default: True.
- **trace_record_schi2** – Flag indicating whether to record the chi2 sensitivities in the trace. Default: True.
- **trace_save_iter** – After how many iterations to store the trace.
- **storage_file** – File to save the history to. Can be any of None, a "{filename}.csv", or a "{filename}.hdf5" file. Depending on the values, the *create_history* method creates the appropriate object. Occurrences of "{id}" in the file name are replaced by the *id* upon creation of a history, if applicable.

__class__
alias of builtins.type

__contains__()
True if D has a key k, else False.

__delattr__
Delete self[key].

__delitem__
Delete self[key].

__dict__ = mappingproxy({'__module__': 'pypesto.objective.history', '__doc__': '\n Opt

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattr__(*key*)

__getattribute__
Return getattr(self, name).

__getitem__()
x.__getitem__(y) <==> x[y]

__gt__
Return self>value.

__hash__ = None

__init__(*trace_record: bool = False*, *trace_record_grad: bool = True*, *trace_record_hess: bool = True*, *trace_record_res: bool = True*, *trace_record_sres: bool = True*, *trace_record_chi2: bool = True*, *trace_record_schi2: bool = True*, *trace_save_iter: int = 10*, *storage_file: str = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__iter__
Implement iter(self).

__le__
Return self<=value.

__len__
Return len(self).

__lt__
Return self<value.

__module__ = 'pypesto.objective.history'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Set self[key] to value.

__setitem__
Set self[key] to value.

__sizeof__() → size of D in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

static assert_instance(*maybe_options: Union[HistoryOptions, Dict]*) → pypesto.objective.history.HistoryOptions
Returns a valid options object.

Parameters **maybe_options** (*HistoryOptions or dict*) –

clear() → None. Remove all items from D.

copy() → a shallow copy of D

create_history(*id: str, x_names: Iterable[str]*) → pypesto.objective.history.History
Factory method creating a *History* object.

Parameters

- **id** – Identifier for the history.
- **x_names** – Parameter names.

fromkeys()
Returns a new dict with keys from iterable and values equal to value.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class pypesto.objective.**MemoryHistory**(*options: Dict = None*)
Bases: pypesto.objective.history.History

Tracks numbers of function evaluations and keeps an in-memory trace of function evaluations.

Parameters **options** – History options.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.objective.history', '__doc__': 'Tracks

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*options: Dict = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.objective.history'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

finalize()
Finalize history. Called after a run.

get_chi2_trace() → Sequence[numpy.ndarray]
Chi2 value trace.

get_fval_trace() → Sequence[float]
Function value trace.

get_grad_trace() → Sequence[numpy.ndarray]
Gradient trace.

get_hess_trace() → Sequence[numpy.ndarray]
Hessian trace.

get_res_trace() → Sequence[numpy.ndarray]
Residual trace.

get_schi2_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Chi2 value sensitivity trace.

get_sres_trace() → Sequence[numpy.ndarray]
Residual sensitivity trace.

get_time_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Execution time trace.

get_x_trace() → Sequence[numpy.ndarray]
Parameter trace.

n_fval
Number of function evaluations.

n_grad
Number of gradient evaluations.

n_hess
Number of Hessian evaluations.

n_res
Number of residual evaluations.

n_sres
Number or residual sensitivity evaluations.

start_time
Start time.

update(*x: numpy.ndarray, sensi_orders: Tuple[int, ...], mode: str, result: Dict[str, Union[float, numpy.ndarray]]*) → None

Update history after a function evaluation.

Parameters

- **x** – The parameter vector.
- **sensi_orders** – The sensitivity orders computed.
- **mode** – The objective function mode computed (function value or residuals).
- **result** – The objective function values for parameters *x*, sensitivities *sensi_orders* and mode *mode*.

class pypesto.objective.**Objective**(*fun: Callable = None, grad: Union[Callable, bool] = None, hess: Callable = None, hessp: Callable = None, res: Callable = None, sres: Union[Callable, bool] = None, fun_accept_sensi_orders: bool = False, res_accept_sensi_orders: bool = False, x_names: List[str] = None*)

Bases: object

The objective class is a simple wrapper around the objective function, giving a standardized way of calling. Apart from that, it manages several things including fixing of parameters and history.

The objective function is assumed to be in the format of a cost function, log-likelihood function, or log-posterior function. These functions are subject to minimization. For profiling and sampling, the sign is internally flipped, all returned and stored values are however given as returned by this objective function. If maximization is to be performed, the sign should be flipped before creating the objective function.

Parameters

- **fun** – The objective function to be minimized. If it only computes the objective function value, it should be of the form

  ```
  fun(x) -> float
  ```

  where x is an 1-D array with shape (n,), and n is the parameter space dimension.

- **grad** – Method for computing the gradient vector. If it is a callable, it should be of the form

  ```
  grad(x) -> array_like, shape (n,).
  ```

  If its value is True, then fun should return the gradient as a second output.

- **hess** – Method for computing the Hessian matrix. If it is a callable, it should be of the form

  ```
  hess(x) -> array, shape (n,n).
  ```

  If its value is True, then fun should return the gradient as a second, and the Hessian as a third output, and grad should be True as well.

- **hessp** – Method for computing the Hessian vector product, i.e.

  ```
  hessp(x, v) -> array_like, shape (n,)
  ```

  computes the product H*v of the Hessian of fun at x with v.

- **res** – Method for computing residuals, i.e.

  ```
  res(x) -> array_like, shape(m,).
  ```

- **sres** – Method for computing residual sensitivities. If its is a callable, it should be of the form

`sres(x) -> array, shape (m,n).`

If its value is True, then res should return the residual sensitivities as a second output.

- **fun_accept_sensi_orders** – Flag indicating whether fun takes sensi_orders as an argument. Default: False.
- **res_accept_sensi_orders** – Flag indicating whether res takes sensi_orders as an argument. Default: False
- **x_names** – Parameter names. None if no names provided, otherwise a list of str, length dim_full (as in the Problem class). Can be read by the problem.

history
For storing the call history. Initialized by the methods, e.g. the optimizer, in *initialize_history()*.

pre_post_processor
Preprocess input values to and postprocess output values from __call__. Configured in *update_from_problem()*.

Notes

If fun_accept_sensi_orders resp. res_accept_sensi_orders is True, fun resp. res can also return dictionaries instead of tuples. In that case, they are expected to follow the naming conventions in `constants.py`. This is of interest, because when __call__ is called with return_dict = True, the full dictionary is returned, which can contain e.g. also simulation data or debugging information.

__call__()
Method to obtain arbitrary sensitivities. This is the central method which is always called, also by the get_* methods.

There are different ways in which an optimizer calls the objective function, and in how the objective function provides information (e.g. derivatives via separate functions or along with the function values). The different calling modes increase efficiency in space and time and make the objective flexible.

Parameters

- **x** – The parameters for which to evaluate the objective function.
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.
- **return_dict** – If False (default), the result is a Tuple of the requested values in the requested order. Tuples of length one are flattened. If True, instead a dict is returned which can carry further information.

Returns By default, this is a tuple of the requested function values and derivatives in the requested order (if only 1 value, the tuple is flattened). If *return_dict*, then instead a dict is returned with function values and derivatives indicated by ids.

Return type result

__class__
alias of `builtins.type`

__deepcopy__(*memodict=None*) → pypesto.objective.objective.Objective

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.objective.objective', '__doc__': '\n T

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*fun: Callable = None*, *grad: Union[Callable, bool] = None*, *hess: Callable = None*, *hessp: Callable = None*, *res: Callable = None*, *sres: Union[Callable, bool] = None*, *fun_accept_sensi_orders: bool = False*, *res_accept_sensi_orders: bool = False*, *x_names: List[str] = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.objective.objective'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

check_grad(*x: numpy.ndarray*, *x_indices: List[int] = None*, *eps: float = 1e-05*, *verbosity: int = 1*, *mode: str = 'mode_fun'*) → pandas.core.frame.DataFrame
Compare gradient evaluation: Firstly approximate via finite differences, and secondly use the objective gradient.

Parameters

- **x** – The parameters for which to evaluate the gradient.
- **x_indices** – List of index values for which to compute gradients. Default: all.
- **eps** – Finite differences step size. Default: 1e-5.
- **verbosity** – Level of verbosity for function output. * 0: no output, * 1: summary for all parameters, * 2: summary for individual parameters. Default: 1.
- **mode** – Residual (MODE_RES) or objective function value (MODE_FUN, default) computation mode.

Returns gradient, finite difference approximations and error estimates.

Return type result

check_sensi_orders(*sensi_orders*, *mode*) → None
Check if the objective is able to compute the requested sensitivities. If not, throw an exception.

Raises

- ValueError if the objective function cannot be called as
- `requested.`

get_fval(*x: numpy.ndarray*) → float
Get the function value at x.

get_grad(*x: numpy.ndarray*) → numpy.ndarray
Get the gradient at x.

get_hess(*x: numpy.ndarray*) → numpy.ndarray
Get the Hessian at x.

get_res(*x: numpy.ndarray*) → numpy.ndarray
Get the residuals at x.

get_sres(*x: numpy.ndarray*) → numpy.ndarray
Get the residual sensitivities at x.

has_fun

has_grad

has_hess

has_hessp

has_res

has_sres

initialize()
Initialize the objective function. This function is used at the beginning of an analysis, e.g. optimization, and can e.g. reset the objective memory. By default does nothing.

static output_to_dict()
Convert output tuple to dict.

static output_to_tuple()
Return values as requested by the caller, since usually only a subset is demanded. One output is returned as-is, more than one output are returned as a tuple in order (fval, grad, hess).

update_from_problem(*dim_full: int, x_free_indices: List[int], x_fixed_indices: List[int], x_fixed_vals: List[int]*)
Handle fixed parameters. Later, the objective will be given parameter vectors x of dimension dim, which have to be filled up with fixed parameter values to form a vector of dimension dim_full >= dim. This vector is then used to compute function value and derivatives. The derivatives must later be reduced again to dimension dim.

This is so as to make the fixing of parameters transparent to the caller.

The methods preprocess, postprocess are overwritten for the above functionality, respectively.

Parameters

- **dim_full** – Dimension of the full vector including fixed parameters.
- **x_free_indices** – Vector containing the indices (zero-based) of free parameters (complimentary to x_fixed_indices).
- **x_fixed_indices** – Vector containing the indices (zero-based) of parameter components that are not to be optimized.
- **x_fixed_vals** – Vector of the same length as x_fixed_indices, containing the values of the fixed parameters.

class pypesto.objective.**OptimizerHistory**(*history: pypesto.objective.history.History, x0: numpy.ndarray*)
Bases: pypesto.objective.history.HistoryBase

Objective call history. Also handles saving of intermediate results.

fval0, fval_min
Initial and best function value found.

x0, x_min
Initial and best parameters found.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.objective.history', '__doc__': '\n Obj

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*history: pypesto.objective.history.History*, *x0: numpy.ndarray*) → None
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.objective.history'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

finalize()
Finalize history. Called after a run.

get_chi2_trace() → Sequence[numpy.ndarray]
Chi2 value trace.

get_fval_trace() → Sequence[float]
Function value trace.

get_grad_trace() → Sequence[numpy.ndarray]
Gradient trace.

get_hess_trace() → Sequence[numpy.ndarray]
Hessian trace.

get_res_trace() → Sequence[numpy.ndarray]
Residual trace.

get_schi2_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Chi2 value sensitivity trace.

get_sres_trace() → Sequence[numpy.ndarray]
Residual sensitivity trace.

get_time_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Execution time trace.

get_x_trace() → Sequence[numpy.ndarray]
Parameter trace.

n_fval
Number of function evaluations.

n_grad
Number of gradient evaluations.

n_hess
Number of Hessian evaluations.

n_res
Number of residual evaluations.

n_sres
Number or residual sensitivity evaluations.

start_time
Start time.

update(*x: numpy.ndarray, sensi_orders: Tuple[int], mode: str, result: Dict[str, Union[float, numpy.ndarray]]*) → None
Update history and best found value.

pypesto.objective.**res_to_chi2**(*res: numpy.ndarray*)
We assume that the residuals res are related to an objective function value fval = chi2 via:

```
fval = 0.5 * sum(res**2)
```

which is the 'Linear' formulation in scipy.

pypesto.objective.**sres_to_schi2**(*res: numpy.ndarray*, *sres: numpy.ndarray*)
In line with the assumptions in res_to_chi2.

CHAPTER 6

Problem

A problem contains the objective as well as all information like prior describing the problem to be solved.

class pypesto.problem.**Iterable**
Bases: collections.abc.Iterable, typing.Generic

__abstractmethods__ = frozenset({'__iter__'})

__args__ = None

__class__
alias of GenericMeta

__delattr__
Implement delattr(self, name).

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__extra__
alias of collections.abc.Iterable

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__iter__()

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'typing'

__ne__
Return self!=value.

static __new__(*cls*, **args*, ***kwds*)
Create and return a new object. See help(type) for accurate signature.

__next_in_mro__
alias of builtins.object

__orig_bases__ = (typing.Generic[+T_co],)

__origin__ = None

__parameters__ = (+T_co,)

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__slots__ = ()

__str__
Return str(self).

__subclasshook__()

__tree_hash__ = -9223366141335292075

class pypesto.problem.**List**
Bases: list, typing.MutableSequence

__abstractmethods__ = frozenset()

__add__
Return self+value.

__args__ = None

__class__
alias of GenericMeta

__contains__
Return key in self.

__delattr__
Implement delattr(self, name).

__delitem__
Delete self[key].

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__extra__
alias of builtins.list

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__getitem__()
x.__getitem__(y) <==> x[y]

__gt__
Return self>value.

__hash__ = None

__iadd__
Implement self+=value.

__imul__
Implement self*=value.

__init__
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__iter__
Implement iter(self).

__le__
Return self<=value.

__len__
Return len(self).

__lt__
Return self<value.

__module__ = 'typing'

__mul__
Return self*value.

__ne__
Return self!=value.

static __new__(*cls*, **args*, ***kwds*)
Create and return a new object. See help(type) for accurate signature.

__next_in_mro__
alias of `builtins.object`

__orig_bases__ = (<class 'list'>, typing.MutableSequence[~T])

__origin__ = None

__parameters__ = (~T,)

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__reversed__()
L.__reversed__() – return a reverse iterator over the list

__rmul__
Return value*self.

__setattr__
Implement setattr(self, name, value).

__setitem__
Set self[key] to value.

__sizeof__()
L.__sizeof__() – size of L in memory, in bytes

__slots__ = ()

__str__
Return str(self).

__subclasshook__()

__tree_hash__ = -9223366141335289084

append(*object*) → None – append object to end

clear() → None – remove all items from L

copy() → list – a shallow copy of L

count(*value*) → integer – return number of occurrences of value

extend(*iterable*) → None – extend list by appending elements from the iterable

index(*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

insert()
L.insert(index, object) – insert object before index

pop([*index*]) → item – remove and return item at index (default last).
Raises IndexError if list is empty or index is out of range.

remove(*value*) → None – remove first occurrence of value.
Raises ValueError if the value is not present.

reverse()
L.reverse() – reverse *IN PLACE*

sort(*key=None*, *reverse=False*) → None – stable sort *IN PLACE*

class pypesto.problem.**Objective**(*fun: Callable = None*, *grad: Union[Callable, bool] = None*, *hess: Callable = None*, *hessp: Callable = None*, *res: Callable = None*, *sres: Union[Callable, bool] = None*, *fun_accept_sensi_orders: bool = False*, *res_accept_sensi_orders: bool = False*, *x_names: List[str] = None*)

Bases: object

The objective class is a simple wrapper around the objective function, giving a standardized way of calling. Apart from that, it manages several things including fixing of parameters and history.

The objective function is assumed to be in the format of a cost function, log-likelihood function, or log-posterior function. These functions are subject to minimization. For profiling and sampling, the sign is internally flipped, all returned and stored values are however given as returned by this objective function. If maximization is to be performed, the sign should be flipped before creating the objective function.

Parameters

- **fun** – The objective function to be minimized. If it only computes the objective function value, it should be of the form

  ```
  fun(x) -> float
  ```

  where x is an 1-D array with shape (n,), and n is the parameter space dimension.

- **grad** – Method for computing the gradient vector. If it is a callable, it should be of the form

  ```
  grad(x) -> array_like, shape (n,).
  ```

  If its value is True, then fun should return the gradient as a second output.

- **hess** – Method for computing the Hessian matrix. If it is a callable, it should be of the form

  ```
  hess(x) -> array, shape (n,n).
  ```

  If its value is True, then fun should return the gradient as a second, and the Hessian as a third output, and grad should be True as well.

- **hessp** – Method for computing the Hessian vector product, i.e.

  ```
  hessp(x, v) -> array_like, shape (n,)
  ```

  computes the product H*v of the Hessian of fun at x with v.

- **res** – Method for computing residuals, i.e.

  ```
  res(x) -> array_like, shape(m,).
  ```

- **sres** – Method for computing residual sensitivities. If its is a callable, it should be of the form

  ```
  sres(x) -> array, shape (m,n).
  ```

  If its value is True, then res should return the residual sensitivities as a second output.

- **fun_accept_sensi_orders** – Flag indicating whether fun takes sensi_orders as an argument. Default: False.
- **res_accept_sensi_orders** – Flag indicating whether res takes sensi_orders as an argument. Default: False
- **x_names** – Parameter names. None if no names provided, otherwise a list of str, length dim_full (as in the Problem class). Can be read by the problem.

history
For storing the call history. Initialized by the methods, e.g. the optimizer, in *initialize_history()*.

pre_post_processor
Preprocess input values to and postprocess output values from __call__. Configured in *update_from_problem()*.

Notes

If fun_accept_sensi_orders resp. res_accept_sensi_orders is True, fun resp. res can also return dictionaries instead of tuples. In that case, they are expected to follow the naming conventions in `constants.py`. This is of interest, because when __call__ is called with return_dict = True, the full dictionary is returned, which can contain e.g. also simulation data or debugging information.

__call__()
Method to obtain arbitrary sensitivities. This is the central method which is always called, also by the get_* methods.

There are different ways in which an optimizer calls the objective function, and in how the objective function provides information (e.g. derivatives via separate functions or along with the function values). The different calling modes increase efficiency in space and time and make the objective flexible.

Parameters

- **x** – The parameters for which to evaluate the objective function.
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.
- **return_dict** – If False (default), the result is a Tuple of the requested values in the requested order. Tuples of length one are flattened. If True, instead a dict is returned which can carry further information.

Returns By default, this is a tuple of the requested function values and derivatives in the requested order (if only 1 value, the tuple is flattened). If *return_dict*, then instead a dict is returned with function values and derivatives indicated by ids.

Return type result

__class__
alias of `builtins.type`

__deepcopy__(*memodict=None*) → pypesto.objective.objective.Objective

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.objective.objective', '__doc__': '\n T

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*fun: Callable = None*, *grad: Union[Callable, bool] = None*, *hess: Callable = None*, *hessp: Callable = None*, *res: Callable = None*, *sres: Union[Callable, bool] = None*, *fun_accept_sensi_orders: bool = False*, *res_accept_sensi_orders: bool = False*, *x_names: List[str] = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.objective.objective'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

check_grad(*x: numpy.ndarray*, *x_indices: List[int] = None*, *eps: float = 1e-05*, *verbosity: int = 1*, *mode: str = 'mode_fun'*) → pandas.core.frame.DataFrame
Compare gradient evaluation: Firstly approximate via finite differences, and secondly use the objective gradient.

Parameters

- **x** – The parameters for which to evaluate the gradient.
- **x_indices** – List of index values for which to compute gradients. Default: all.
- **eps** – Finite differences step size. Default: 1e-5.
- **verbosity** – Level of verbosity for function output. * 0: no output, * 1: summary for all parameters, * 2: summary for individual parameters. Default: 1.
- **mode** – Residual (MODE_RES) or objective function value (MODE_FUN, default) computation mode.

Returns gradient, finite difference approximations and error estimates.

Return type result

check_sensi_orders(*sensi_orders*, *mode*) → None
Check if the objective is able to compute the requested sensitivities. If not, throw an exception.

Raises

- ValueError if the objective function cannot be called as
- `requested.`

get_fval(*x: numpy.ndarray*) → float
Get the function value at x.

get_grad(*x: numpy.ndarray*) → numpy.ndarray
Get the gradient at x.

get_hess(*x: numpy.ndarray*) → numpy.ndarray
Get the Hessian at x.

get_res(*x: numpy.ndarray*) → numpy.ndarray
Get the residuals at x.

get_sres(*x: numpy.ndarray*) → numpy.ndarray
Get the residual sensitivities at x.

has_fun

has_grad

has_hess

has_hessp

has_res

has_sres

initialize()
Initialize the objective function. This function is used at the beginning of an analysis, e.g. optimization, and can e.g. reset the objective memory. By default does nothing.

static output_to_dict()
Convert output tuple to dict.

static output_to_tuple()
Return values as requested by the caller, since usually only a subset is demanded. One output is returned as-is, more than one output are returned as a tuple in order (fval, grad, hess).

update_from_problem(*dim_full: int, x_free_indices: List[int], x_fixed_indices: List[int], x_fixed_vals: List[int]*)
Handle fixed parameters. Later, the objective will be given parameter vectors x of dimension dim, which have to be filled up with fixed parameter values to form a vector of dimension dim_full >= dim. This vector is then used to compute function value and derivatives. The derivatives must later be reduced again to dimension dim.

This is so as to make the fixing of parameters transparent to the caller.

The methods preprocess, postprocess are overwritten for the above functionality, respectively.

Parameters

- **dim_full** – Dimension of the full vector including fixed parameters.
- **x_free_indices** – Vector containing the indices (zero-based) of free parameters (complimentary to x_fixed_indices).
- **x_fixed_indices** – Vector containing the indices (zero-based) of parameter components that are not to be optimized.
- **x_fixed_vals** – Vector of the same length as x_fixed_indices, containing the values of the fixed parameters.

class pypesto.problem.**Problem**(*objective: pypesto.objective.objective.Objective, lb: Union[numpy.ndarray, List[float]], ub: Union[numpy.ndarray, List[float]], dim_full: Optional[int] = None, x_fixed_indices: Optional[Iterable[int]] = None, x_fixed_vals: Optional[Iterable[float]] = None, x_guesses: Optional[Iterable[float]] = None, x_names: Optional[Iterable[str]] = None*)

Bases: object

The problem formulation. A problem specifies the objective function, boundaries and constraints, parameter guesses as well as the parameters which are to be optimized.

Parameters

- **objective** – The objective function for minimization. Note that a shallow copy is created.
- **ub** (*lb,*) – The lower and upper bounds. For unbounded directions set to inf.
- **dim_full** – The full dimension of the problem, including fixed parameters.
- **x_fixed_indices** – Vector containing the indices (zero-based) of parameter components that are not to be optimized.
- **x_fixed_vals** – Vector of the same length as x_fixed_indices, containing the values of the fixed parameters.
- **x_guesses** – Guesses for the parameter values, shape (g, dim), where g denotes the number of guesses. These are used as start points in the optimization.

- **x_names** – Parameter names that can be optionally used e.g. in visualizations. If objective.get_x_names() is not None, those values are used, else the values specified here are used if not None, otherwise the variable names are set to ['x0', ... 'x{dim_full}']. The list must always be of length dim_full.

dim
The number of non-fixed parameters. Computed from the other values.

x_free_indices
Vector containing the indices (zero-based) of free parameters (complimentary to x_fixed_indices).

Type array_like of int

Notes

On the fixing of parameter values:

The number of parameters dim_full the objective takes as input must be known, so it must be either lb a vector of that size, or dim_full specified as a parameter.

All vectors are mapped to the reduced space of dimension dim in __init__, regardless of whether they were in dimension dim or dim_full before. If the full representation is needed, the methods get_full_vector() and get_full_matrix() can be used.

__class__
alias of `builtins.type`

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.problem', '__doc__': "\n The problem f

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*objective: pypesto.objective.objective.Objective, lb: Union[numpy.ndarray, List[float]], ub: Union[numpy.ndarray, List[float]], dim_full: Optional[int] = None, x_fixed_indices: Optional[Iterable[int]] = None, x_fixed_vals: Optional[Iterable[float]] = None, x_guesses: Optional[Iterable[float]] = None, x_names: Optional[Iterable[str]] = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.problem'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

fix_parameters(*parameter_indices: Union[Iterable[int], int], parameter_vals: Union[Iterable[float], float]*) → None
Fix specified parameters to specified values

get_full_matrix(*x: Optional[numpy.ndarray]*) → Optional[numpy.ndarray]
Map matrix from dim to dim_full. Usually used for hessian.

Parameters **x** (*array_like, shape=(dim, dim)*) – The matrix in dimension dim.

get_full_vector(*x: Optional[numpy.ndarray], x_fixed_vals: Iterable[float] = None*) → Optional[numpy.ndarray]
Map vector from dim to dim_full. Usually used for x, grad.

Parameters

- **x** (*array_like, shape=(dim,)*) – The vector in dimension dim.
- **x_fixed_vals** (*array_like, ndim=1, optional*) – The values to be used for the fixed indices. If None, then nans are inserted. Usually, None will be used for grad and problem.x_fixed_vals for x.

get_reduced_matrix(*x_full: Optional[numpy.ndarray]*) → Optional[numpy.ndarray]
Map matrix from dim_full to dim, i.e. delete fixed indices.

Parameters **x_full** (*array_like, ndim=2*) – The matrix in dimension dim_full.

get_reduced_vector(*x_full: Optional[numpy.ndarray]*) → Optional[numpy.ndarray]
Map vector from dim_full to dim, i.e. delete fixed indices.

Parameters **x_full** (*array_like, ndim=1*) – The vector in dimension dim_full.

normalize_input(*check_x_guesses: bool = True*) → None
Reduce all vectors to dimension dim and have the objective accept vectors of dimension dim.

print_parameter_summary() → None
Prints a summary of what parameters are being optimized and what parameter boundaries are

unfix_parameters(*parameter_indices: Union[Iterable[int], int]*) → None
Free specified parameters

CHAPTER 7

PEtab

pyPESTO support for the PEtab data format.

class pypesto.petab.**PetabImporter**(*petab_problem: petab.Problem*, *output_folder: str = None*, *model_name: str = None*)

Bases: pypesto.objective.amici_objective.AmiciObjectBuilder

MODEL_BASE_DIR = 'amici_models'

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.petab.importer', 'MODEL_BASE_DIR': 'am

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__ (*petab_problem: petab.Problem*, *output_folder: str = None*, *model_name: str = None*)

petab_problem: Managing access to the model and data.

output_folder: Folder to contain the amici model. Defaults to './amici_models/{model_name}'.

model_name: Name of the model, which will in particular be the name of the compiled model python module.

__init_subclass__ ()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.petab.importer'

__ne__
Return self!=value.

__new__ ()
Create and return a new object. See help(type) for accurate signature.

__reduce__ ()
helper for pickle

__reduce_ex__ ()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__ () → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__ ()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

compile_model (***kwargs*)
Compile the model. If the output folder exists already, it is first deleted.

Parameters **kwargs** (Extra arguments passed to *amici.SbmlImporter.sbml2amici*.) –

create_edatas (*model: amici.Model = None*, *simulation_conditions=None*) → List[amici.ExpData]
Create list of amici.ExpData objects.

create_model(*force_compile: bool = False*, ***kwargs*) → amici.Model
Import amici model. If necessary or force_compile is True, compile first.

Parameters

- **force_compile** – If False, the model is compiled only if the output folder does not exist yet. If True, the output folder is deleted and the model (re-)compiled in either case.

  Warning: If *force_compile*, then an existing folder of that name will be deleted.

- **kwargs** (*Extra arguments passed to amici.SbmlImporter.sbml2amici*) –

create_objective(*model: amici.Model = None*, *solver: amici.Solver = None*, *edatas: Sequence[amici.ExpData] = None*, *force_compile: bool = False*, ***kwargs*) → pypesto.objective.amici_objective.AmiciObjective
Create a `pypesto.AmiciObjective`.

Parameters

- **model** – The AMICI model.
- **solver** – The AMICI solver.
- **edatas** – The experimental data in AMICI format.
- **force_compile** – Whether to force-compile the model if not passed.
- ****kwargs** – Additional arguments passed on to the objective.

Returns A `pypesto.AmiciObjective` for the model and the data.

Return type objective

create_problem(*objective: pypesto.objective.amici_objective.AmiciObjective = None*, ***kwargs*) → pypesto.problem.Problem
Create a `pypesto.Problem`.

Parameters

- **objective** – Objective as created by *create_objective*.
- ****kwargs** – Additional key word arguments passed on to the objective, if not provided.

Returns A `pypesto.Problem` for the objective.

Return type problem

create_solver(*model: amici.Model = None*) → amici.Solver
Return model solver.

static from_yaml(*yaml_config: Union[dict, str], output_folder: str = None, model_name: str = None*) → pypesto.petab.importer.PetabImporter
Simplified constructor using a petab yaml file.

rdatas_to_measurement_df(*rdatas: Sequence[amici.ReturnData], model: amici.Model = None*) → pandas.core.frame.DataFrame
Create a measurement dataframe in the petab format from the passed *rdatas* and own information.

Parameters

- **rdatas** – A list of rdatas as produced by pypesto.AmiciObjective.__call__(x, return_dict=True)['rdatas'].
- **model** – The amici model.

Returns A dataframe built from the rdatas in the format as in self.petab_problem.measurement_df.

Return type measurement_df

rdatas_to_simulation_df(*rdatas: Sequence[amici.ReturnData], model: amici.Model = None*) → pandas.core.frame.DataFrame

Same as *rdatas_to_measurement_df*, execpt a petab simulation dataframe is created, i.e. the measurement column label is adjusted.

CHAPTER 8

Optimize

Multistart optimization with support for various optimizers.

class pypesto.optimize.**DlibOptimizer**(*method: str*, *options: Dict = None*)

Bases: pypesto.optimize.optimizer.Optimizer

Use the Dlib toolbox for optimization.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.optimize.optimizer', '__doc__': '\n Us

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*method: str*, *options: Dict = None*)
Default constructor.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.optimize.optimizer'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

static get_default_options(*self*)
Create default options specific for the optimizer.

is_least_squares()

minimize(*problem*, *x0*, *id*, *history_options=None*)

class pypesto.optimize.**OptimizeOptions**(*startpoint_resample: bool = False*, *allow_failed_starts: bool = True*)

Bases: dict

Options for the multistart optimization.

Parameters

- **startpoint_resample** – Flag indicating whether initial points are supposed to be resampled if function evaluation fails at the initial point
- **allow_failed_starts** (*bool, optional*) – Flag indicating whether we tolerate that exceptions are thrown during the minimization process.

__class__
alias of builtins.type

__contains__()
True if D has a key k, else False.

__delattr__
Delete self[key].

__delitem__
Delete self[key].

__dict__ = mappingproxy({'__module__': 'pypesto.optimize.options', '__doc__': '\n Opti

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattr__(*key*)

__getattribute__
Return getattr(self, name).

__getitem__()
x.__getitem__(y) <==> x[y]

__gt__
Return self>value.

__hash__ = None

__init__(*startpoint_resample: bool = False*, *allow_failed_starts: bool = True*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__iter__
Implement iter(self).

__le__
Return self<=value.

__len__
Return len(self).

__lt__
Return self<value.

__module__ = 'pypesto.optimize.options'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Set self[key] to value.

__setitem__
Set self[key] to value.

__sizeof__() → size of D in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

static assert_instance(*maybe_options: Union[OptimizeOptions, Dict]*) → pypesto.optimize.options.OptimizeOptions
Returns a valid options object.

Parameters **maybe_options** (*OptimizeOptions or dict*) –

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys()
Returns a new dict with keys from iterable and values equal to value.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update(*[E]*, ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class pypesto.optimize.**Optimizer**
Bases: abc.ABC

This is the optimizer base class, not functional on its own.

An optimizer takes a problem, and possibly a start point, and then performs an optimization. It returns an OptimizerResult.

__abstractmethods__ = frozenset({'minimize', 'is_least_squares'})

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.optimize.optimizer', '__doc__': '\n Th

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__()
Default constructor.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.optimize.optimizer'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

static get_default_options()
Create default options specific for the optimizer.

is_least_squares()

minimize(*problem*, *x0*, *id*, *history_options=None*)

class pypesto.optimize.**OptimizerResult**(*id: str = None*, *x: numpy.ndarray = None*, *fval: float = None*, *grad: numpy.ndarray = None*, *hess: numpy.ndarray = None*, *res: numpy.ndarray = None*, *sres: numpy.ndarray = None*, *n_fval: int = None*, *n_grad: int = None*, *n_hess: int = None*, *n_res: int = None*, *n_sres: int = None*, *x0: numpy.ndarray = None*, *fval0: float = None*, *history: pypesto.objective.history.History = None*, *exitflag: int = None*, *time: float = None*, *message: str = None*)

Bases: dict

The result of an optimizer run. Used as a standardized return value to map from the individual result objects returned by the employed optimizers to the format understood by pypesto.

Can be used like a dict.

id
Id of the optimizer run. Usually the start index.

x
The best found parameters.

fval
The best found function value, *fun(x)*.

grad
The gradient at *x*.

hess
The Hessian at *x*.

res
The residuals at *x*.

sres
The residual sensitivities at *x*.

n_fval
Number of function evaluations.

n_grad
Number of gradient evaluations.

n_hess
Number of Hessian evaluations.

n_res
Number of residuals evaluations.

n_sres
Number of residual sensitivity evaluations.

x0
The starting parameters.

fval0
The starting function value, *fun(x0)*.

history
Objective history.

exitflag
The exitflag of the optimizer.

time
Execution time.

message
Textual comment on the optimization result.

Type str

Notes

Any field not supported by the optimizer is filled with None.

__class__
alias of `builtins.type`

__contains__()
True if D has a key k, else False.

__delattr__
Delete self[key].

__delitem__
Delete self[key].

__dict__ = mappingproxy({'__module__': 'pypesto.optimize.result', '__doc__': '\n The r

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattr__(*key*)

__getattribute__
Return getattr(self, name).

__getitem__()
x.__getitem__(y) <==> x[y]

__gt__
Return self>value.

__hash__ = None

__init__(*id: str = None*, *x: numpy.ndarray = None*, *fval: float = None*, *grad: numpy.ndarray = None*, *hess: numpy.ndarray = None*, *res: numpy.ndarray = None*, *sres: numpy.ndarray = None*, *n_fval: int = None*, *n_grad: int = None*, *n_hess: int = None*, *n_res: int = None*, *n_sres: int = None*, *x0: numpy.ndarray = None*, *fval0: float = None*, *history: pypesto.objective.history.History = None*, *exitflag: int = None*, *time: float = None*, *message: str = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__iter__
Implement iter(self).

__le__
Return self<=value.

__len__
Return len(self).

__lt__
Return self<value.

__module__ = 'pypesto.optimize.result'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Set self[key] to value.

__setitem__
Set self[key] to value.

__sizeof__() → size of D in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys()
Returns a new dict with keys from iterable and values equal to value.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class pypesto.optimize.**PyswarmOptimizer**(*options: Dict = None*)
Bases: pypesto.optimize.optimizer.Optimizer

Global optimization using pyswarm.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.optimize.optimizer', '__doc__': '\n Gl

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*options: Dict = None*)
Default constructor.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.optimize.optimizer'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

static get_default_options()
Create default options specific for the optimizer.

is_least_squares()

minimize(*problem*, *x0*, *id*, *history_options=None*)

class pypesto.optimize.**ScipyOptimizer**(*method: str = 'L-BFGS-B'*, *tol: float = 1e-09*, *options: Dict = None*)
Bases: pypesto.optimize.optimizer.Optimizer

Use the SciPy optimizers.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.optimize.optimizer', '__doc__': '\n Us

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*method: str = 'L-BFGS-B'*, *tol: float = 1e-09*, *options: Dict = None*)
Default constructor.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.optimize.optimizer'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

static get_default_options(*self*)
Create default options specific for the optimizer.

is_least_squares()

minimize(*problem*, *x0*, *id*, *history_options=None*)

pypesto.optimize.**minimize**(*problem: pypesto.problem.Problem*, *optimizer: pypesto.optimize.optimizer.Optimizer = None*, *n_starts: int = 100*, *ids: Iterable[str] = None*, *startpoint_method: Union[Callable, bool] = None*, *result: pypesto.result.Result = None*, *engine: pypesto.engine.base.Engine = None*, *options: pypesto.optimize.options.OptimizeOptions = None*, *history_options: pypesto.objective.history.HistoryOptions = None*) → pypesto.result.Result

This is the main function to call to do multistart optimization.

Parameters

- **problem** – The problem to be solved.
- **optimizer** – The optimizer to be used n_starts times.
- **n_starts** – Number of starts of the optimizer.
- **ids** – Ids assigned to the startpoints.
- **startpoint_method** – Method for how to choose start points. False means the optimizer does not require start points, e.g. 'pso' method in 'GlobalOptimizer'

- **result** – A result object to append the optimization results to. For example, one might append more runs to a previous optimization. If None, a new object is created.
- **engine** – Parallelization engine. Defaults to sequential execution on a SingleCoreEngine.
- **options** – Various options applied to the multistart optimization.
- **history_options** – Optimizer history options.

Returns Result object containing the results of all multistarts in *result.optimize_result*.

Return type result

CHAPTER 9

Profile

class pypesto.profile.**ProfileOptions**(*default_step_size: float = 0.01*, *min_step_size: float = 0.001*, *max_step_size: float = 1.0*, *step_size_factor: float = 1.25*, *delta_ratio_max: float = 0.1*, *ratio_min: float = 0.145*, *reg_points: int = 10*, *reg_order: int = 4*, *magic_factor_obj_value: float = 0.5*)

Bases: dict

Options for optimization based profiling.

Parameters

- **default_step_size** – default step size of the profiling routine along the profile path (adaptive step lengths algorithms will only use this as a first guess and then refine the update)
- **min_step_size** – lower bound for the step size in adaptive methods
- **max_step_size** – upper bound for the step size in adaptive methods
- **step_size_factor** – Adaptive methods recompute the likelihood at the predicted point and try to find a good step length by a sort of line search algorithm. This factor controls step handling in this line search
- **delta_ratio_max** – maximum allowed drop of the posterior ratio between two profile steps
- **ratio_min** – lower bound for likelihood ratio of the profile, based on inverse chi2-distribution. The default corresponds to 95% confidence
- **reg_points** – number of profile points used for regression in regression based adaptive profile points proposal
- **reg_order** – maximum degree of regression polynomial used in regression based adaptive profile points proposal
- **magic_factor_obj_value** – There is this magic factor in the old profiling code which slows down profiling at small ratios (must be >= 0 and < 1)

__class__
alias of builtins.type

__contains__()
True if D has a key k, else False.

__delattr__
Delete self[key].

__delitem__
Delete self[key].

__dict__ = mappingproxy({'__module__': 'pypesto.profile.profile', '__doc__': '\n Optio

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattr__(*key*)

__getattribute__
Return getattr(self, name).

__getitem__()
x.__getitem__(y) <==> x[y]

__gt__
Return self>value.

__hash__ = None

__init__(*default_step_size: float = 0.01*, *min_step_size: float = 0.001*, *max_step_size: float = 1.0*, *step_size_factor: float = 1.25*, *delta_ratio_max: float = 0.1*, *ratio_min: float = 0.145*, *reg_points: int = 10*, *reg_order: int = 4*, *magic_factor_obj_value: float = 0.5*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__iter__
Implement iter(self).

__le__
Return self<=value.

__len__
Return len(self).

__lt__
Return self<value.

__module__ = 'pypesto.profile.profile'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Set self[key] to value.

__setitem__
Set self[key] to value.

__sizeof__() → size of D in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

clear() → None. Remove all items from D.

copy() → a shallow copy of D

static create_instance(*maybe_options: Union[ProfileOptions, Dict]*) → pypesto.profile.profile.ProfileOptions
Returns a valid options object.

Parameters **maybe_options** (*ProfileOptions or dict*) –

fromkeys()
Returns a new dict with keys from iterable and values equal to value.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class pypesto.profile.**ProfilerResult**(*x_path*, *fval_path*, *ratio_path*, *gradnorm_path=None*, *exitflag_path=None*, *time_path=None*, *time_total=0.0*, *n_fval=0*, *n_grad=0*, *n_hess=0*, *message=None*)

Bases: dict

The result of a profiler run. The standardized return return value from pypesto.profile, which can either be initialized from an OptimizerResult or from an existing ProfilerResult (in order to extend the computation).

Can be used like a dict.

x_path
The path of the best found parameters along the profile (Dimension: n_par x n_profile_points)

Type ndarray

fval_path
The function values, fun(x), along the profile.

Type ndarray

ratio_path
The ratio of the posterior function along the profile.

Type ndarray

gradnorm_path
The gradient norm along the profile.

Type ndarray

exitflag_path
The exitflags of the optimizer along the profile.

Type ndarray

time_path
The computation time of the optimizer runs along the profile.

Type ndarray

time_total
The total computation time for the profile.

Type ndarray

n_fval
Number of function evaluations.

Type int

n_grad
Number of gradient evaluations.

Type int

n_hess
Number of Hessian evaluations.

Type int

message
Textual comment on the profile result.

Type str

Notes

Any field not supported by the profiler or the profiling optimizer is filled with None. Some fields are filled by pypesto itself.

__class__
alias of `builtins.type`

__contains__()
True if D has a key k, else False.

__delattr__
Delete self[key].

__delitem__
Delete self[key].

__dict__ = mappingproxy({'__module__': 'pypesto.profile.result', '__doc__': '\n The re

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattr__(*key*)

__getattribute__
Return getattr(self, name).

__getitem__()
x.__getitem__(y) <==> x[y]

__gt__
Return self>value.

__hash__ = None

__init__(*x_path*, *fval_path*, *ratio_path*, *gradnorm_path=None*, *exitflag_path=None*, *time_path=None*, *time_total=0.0*, *n_fval=0*, *n_grad=0*, *n_hess=0*, *message=None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__iter__
Implement iter(self).

__le__
Return self<=value.

__len__
Return len(self).

__lt__
Return self<value.

__module__ = 'pypesto.profile.result'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Set self[key] to value.

__setitem__
Set self[key] to value.

__sizeof__() → size of D in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

append_profile_point(*x*, *fval*, *ratio*, *gradnorm=nan*, *exitflag=nan*, *time=nan*, *n_fval=0*, *n_grad=0*, *n_hess=0*)
This function appends a new OptimizerResult to an existing ProfilerResults

clear() → None. Remove all items from D.

copy() → a shallow copy of D

flip_profile()
This function flips the profiling direction (left-right) Profiling direction needs to be changed once (if the profile is new) and twice, if we append to an existing profile

fromkeys()
Returns a new dict with keys from iterable and values equal to value.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update(*[E]*, ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

pypesto.profile.**parameter_profile**(*problem: pypesto.problem.Problem*, *result: pypesto.result.Result*, *optimizer: pypesto.optimize.optimizer.Optimizer*, *profile_index: numpy.ndarray = None*, *profile_list: int = None*, *result_index: int = 0*, *next_guess_method: Callable = None*, *profile_options: pypesto.profile.profile.ProfileOptions = None*) → pypesto.result.Result

This is the main function to call to do parameter profiling.

Parameters

- **problem** – The problem to be solved.
- **result** – A result object to initialize profiling and to append the profiling results to. For example, one might append more profiling runs to a previous profile, in order to merge these. The existence of an optimization result is obligatory.
- **optimizer** – The optimizer to be used along each profile.
- **profile_index** – array with parameter indices, whether a profile should be computed (1) or not (0) Default is all profiles should be computed
- **profile_list** – integer which specifies whether a call to the profiler should create a new list of profiles (default) or should be added to a specific profile list
- **result_index** – index from which optimization result profiling should be started (default: global optimum, i.e., index = 0)
- **next_guess_method** – function handle to a method that creates the next starting point for optimization in profiling.
- **profile_options** – Various options applied to the profile optimization.

Returns The profile results are filled into *result.profile_result*.

Return type result

CHAPTER 10

Sampling

Draw samples from the distribution, with support for various samplers.

class pypesto.sampling.**AdaptiveMetropolisSampler**(*options: Dict = None*)

Bases: pypesto.sampling.metropolis.MetropolisSampler

Metropolis-Hastings sampler with adaptive proposal covariance.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.sampling.adaptive_metropolis', '__doc__

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*options: Dict = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.sampling.adaptive_metropolis'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

classmethod default_options()
Convenience method to set/get default options.

Returns Default sampler options.

Return type default_options

get_last_sample() → pypesto.sampling.sampler.InternalSample
Get the last sample in the chain.

Returns The last sample in the chain in the exchange format.

Return type internal_sample

get_samples() → pypesto.sampling.result.McmcPtResult
Get the generated samples.

initialize(*problem: pypesto.problem.Problem*, *x0: numpy.ndarray*)
Initialize the sampler.

Parameters

- **problem** – The problem for which to sample.
- **x0** – Should, but is not required to, be used as initial parameter.

sample(*n_samples: int*, *beta: float = 1.0*)
Perform sampling.

Parameters

- **n_samples** – Number of samples to generate.
- **beta** – Inverse of the temperature to which the system is elevated.

set_last_sample(*sample: pypesto.sampling.sampler.InternalSample*)
Set the last sample in the chain to the passed value.

Parameters **sample** – The sample that will replace the last sample in the chain.

classmethod translate_options(*options*)
Convenience method to translate options and fill in defaults.

Parameters **options** – Options configuring the sampler.

class pypesto.sampling.**AdaptiveParallelTemperingSampler**(*internal_sampler: pypesto.sampling.sampler.InternalSampler*, *betas: Sequence[float] = None*, *n_chains: int = None*, *options: Dict = None*)
Bases: pypesto.sampling.parallel_tempering.ParallelTemperingSampler

Parallel tempering sampler with adaptive temperature adaptation.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.sampling.adaptive_parallel_tempering',

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*internal_sampler: pypesto.sampling.sampler.InternalSampler*, *betas: Sequence[float] = None*, *n_chains: int = None*, *options: Dict = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.sampling.adaptive_parallel_tempering'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

adjust_betas(*i_sample: int, swapped: Sequence[bool]*)
Update temperatures as in Vousden2016.

classmethod default_options() → Dict
Convenience method to set/get default options.

Returns Default sampler options.

Return type default_options

get_samples() → pypesto.sampling.result.McmcPtResult
Concatenate all chains.

initialize(*problem: pypesto.problem.Problem, x0: Union[numpy.ndarray, List[numpy.ndarray]]*)
Initialize the sampler.

Parameters

- **problem** – The problem for which to sample.
- **x0** – Should, but is not required to, be used as initial parameter.

sample(*n_samples: int*, *beta: float = 1.0*)
Perform sampling.

Parameters

- **n_samples** – Number of samples to generate.
- **beta** – Inverse of the temperature to which the system is elevated.

swap_samples() → Sequence[bool]
Swap samples as in Vousden2016.

classmethod translate_options(*options*)
Convenience method to translate options and fill in defaults.

Parameters **options** – Options configuring the sampler.

class pypesto.sampling.**InternalSampler**(*options: Dict = None*)
Bases: pypesto.sampling.sampler.Sampler

Sampler to be used inside a parallel tempering sampler.

The last sample can be obtained via *get_last_sample* and set via *set_last_sample*.

__abstractmethods__ = frozenset({'get_last_sample', 'initialize', 'sample', 'get_sampl

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.sampling.sampler', '__doc__': 'Sampler

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*options: Dict = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.sampling.sampler'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

classmethod default_options() → Dict
Convenience method to set/get default options.

Returns Default sampler options.

Return type default_options

get_last_sample() → pypesto.sampling.sampler.InternalSample
Get the last sample in the chain.

Returns The last sample in the chain in the exchange format.

Return type internal_sample

get_samples() → pypesto.sampling.result.McmcPtResult
Get the generated samples.

initialize(*problem: pypesto.problem.Problem, x0: Union[numpy.ndarray, List[numpy.ndarray]]*)
Initialize the sampler.

Parameters

- **problem** – The problem for which to sample.
- **x0** – Should, but is not required to, be used as initial parameter.

sample(*n_samples: int*, *beta: float = 1.0*)
Perform sampling.

Parameters

- **n_samples** – Number of samples to generate.
- **beta** – Inverse of the temperature to which the system is elevated.

set_last_sample(*sample: pypesto.sampling.sampler.InternalSample*)
Set the last sample in the chain to the passed value.

Parameters **sample** – The sample that will replace the last sample in the chain.

classmethod translate_options(*options*)
Convenience method to translate options and fill in defaults.

Parameters **options** – Options configuring the sampler.

class pypesto.sampling.**McmcPtResult**(*trace_x: numpy.ndarray, trace_fval: numpy.ndarray, betas: Iterable[float], message: str = None*)

Bases: dict

The result of a sampler run using Markov-chain Monte Carlo, and optionally parallel tempering.

Can be used like a dict.

Parameters

- **trace_x** (*[n_chain, n_iter, n_par]*) – Parameters
- **trace_fval** (*[n_chain, n_iter]*) – Function values.
- **betas** (*[n_chain]*) – The associated inverse temperatures.
- **message** (*str*) – Textual comment on the profile result.
- **n_chain denotes the number of chains, n_iter the number of** (*Here,*) –
- **(i.e., the chain length), and n_par the number of parameters.** (*iterations*) –

__class__
alias of builtins.type

__contains__()
True if D has a key k, else False.

__delattr__
Delete self[key].

__delitem__
Delete self[key].

__dict__ = mappingproxy({'__module__': 'pypesto.sampling.result', '__doc__': 'The resu

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattr__(*key*)

__getattribute__
Return getattr(self, name).

__getitem__()
x.__getitem__(y) <==> x[y]

__gt__
Return self>value.

__hash__ = None

__init__(*trace_x: numpy.ndarray, trace_fval: numpy.ndarray, betas: Iterable[float], message: str = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__iter__
Implement iter(self).

__le__
Return self<=value.

__len__
Return len(self).

__lt__
Return self<value.

__module__ = 'pypesto.sampling.result'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Set self[key] to value.

__setitem__
Set self[key] to value.

__sizeof__() → size of D in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys()
Returns a new dict with keys from iterable and values equal to value.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class pypesto.sampling.**MetropolisSampler**(*options: Dict = None*)
Bases: pypesto.sampling.sampler.InternalSampler

Simple Metropolis-Hastings sampler with fixed proposal variance.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.sampling.metropolis', '__doc__': '\n S

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*options: Dict = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.sampling.metropolis'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

classmethod default_options()
Convenience method to set/get default options.

Returns Default sampler options.

Return type default_options

get_last_sample() → pypesto.sampling.sampler.InternalSample
Get the last sample in the chain.

Returns The last sample in the chain in the exchange format.

Return type internal_sample

get_samples() → pypesto.sampling.result.McmcPtResult
Get the generated samples.

initialize(*problem: pypesto.problem.Problem*, *x0: numpy.ndarray*)
Initialize the sampler.

Parameters

- **problem** – The problem for which to sample.
- **x0** – Should, but is not required to, be used as initial parameter.

sample(*n_samples: int*, *beta: float = 1.0*)
Perform sampling.

Parameters

- **n_samples** – Number of samples to generate.
- **beta** – Inverse of the temperature to which the system is elevated.

set_last_sample(*sample: pypesto.sampling.sampler.InternalSample*)
Set the last sample in the chain to the passed value.

Parameters **sample** – The sample that will replace the last sample in the chain.

classmethod translate_options(*options*)
Convenience method to translate options and fill in defaults.

Parameters **options** – Options configuring the sampler.

class pypesto.sampling.**ParallelTemperingSampler**(*internal_sampler: pypesto.sampling.sampler.InternalSampler*, *betas: Sequence[float] = None*, *n_chains: int = None*, *options: Dict = None*)

Bases: pypesto.sampling.sampler.Sampler

Simple parallel tempering sampler.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.sampling.parallel_tempering', '__doc__

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*internal_sampler: pypesto.sampling.sampler.InternalSampler*, *betas: Sequence[float] = None*, *n_chains: int = None*, *options: Dict = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.sampling.parallel_tempering'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

adjust_betas(*i_sample: int, swapped: Sequence[bool]*)
Adjust temperature values. Default: Do nothing.

classmethod default_options() → Dict
Convenience method to set/get default options.

Returns Default sampler options.

Return type default_options

get_samples() → pypesto.sampling.result.McmcPtResult
Concatenate all chains.

initialize(*problem: pypesto.problem.Problem, x0: Union[numpy.ndarray, List[numpy.ndarray]]*)
Initialize the sampler.

Parameters

- **problem** – The problem for which to sample.
- **x0** – Should, but is not required to, be used as initial parameter.

sample(*n_samples: int*, *beta: float = 1.0*)
Perform sampling.

Parameters

- **n_samples** – Number of samples to generate.
- **beta** – Inverse of the temperature to which the system is elevated.

swap_samples() → Sequence[bool]
Swap samples as in Vousden2016.

classmethod translate_options(*options*)
Convenience method to translate options and fill in defaults.

Parameters **options** – Options configuring the sampler.

class pypesto.sampling.**Sampler**(*options: Dict = None*)
Bases: abc.ABC

Sampler base class, not functional on its own.

The sampler maintains an internal chain, which is initialized in *initialize*, and updated in *sample*.

__abstractmethods__ = frozenset({'sample', 'get_samples', 'initialize'})

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.sampling.sampler', '__doc__': 'Sampler

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*options: Dict = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.sampling.sampler'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

classmethod default_options() → Dict
Convenience method to set/get default options.

Returns Default sampler options.

Return type default_options

get_samples() → pypesto.sampling.result.McmcPtResult
Get the generated samples.

initialize(*problem: pypesto.problem.Problem, x0: Union[numpy.ndarray, List[numpy.ndarray]]*)
Initialize the sampler.

Parameters

- **problem** – The problem for which to sample.
- **x0** – Should, but is not required to, be used as initial parameter.

sample(*n_samples: int*, *beta: float = 1.0*)
Perform sampling.

Parameters

- **n_samples** – Number of samples to generate.
- **beta** – Inverse of the temperature to which the system is elevated.

classmethod translate_options(*options*)
Convenience method to translate options and fill in defaults.

Parameters **options** – Options configuring the sampler.

pypesto.sampling.**sample**(*problem: pypesto.problem.Problem*, *n_samples: int*, *sampler: pypesto.sampling.sampler.Sampler = None*, *x0: Union[numpy.ndarray, List[numpy.ndarray]] = None*, *result: pypesto.result.Result = None*) → pypesto.result.Result
This is the main function to call to do parameter sampling.

Parameters

- **problem** – The problem to be solved. If None is provided, a `pypesto.AdaptiveMetropolisSampler` is used.
- **n_samples** – Number of samples to generate.
- **sampler** – The sampler to perform the actual sampling.
- **x0** – Initial parameter for the Markov chain. If None, the best parameter found in optimization is used. Note that some samplers require an initial parameter, some may ignore it. x0 can also be a list, to have separate starting points for parallel tempering chains.
- **result** – A result to write to. If None provided, one is created from the problem.

Returns A result with filled in sample_options part.

Return type result

CHAPTER 11

Visualize

pypesto comes with various visualization routines. To use these, import pypesto.visualize.

class pypesto.visualize.**ReferencePoint**(*reference=None*, *x=None*, *fval=None*, *color=None*, *legend=None*)

Bases: dict

Reference point for plotting. Should contain a parameter value and an objective function value, may alos contain a color and a legend.

Can be used like a dict.

x
Reference parameters.

Type ndarray

fval
Function value, fun(x), for reference parameters.

Type float

color
Color which should be used for reference point.

Type RGBA, optional

auto_color
flag indicating whether color for this reference point should be assigned automatically or whether it was assigned by user

Type boolean

legend
legend text for reference point

Type str

__class__
alias of builtins.type

__contains__()
True if D has a key k, else False.

__delattr__
Delete self[key].

__delitem__
Delete self[key].

__dict__ = mappingproxy({'__module__': 'pypesto.visualize.reference_points', '__doc__'

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattr__(*key*)

__getattribute__
Return getattr(self, name).

__getitem__()
x.__getitem__(y) <==> x[y]

__gt__
Return self>value.

__hash__ = None

__init__(*reference=None*, *x=None*, *fval=None*, *color=None*, *legend=None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__iter__
Implement iter(self).

__le__
Return self<=value.

__len__
Return len(self).

__lt__
Return self<value.

__module__ = 'pypesto.visualize.reference_points'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Set self[key] to value.

__setitem__
Set self[key] to value.

__sizeof__() → size of D in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys()
Returns a new dict with keys from iterable and values equal to value.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

pypesto.visualize.**assign_clustered_colors**(*vals*, *balance_alpha=True*, *highlight_global=True*)

Cluster and assign colors.

Parameters

- **vals** (*numeric list or array*) – List to be clustered and assigned colors.

- **balance_alpha** (*bool (optional)*) – Flag indicating whether alpha for large clusters should be reduced to avoid overplotting (default: True)
- **highlight_global** (*bool (optional)*) – flag indicating whether global optimum should be highlighted

Returns **colors** – One for each element in 'vals'.

Return type list of RGBA

pypesto.visualize.**assign_clusters**(*vals*)
Find clustering.

Parameters **vals** (*numeric list or array*) – List to be clustered.

Returns

- **clust** (*numeric list*) – Indicating the corresponding cluster of each element from 'vals'.
- **clustsize** (*numeric list*) – Size of clusters, length equals number of clusters.

pypesto.visualize.**assign_colors**(*vals*, *colors=None*, *balance_alpha=True*, *highlight_global=True*)
Assign colors or format user specified colors.

Parameters

- **vals** (*numeric list or array*) – List to be clustered and assigned colors.
- **colors** (*list, or RGBA, optional*) – list of colors, or single color
- **balance_alpha** (*bool (optional)*) – Flag indicating whether alpha for large clusters should be reduced to avoid overplotting (default: True)
- **highlight_global** (*bool (optional)*) – flag indicating whether global optimum should be highlighted

Returns **colors** – One for each element in 'vals'.

Return type list of RGBA

pypesto.visualize.**create_references**(*references=None*, *x=None*, *fval=None*, *color=None*, *legend=None*) → List[pypesto.visualize.reference_points.ReferencePoint]
This function creates a list of reference point objects from user inputs

Parameters

- **references** (ReferencePoint *or dict or list, optional*) – Will be converted into a list of RefPoints
- **x** (*ndarray, optional*) – Parameter vector which should be used for reference point
- **fval** (*float, optional*) – Objective function value which should be used for reference point
- **color** (*RGBA, optional*) – Color which should be used for reference point.
- **legend** (*str*) – legend text for reference point

Returns **colors** – One for each element in 'vals'.

Return type list of RGBA

pypesto.visualize.**delete_nan_inf**(*fvals: numpy.ndarray*, *x: numpy.ndarray = None*) → Tuple[numpy.ndarray, numpy.ndarray]
Delete nan and inf values in fvals. If parameters 'x' are passend, also the corresponding entries are deleted.

Parameters

- **x** – array of parameters
- **fvals** – array of fval

Returns

- **x** (*np.array*) – array of parameters without nan or inf
- **fvals** (*np.array*) – array of fval without nan or inf

pypesto.visualize.**optimizer_history**(*results*, *ax=None*, *size=(18.5*, *10.5)*, *trace_x='steps'*, *trace_y='fval'*, *scale_y='log10'*, *offset_y=None*, *colors=None*, *y_limits=None*, *start_indices=None*, *reference=None*, *legends=None*)

Plot history of optimizer. Can plot either the history of the cost function or of the gradient norm, over either the optimizer steps or the computation time.

Parameters

- **results** (*pypesto.Result or list*) – Optimization result obtained by 'optimize.py' or list of those
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **size** (*tuple, optional*) – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **trace_x** (*str, optional*) – What should be plotted on the x-axis? Possibilities: 'time', 'steps' Default: 'steps'
- **trace_y** (*str, optional*) – What should be plotted on the y-axis? Possibilities: 'fval', 'gradnorm', 'stepsize' Default: 'fval'
- **scale_y** (*str, optional*) – May be logarithmic or linear ('log10' or 'lin')
- **offset_y** (*float, optional*) – Offset for the y-axis-values, as these are plotted on a log10-scale Will be computed automatically if necessary
- **colors** (*list, or RGBA, optional*) – list of colors, or single color color or list of colors for plotting. If not set, clustering is done and colors are assigned automatically
- **y_limits** (*float or ndarray, optional*) – maximum value to be plotted on the y-axis, or y-limits
- **start_indices** (*list or int*) – list of integers specifying the multistart to be plotted or int specifying up to which start index should be plotted
- **reference** (*list, optional*) – List of reference points for optimization results, containing et least a function value fval
- **legends** (*list or str*) – Labels for line plots, one label per result object

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**optimizer_history_lowlevel**(*vals*, *scale_y='log10'*, *colors=None*, *ax=None*, *size=(18.5*, *10.5)*, *x_label='Optimizer steps'*, *y_label='Objective value'*, *legend_text=None*)

Plot optimizer history using list of numpy arrays.

Parameters

- **vals** (*list of numpy arrays*) – list of 2xn-arrays (x_values and y_values of the trace)
- **scale_y** (*str, optional*) – May be logarithmic or linear ('log10' or 'lin')
- **colors** (*list, or RGBA, optional*) – list of colors, or single color color or list of colors for plotting. If not set, clustering is done and colors are assigned automatically
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **size** (*tuple, optional*) – see waterfall
- **x_label** (*str*) – label for x-axis
- **y_label** (*str*) – label for y-axis
- **legend_text** (*str*) – Label for line plots

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**parameters**(*results*, *ax=None*, *free_indices_only=True*, *lb=None*, *ub=None*, *size=None*, *reference=None*, *colors=None*, *legends=None*, *balance_alpha=True*, *start_indices=None*)

Plot parameter values.

Parameters

- **results** (*pypesto.Result or list*) – Optimization result obtained by 'optimize.py' or list of those
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **free_indices_only** (*bool, optional*) – If True, only free parameters are shown. If False, also the fixed parameters are shown.
- **ub** (*lb,*) – If not None, override result.problem.lb, problem.problem.ub. Dimension either result.problem.dim or result.problem.dim_full.
- **size** (*tuple, optional*) – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **reference** (*list, optional*) – List of reference points for optimization results, containing et least a function value fval
- **colors** (*list, or RGBA, optional*) – list of colors, or single color color or list of colors for plotting. If not set, clustering is done and colors are assigned automatically
- **legends** (*list or str*) – Labels for line plots, one label per result object
- **balance_alpha** (*bool (optional)*) – Flag indicating whether alpha for large clusters should be reduced to avoid overplotting (default: True)
- **start_indices** (*list or int*) – list of integers specifying the multistarts to be plotted or int specifying up to which start index should be plotted

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**parameters_lowlevel**(*xs*, *fvals*, *lb=None*, *ub=None*, *x_labels=None*, *ax=None*, *size=None*, *colors=None*, *linestyle='-'*, *legend_text=None*, *balance_alpha=True*)

Plot parameters plot using list of parameters.

Parameters

- **xs** (*nested list or array*) – Including optimized parameters for each startpoint. Shape: (n_starts, dim).
- **fvals** (*numeric list or array*) – Function values. Needed to assign cluster colors.
- **ub** (*lb,*) – The lower and upper bounds.
- **x_labels** (*array_like of str, optional*) – Labels to be used for the parameters.
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **size** (*tuple, optional*) – see parameters
- **colors** (*list of RGBA*) – One for each element in 'fvals'.
- **linestyle** (*str, optional*) – linestyle argument for parameter plot
- **legend_text** (*str*) – Label for line plots
- **balance_alpha** (*bool (optional)*) – Flag indicating whether alpha for large clusters should be reduced to avoid overplotting (default: True)

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**process_offset_y**(*offset_y: Optional[float], scale_y: str, min_val: float*) → float

compute offset for y-axis, depend on user settings

Parameters

- **offset_y** – value for offsetting the later plotted values, in order to ensure positivity if a semilog-plot is used
- **scale_y** – Can be 'lin' or 'log10', specifying whether values should be plotted on linear or on log10-scale
- **min_val** – Smallest value to be plotted

Returns **offset_y** – value for offsetting the later plotted values, in order to ensure positivity if a semilog-plot is used

Return type float

pypesto.visualize.**process_result_list**(*results*, *colors=None*, *legends=None*)

assigns colors and legends to a list of results, chekc user provided lists

Parameters

- **results** (*list or pypesto.Result*) – list of pypesto.Result objects or a single pypesto.Result
- **colors** (*list, optional*) – list of RGBA colors
- **legends** (*str or list*) – labels for line plots

Returns

- **results** (*list of pypesto.Result*) – list of pypesto.Result objects
- **colors** (*list of RGBA*) – One for each element in 'results'.
- **legends** (*list of str*) – labels for line plots

pypesto.visualize.**process_y_limits**(*ax*, *y_limits*)
apply user specified limits of y-axis

Parameters

- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **y_limits** (*ndarray*) – y_limits, minimum and maximum, for current axes object
- **min_val** (*float*) – Smallest value to be plotted

Returns **ax** – Axes object to use.

Return type matplotlib.Axes, optional

pypesto.visualize.**profile_lowlevel**(*fvals*, *ax=None*, *size=(18.5, 6.5)*, *color=None*, *legend_text=None*)
Lowlevel routine for plotting one profile, working with a numpy array only

Parameters

- **fvals** (*numeric list or array*) – Including values need to be plotted.
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **size** (*tuple, optional*) – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **color** (*RGBA, optional*) – color for profiles in plot.
- **legend_text** (*str*) – Label for line plots

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**profiles**(*results*, *ax=None*, *profile_indices=None*, *size=(18.5, 6.5)*, *reference=None*, *colors=None*, *legends=None*, *profile_list_id=0*)
Plot classical 1D profile plot (using the posterior, e.g. Gaussian like profile)

Parameters

- **results** (*list or pypesto.Result*) – list of pypesto.Result or single pypesto.Result
- **ax** (*list of matplotlib.Axes, optional*) – List of axes objects to use.
- **profile_indices** (*list of integer values*) – list of integer values specifying which profiles should be plotted
- **size** (*tuple, optional*) – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **reference** (*list, optional*) – List of reference points for optimization results, containing et least a function value fval
- **colors** (*list, or RGBA, optional*) – list of colors, or single color color or list of colors for plotting. If not set, clustering is done and colors are assigned automatically
- **legends** (*list or str, optional*) – Labels for line plots, one label per result object
- **profile_list_id** (*int, optional*) – index of the profile list to be used for profiling

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**profiles_lowlevel**(*fvals*, *ax=None*, *size=(18.5, 6.5)*, *color=None*, *legend_text=None*)

Lowlevel routine for profile plotting, working with a list of arrays only, opening different axes objects in case

Parameters

- **fvals** (*numeric list or array*) – Including values need to be plotted.
- **ax** (*list of matplotlib.Axes, optional*) – list of axes object to use.
- **size** (*tuple, optional*) – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **size** – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **color** (*RGBA, optional*) – color for profiles in plot.
- **legend_text** (*str*) – Label for line plots

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**sampling_1d_marginals**(*result: pypesto.result.Result*, *i_chain: int = 0*, *burn_in: int = None*, *stepsize: int = 1*, *plot_type: str = 'both'*, *bw: str = 'scott'*, *suptitle: str = None*, *size: Tuple[float, float] = None*)

Plot marginals.

Parameters

- **result** – The pyPESTO result object with filled sample result.
- **i_chain** – Which chain to plot. Default: First chain.
- **burn_in** – Index after burn-in phase, thus also the burn-in length.
- **stepsize** – Only one in *stepsize* values is plotted.
- **plot_type** (*{'hist'|'kde'|'both'}*) – Specify whether to plot a histogram ('hist'), a kernel density estimate ('kde'), or both ('both').
- **bw** (*{'scott', 'silverman' | scalar | pair of scalars}*) – Kernel bandwidth method.
- **suptitle** – Figure super title.
- **size** – Figure size in inches.

Returns **ax**

Return type matplotlib-axes

pypesto.visualize.**sampling_fval_trace**(*result: pypesto.result.Result*, *i_chain: int = 0*, *burn_in: int = None*, *stepsize: int = 1*, *title: str = None*, *size: Tuple[float, float] = None*, *ax: matplotlib.axes._axes.Axes = None*)

Plot log-posterior (=function value) over iterations.

Parameters

- **result** – The pyPESTO result object with filled sample result.
- **i_chain** – Which chain to plot. Default: First chain.
- **burn_in** – Index after burn-in phase, thus also the burn-in length.
- **stepsize** – Only one in *stepsize* values is plotted.

- **title** – Axes title.
- **size** (*ndarray*) – Figure size in inches.
- **ax** – Axes object to use.

Returns The plot axes.

Return type ax

pypesto.visualize.**sampling_parameters_trace**(*result: pypesto.result.Result*, *i_chain: int = 0*, *burn_in: int = None*, *stepsize: int = 1*, *use_problem_bounds: bool = True*, *suptitle: str = None*, *size: Tuple[float, float] = None*, *ax: matplotlib.axes._axes.Axes = None*)

Plot parameter values over iterations.

Parameters

- **result** – The pyPESTO result object with filled sample result.
- **i_chain** – Which chain to plot. Default: First chain.
- **burn_in** – Index after burn-in phase, thus also the burn-in length.
- **stepsize** – Only one in *stepsize* values is plotted.
- **use_problem_bounds** – Defines if the y-limits shall be the lower and upper bounds of parameter estimation problem.
- **suptitle** – Figure suptitle.
- **size** – Figure size in inches.
- **ax** – Axes object to use.

Returns The plot axes.

Return type ax

pypesto.visualize.**sampling_scatter**(*result: pypesto.result.Result*, *i_chain: int = 0*, *burn_in: int = None*, *stepsize: int = 1*, *suptitle: str = None*, *size: Tuple[float, float] = None*)

Parameter scatter plot.

Parameters

- **result** – The pyPESTO result object with filled sample result.
- **i_chain** – Which chain to plot. Default: First chain.
- **burn_in** – Index after burn-in phase, thus also the burn-in length.
- **stepsize** – Only one in *stepsize* values is plotted.
- **suptitle** – Figure super title.
- **size** – Figure size in inches.

Returns The plot axes.

Return type ax

pypesto.visualize.**waterfall**(*results*, *ax=None*, *size=(18.5, 10.5)*, *y_limits=None*, *scale_y='log10'*, *offset_y=None*, *start_indices=None*, *reference=None*, *colors=None*, *legends=None*)

Plot waterfall plot.

Parameters

- **results** (*pypesto.Result or list*) – Optimization result obtained by 'optimize.py' or list of those
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **size** (*tuple, optional*) – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **y_limits** (*float or ndarray, optional*) – maximum value to be plotted on the y-axis, or y-limits
- **scale_y** (*str, optional*) – May be logarithmic or linear ('log10' or 'lin')
- **offset_y** – offset for the y-axis, if it is supposed to be in log10-scale
- **start_indices** (*list or int*) – list of integers specifying the multistart to be plotted or int specifying up to which start index should be plotted
- **reference** (*list, optional*) – List of reference points for optimization results, containing et least a function value fval
- **colors** (*list, or RGBA, optional*) – list of colors, or single color color or list of colors for plotting. If not set, clustering is done and colors are assigned automatically
- **legends** (*list or str*) – Labels for line plots, one label per result object

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**waterfall_lowlevel**(*fvals*, *scale_y='log10'*, *offset_y=0.0*, *ax=None*, *size=(18.5*, *10.5)*, *colors=None*, *legend_text=None*)

Plot waterfall plot using list of function values.

Parameters

- **fvals** (*numeric list or array*) – Including values need to be plotted.
- **scale_y** (*str, optional*) – May be logarithmic or linear ('log10' or 'lin')
- **offset_y** – offset for the y-axis, if it is supposed to be in log10-scale
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **size** (*tuple, optional*) – see waterfall
- **colors** (*list, or RGBA, optional*) – list of colors, or single color color or list of colors for plotting. If not set, clustering is done and colors are assigned automatically
- **legend_text** (*str*) – Label for line plots

Returns **ax** – The plot axes.

Return type matplotlib.Axes

CHAPTER 12

Result

The pypesto.Result object contains all results generated by the pypesto components. It contains sub-results for optimization, profiles, sampling.

class pypesto.result.**OptimizeResult**

Bases: object

Result of the minimize() function.

__class__
alias of builtins.type

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.result', '__doc__': '\n Result of the

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__()
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.result'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

append(*optimizer_result: pypesto.optimize.result.OptimizerResult*)
Append an optimizer result to the result object.

Parameters **optimizer_result** – The result of one (local) optimizer run.

as_dataframe(*keys=None*) → pandas.core.frame.DataFrame
Get as pandas DataFrame. If keys is a list, return only the specified values.

as_list(*keys=None*) → Sequence
Get as list. If keys is a list, return only the specified values.

Parameters **keys** (*list(str), optional*) – Labels of the field to extract.

get_for_key(*key*) → list
Extract the list of values for the specified key as a list.

sort()
Sort the optimizer results by function value fval (ascending).

class pypesto.result.**ProfileResult**

Bases: object

Result of the profile() function.

__class__
alias of builtins.type

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.result', '__doc__': '\n Result of the p

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__()
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.result'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

add_profile(*profiler_result*, *i_parameter*)
Writes a profiler result to the result object at i_parameter.

Parameters

- **profiler_result** – The result of one (local) profiler run.
- **i_parameter** – integer specifying the parameter index

create_new_profile(*profiler_result: Optional[pypesto.profile.result.ProfilerResult] = None*)
Append an profiler result to the result object.

Parameters

- **profiler_result** – The result of one (local) profiler run or None, if to be left empty
- **profile_list** (*integer*) – index specifying the list of profiles, to which we want to append

create_new_profile_list()
Append an profiler result to the result object.

get_current_profile(*i_parameter*)
Append an profiler result to the result object.

Parameters **i_parameter** – integer specifying the profile index

class pypesto.result.**Result**(*problem=None*)
Bases: object

Universal result object for pypesto. The algorithms like optimize, profile, sample fill different parts of it.

problem
The problem underlying the results.

Type pypesto.Problem

optimize_result
The results of the optimizer runs.

profile_result
The results of the profiler run.

sample_result
The results of the sampler run.

__class__
alias of builtins.type

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.result', '__doc__': '\n Universal resu

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*problem=None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.result'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

class pypesto.result.**SampleResult**
Bases: object

Result of the sample() function.

__class__
alias of builtins.type

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.result', '__doc__': '\n Result of the

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__()
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.result'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

class pypesto.result.**Sequence**
Bases: collections.abc.Sequence, typing.Reversible, typing.Collection

__abstractmethods__ = frozenset({'__len__', '__getitem__'})

__args__ = None

__class__
alias of GenericMeta

__contains__(*value*)

__delattr__
Implement delattr(self, name).

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__extra__
alias of collections.abc.Sequence

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__getitem__(*index*)

__gt__
Return self>value.

__hash__
Return hash(self).

__init__
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__iter__()

__le__
Return self<=value.

__len__()

__lt__
Return self<value.

__module__ = 'typing'

__ne__
Return self!=value.

static __new__(*cls*, **args*, ***kwds*)
Create and return a new object. See help(type) for accurate signature.

__next_in_mro__
alias of `builtins.object`

__orig_bases__ = (typing.Reversible[+T_co], typing.Collection[+T_co])

__origin__ = None

__parameters__ = (+T_co,)

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__reversed__()

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__slots__ = ()

__str__
Return str(self).

__subclasshook__()

__tree_hash__ = -9223366141335289660

count(*value*) → integer – return number of occurrences of value

index(*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

CHAPTER 13

Engines

The execution of the multistarts can be parallelized in different ways, e.g. multi-threaded or cluster-based. Note that it is not checked whether a single task itself is internally parallelized.

class pypesto.engine.**Engine**

Bases: abc.ABC

Abstract engine base class.

__abstractmethods__ = frozenset({'execute'})

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.engine.base', '__doc__': '\n Abstract

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__()
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.engine.base'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

execute(*tasks: List[pypesto.engine.task.Task]*)
Execute tasks.

Parameters **tasks** – List of tasks to execute.

class pypesto.engine.**MultiProcessEngine**(*n_procs: int = None*)
Bases: pypesto.engine.base.Engine

Parallelize the task execution using multiprocessing.

Parameters **n_procs** – The maximum number of processes to use in parallel. Defaults to the number of CPUs available on the system according to *os.cpu_count()*. The effectively used number of processes will be the minimum of *n_procs* and the number of tasks submitted.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.engine.multi_process', '__doc__': '\n

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*n_procs: int = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.engine.multi_process'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

execute(*tasks: List[pypesto.engine.task.Task]*)
Pickle tasks and distribute work over parallel processes.

class pypesto.engine.**MultiThreadEngine**(*n_threads: int = None*)
Bases: pypesto.engine.base.Engine

Parallelize the task execution using multithreading.

Parameters **n_threads** – The maximum number of threads to use in parallel. Defaults to the number of CPUs available on the system according to *os.cpu_count()*. The effectively used number of threads will be the minimum of *n_threads* and the number of tasks submitted.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.engine.multi_thread', '__doc__': '\n Pa

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*n_threads: int = None*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.engine.multi_thread'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

execute(*tasks: List[pypesto.engine.task.Task]*)
Deepcopy tasks and distribute work over parallel threads.

class pypesto.engine.**OptimizerTask**(*optimizer: pypesto.optimize.optimizer.Optimizer*, *problem: pypesto.problem.Problem*, *x0: numpy.ndarray*, *id: str*, *options: pypesto.optimize.options.OptimizeOptions*, *history_options: pypesto.objective.history.HistoryOptions*, *handle_exception: Callable*)

Bases: pypesto.engine.task.Task

A multistart optimization task, performed in *pypesto.minimize*.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.engine.task', '__doc__': '\n A multista

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*optimizer: pypesto.optimize.optimizer.Optimizer*, *problem: pypesto.problem.Problem*, *x0: numpy.ndarray*, *id: str*, *options: pypesto.optimize.options.OptimizeOptions*, *history_options: pypesto.objective.history.HistoryOptions*, *handle_exception: Callable*)
Create the task object.

Parameters

- **optimizer** – The optimizer to use.
- **problem** – The problem to solve.
- **x0** – The point from which to start.
- **id** – The multistart id.
- **options** – Options object applying to optimization.
- **history_options** – Optimizer history options.
- **handle_exception** – Callable to apply when the optimization fails.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.engine.task'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

execute() → pypesto.optimize.result.OptimizerResult
Execute the task and return its results.

class pypesto.engine.**SingleCoreEngine**
Bases: pypesto.engine.base.Engine

Dummy engine for sequential execution on one core. Note that the objective itself may be multithreaded.

__abstractmethods__ = frozenset()

__class__
alias of abc.ABCMeta

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pypesto.engine.single_core', '__doc__': '\n Du

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__()
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pypesto.engine.single_core'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__str__
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

execute(*tasks: List[pypesto.engine.task.Task]*)
Execute all tasks in a simple for loop sequentially.

CHAPTER 14

Startpoint

Methods for selecting points that can be used as start points for multistart optimization. All methods have the form

```
method(**kwargs) -> startpoints
```

where the kwargs can/should include the following parameters, which are passed by pypesto:

n_starts: int Number of points to generate.

lb, ub: ndarray Lower and upper bound, may for most methods not contain nan or inf values.

x_guesses: ndarray, shape=(g, dim), optional Parameter guesses by the user, where g denotes the number of guesses. Note that these are only possibly taken as reference points to generate new start points (e.g. to maximize some distance) depending on the method, but regardless of g, there are always n_starts points generated and returned.

objective: pypesto.Objective, optional The objective can be used to evaluate the goodness of start points.

max_n_fval: int, optional The maximum number of evaluations of the objective function allowed.

pypesto.startpoint.**assign_startpoints**(*n_starts: int*, *startpoint_method: Callable*, *problem: pypesto.problem.Problem*, *options: pypesto.optimize.options.OptimizeOptions*) → numpy.ndarray

Assign startpoints.

pypesto.startpoint.**latin_hypercube**(***kwargs*) → numpy.ndarray

Generate latin hypercube points.

pypesto.startpoint.**uniform**(***kwargs*) → numpy.ndarray

Generate uniform points.

CHAPTER 15

Logging

Logging convenience functions.

pypesto.logging.**log_to_console**(*level=None*)
Log to console.

Parameters **level** (*int*) – The output level to use. Default: logging.DEBUG.

pypesto.logging.**log_to_file**(*level=None*, *filename=None*)
Log to file.

Parameters

- **level** (*int*) – The output level to use. Default: logging.DEBUG.
- **filename** (*str*) – The name of the file to append to. Default: .pypesto_logging.log.

CHAPTER 16

Release notes

16.1 0.0 series

16.1.1 0.0.13 (2020-05-03)

- Tidy up and speed up tests (#265 and others).
- Basic self-implemented Adaptive Metropolis and Adaptive Parallel Tempering sampling routines (#268).
- Fix namespace sample -> sampling (#275).
- Fix covariance matrix regularization (#275).
- Fix circular dependency *PetabImporter* - *PetabAmiciObjective* via *AmiciObjectBuilder*, *PetabAmiciObjective* becomes obsolete (#274).
- Define *AmiciCalculator* to separate the AMICI call logic (required for hierarchical optimization) (#277).
- Define initialize function for resetting steady states in *AmiciObjective* (#281).
- Fix scipy least squares options (#283).
- Allow failed starts by default (#280).
- Always copy parameter vector in objective to avoid side effects (#291).
- Add Dockerfile (#288).
- Fix header names in CSV history (#299).

Documentation:

- Use imported members in autodoc (#270).
- Enable python syntax highlighting in notebooks (#271).

16.1.2 0.0.12 (2020-04-06)

- Add typehints to global functions and classes.
- Add *PetabImporter.rdatas_to_simulation_df* function (all #235).
- Adapt y scale in waterfall plot if convergence was too good (#236).
- Clarify that *Objective* is of type negative log-posterior, for minimization (#243).
- Tidy up *AmiciObjective.parameter_mapping* as implemented in AMICI now (#247).
- Add *MultiThreadEngine* implementing multi-threading aside the *MultiProcessEngine* implementing multi-processing (#254).
- Fix copying and pickling of *AmiciObjective* (#252, #257).
- Remove circular dependence history-objective (#254).
- Fix problem of visualizing results with failed starts (#249).
- Rework history: make thread-safe, use factory methods, make context-specific (#256).
- Improve PEtab usage example (#258).
- Define history base contract, enabling different backends (#260).
- Store optimization results to HDF5 (#261).
- Simplify tests (#263).

Breaking changes:

- *HistoryOptions* passed to *pypesto.minimize* instead of *Objective* (#256).
- *GlobalOptimizer* renamed to *PyswarmOptimizer* (#235).

16.1.3 0.0.11 (2020-03-17)

- Rewrite AmiciObjective and PetabAmiciObjective simulation routine to directly use amici.petab_objective routines (#209, #219, #225).
- Implement petab test suite checks (#228).
- Various error fixes, in particular regarding PEtab and visualization.
- Improve trace structure.
- Fix conversion between fval and chi2, fix FIM (all #223).

16.1.4 0.0.10 (2019-12-04)

- Only compute FIM when sensitivities are available (#194).
- Fix documentation build (#197).
- Add support for pyswarm optimizer (#198).
- Run travis tests for documentation and notebooks only on pull requests (#199).

16.1.5 0.0.9 (2019-10-11)

- Update to AMICI 0.10.13, fix API changes (#185).
- Start using PEtab import from AMICI to be able to import constant species (#184, #185)
- Require PEtab>=0.0.0a16 (#183)

16.1.6 0.0.8 (2019-09-01)

- Add logo (#178).
- Fix petab API changes (#179).
- Some minor bugfixes (#168).

16.1.7 0.0.7 (2019-03-21)

- Support noise models in Petab and Amici.
- Minor Petab update bug fixes.

16.1.8 0.0.6 (2019-03-13)

- Several minor error fixes, in particular on tests and steady state.

16.1.9 0.0.5 (2019-03-11)

- Introduce AggregatedObjective to use multiple objectives at once.
- Estimate steady state in AmiciObjective.
- Check amici model build version in PetabImporter.
- Use Amici multithreading in AmiciObjective.
- Allow to sort multistarts by initial value.
- Show usage of visualization routines in notebooks.
- Various fixes, in particular to visualization.

16.1.10 0.0.4 (2019-02-25)

- Implement multi process parallelization engine for optimization.
- Introduce PrePostProcessor to more reliably handle pre- and post-processing.
- Fix problems with simulating for multiple conditions.
- Add more visualization routines and options for those (colors, reference points, plotting of lists of result obejcts)

16.1.11 0.0.3 (2019-01-30)

- Import amici models and the petab data format automatically using pypesto.PetabImporter.
- Basic profiling routines.

16.1.12 0.0.2 (2018-10-18)

- Fix parameter values
- Record trace of function values
- Amici objective to directly handle amici models

16.1.13 0.0.1 (2018-07-25)

- Basic framework and implementation of the optimization

CHAPTER 17

Authors

This package was mainly developed by:

- Jan Hasenauer
- Yannik Schälte
- Fabian Fröhlich
- Daniel Weindl
- Paul Stapor
- Leonard Schmiester
- Dantong Wang
- Leonard Schmiester
- Caro Loos

CHAPTER 18

Contact

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

- Yannik Schälte: yannik.schaelte@gmail.com

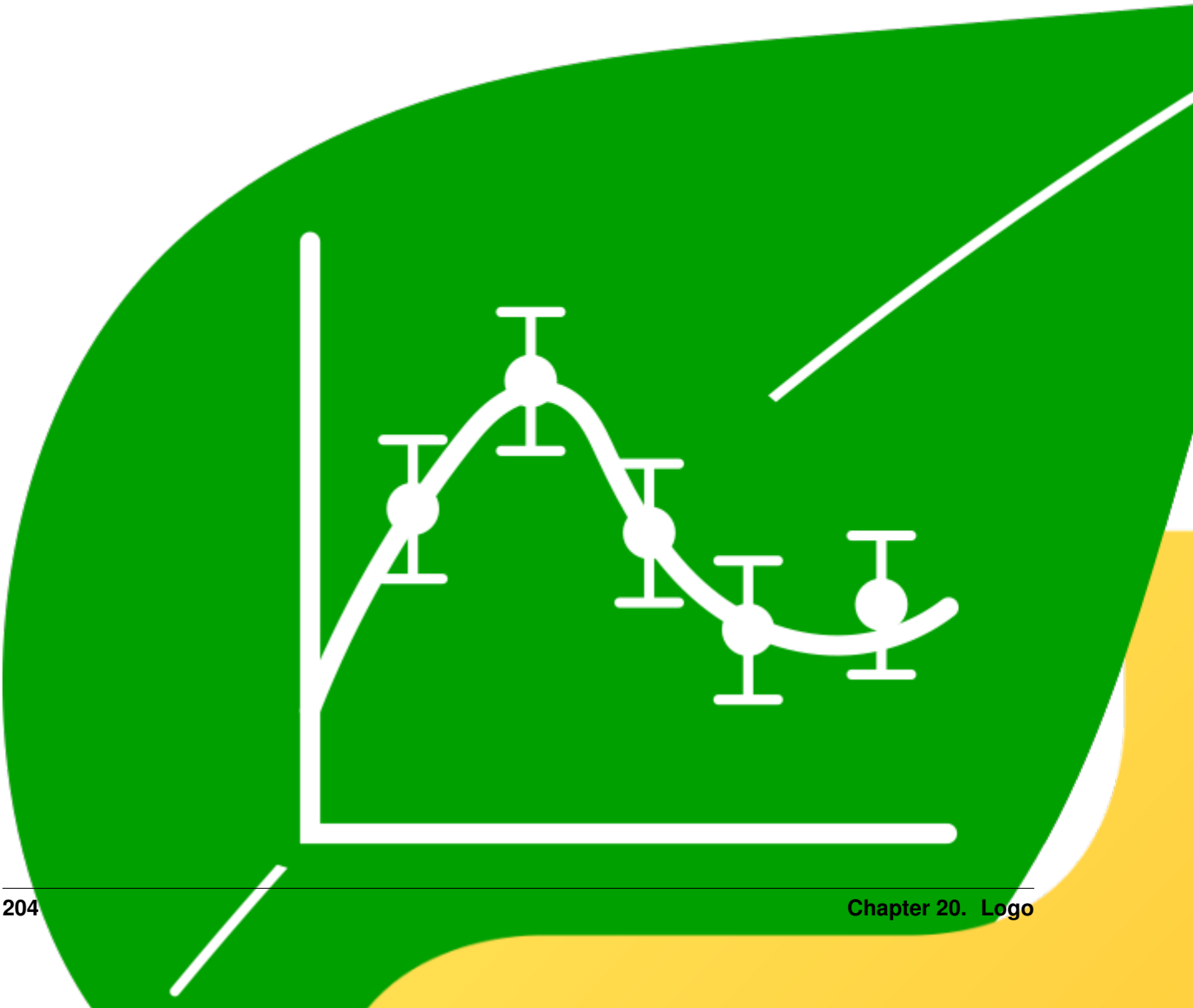
CHAPTER 19

License

```
Copyright (c) 2018, Jan Hasenauer
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are␣
↪permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list␣
↪of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this␣
↪list of conditions and the following disclaimer in the documentation and/or other␣
↪materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be␣
↪used to endorse or promote products derived from this software without specific␣
↪prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY␣
↪EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED␣
↪WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.␣
↪IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,␣
↪INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT␣
↪NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR␣
↪PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,␣
↪WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)␣
↪ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE␣
↪POSSIBILITY OF SUCH DAMAGE.
```

CHAPTER 20

Logo



pyPESTO's logo can be found in multiple variants in the doc/logo directory on github, 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.

We thank Patrick Beart for his contribution to the logo.

CHAPTER 21

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

pypesto.engine, 179
pypesto.logging, 189
pypesto.objective, 70
pypesto.optimize, 120
pypesto.petab, 116
pypesto.problem, 103
pypesto.profile, 133
pypesto.result, 169
pypesto.sampling, 141
pypesto.startpoint, 188
pypesto.visualize, 157

Index

Symbols

__abstractmethods__ (*pypesto.engine.Engine attribute*), 181
__abstractmethods__ (*pypesto.engine.MultiProcessEngine attribute*), 182
__abstractmethods__ (*pypesto.engine.MultiThreadEngine attribute*), 184
__abstractmethods__ (*pypesto.engine.OptimizerTask attribute*), 185
__abstractmethods__ (*pypesto.engine.SingleCoreEngine attribute*), 187
__abstractmethods__ (*pypesto.objective.AmiciObjectBuilder attribute*), 76
__abstractmethods__ (*pypesto.objective.CsvHistory attribute*), 82
__abstractmethods__ (*pypesto.objective.Hdf5History attribute*), 85
__abstractmethods__ (*pypesto.objective.History attribute*), 87
__abstractmethods__ (*pypesto.objective.HistoryBase attribute*), 89
__abstractmethods__ (*pypesto.objective.MemoryHistory attribute*), 94
__abstractmethods__ (*pypesto.objective.OptimizerHistory attribute*), 101
__abstractmethods__ (*pypesto.optimize.DlibOptimizer attribute*), 121
__abstractmethods__ (*pypesto.optimize.Optimizer attribute*), 125
__abstractmethods__ (*pypesto.optimize.PyswarmOptimizer attribute*), 129
__abstractmethods__ (*pypesto.optimize.ScipyOptimizer attribute*), 131
__abstractmethods__ (*pypesto.petab.PetabImporter attribute*), 117
__abstractmethods__ (*pypesto.problem.Iterable attribute*), 105
__abstractmethods__ (*pypesto.problem.List attribute*), 106
__abstractmethods__ (*pypesto.result.Sequence attribute*), 177
__abstractmethods__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 143
__abstractmethods__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 145
__abstractmethods__ (*pypesto.sampling.InternalSampler attribute*), 147
__abstractmethods__ (*pypesto.sampling.MetropolisSampler attribute*), 151
__abstractmethods__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 153
__abstractmethods__ (*pypesto.sampling.Sampler attribute*), 155
__add__ (*pypesto.problem.List attribute*), 106
__args__ (*pypesto.problem.Iterable attribute*), 105
__args__ (*pypesto.problem.List attribute*), 106
__args__ (*pypesto.result.Sequence attribute*), 177
__call__() (*pypesto.objective.AggregatedObjective method*), 71
__call__() (*pypesto.objective.AmiciCalculator*

method), 74
__call__() (*pypesto.objective.AmiciObjective method*), 78
__call__() (*pypesto.objective.Objective method*), 98
__call__() (*pypesto.problem.Objective method*), 110
__class__ (*pypesto.engine.Engine attribute*), 181
__class__ (*pypesto.engine.MultiProcessEngine attribute*), 182
__class__ (*pypesto.engine.MultiThreadEngine attribute*), 184
__class__ (*pypesto.engine.OptimizerTask attribute*), 185
__class__ (*pypesto.engine.SingleCoreEngine attribute*), 187
__class__ (*pypesto.objective.AggregatedObjective attribute*), 71
__class__ (*pypesto.objective.AmiciCalculator attribute*), 75
__class__ (*pypesto.objective.AmiciObjectBuilder attribute*), 76
__class__ (*pypesto.objective.AmiciObjective attribute*), 78
__class__ (*pypesto.objective.CsvHistory attribute*), 82
__class__ (*pypesto.objective.Hdf5History attribute*), 85
__class__ (*pypesto.objective.History attribute*), 87
__class__ (*pypesto.objective.HistoryBase attribute*), 89
__class__ (*pypesto.objective.HistoryOptions attribute*), 92
__class__ (*pypesto.objective.MemoryHistory attribute*), 95
__class__ (*pypesto.objective.Objective attribute*), 98
__class__ (*pypesto.objective.OptimizerHistory attribute*), 101
__class__ (*pypesto.optimize.DlibOptimizer attribute*), 121
__class__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__class__ (*pypesto.optimize.Optimizer attribute*), 125
__class__ (*pypesto.optimize.OptimizerResult attribute*), 127
__class__ (*pypesto.optimize.PyswarmOptimizer attribute*), 129
__class__ (*pypesto.optimize.ScipyOptimizer attribute*), 131
__class__ (*pypesto.petab.PetabImporter attribute*), 117
__class__ (*pypesto.problem.Iterable attribute*), 105
__class__ (*pypesto.problem.List attribute*), 106
__class__ (*pypesto.problem.Objective attribute*), 110
__class__ (*pypesto.problem.Problem attribute*), 114
__class__ (*pypesto.profile.ProfileOptions attribute*), 135
__class__ (*pypesto.profile.ProfilerResult attribute*), 139
__class__ (*pypesto.result.OptimizeResult attribute*), 171
__class__ (*pypesto.result.ProfileResult attribute*), 173
__class__ (*pypesto.result.Result attribute*), 174
__class__ (*pypesto.result.SampleResult attribute*), 176
__class__ (*pypesto.result.Sequence attribute*), 177
__class__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 143
__class__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 145
__class__ (*pypesto.sampling.InternalSampler attribute*), 147
__class__ (*pypesto.sampling.McmcPtResult attribute*), 149
__class__ (*pypesto.sampling.MetropolisSampler attribute*), 151
__class__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 153
__class__ (*pypesto.sampling.Sampler attribute*), 155
__class__ (*pypesto.visualize.ReferencePoint attribute*), 159
__contains__ (*pypesto.problem.List attribute*), 107
__contains__() (*pypesto.objective.HistoryOptions method*), 92
__contains__() (*pypesto.optimize.OptimizeOptions method*), 123
__contains__() (*pypesto.optimize.OptimizerResult method*), 127
__contains__() (*pypesto.profile.ProfileOptions method*), 136
__contains__() (*pypesto.profile.ProfilerResult method*), 139
__contains__() (*pypesto.result.Sequence method*), 177
__contains__() (*pypesto.sampling.McmcPtResult method*), 149
__contains__() (*pypesto.visualize.ReferencePoint method*), 159
__deepcopy__() (*pypesto.objective.AggregatedObjective method*), 71
__deepcopy__() (*pypesto.objective.AmiciObjective method*), 78
__deepcopy__() (*pypesto.objective.Objective method*), 98
__deepcopy__() (*pypesto.problem.Objective method*), 110
__delattr__ (*pypesto.engine.Engine attribute*), 181
__delattr__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__delattr__ (*pypesto.engine.MultiThreadEngine attribute*), 184

__delattr__ (*pypesto.engine.OptimizerTask attribute*), 185
__delattr__ (*pypesto.engine.SingleCoreEngine attribute*), 187
__delattr__ (*pypesto.objective.AggregatedObjective attribute*), 71
__delattr__ (*pypesto.objective.AmiciCalculator attribute*), 75
__delattr__ (*pypesto.objective.AmiciObjectBuilder attribute*), 76
__delattr__ (*pypesto.objective.AmiciObjective attribute*), 78
__delattr__ (*pypesto.objective.CsvHistory attribute*), 82
__delattr__ (*pypesto.objective.Hdf5History attribute*), 85
__delattr__ (*pypesto.objective.History attribute*), 87
__delattr__ (*pypesto.objective.HistoryBase attribute*), 89
__delattr__ (*pypesto.objective.HistoryOptions attribute*), 92
__delattr__ (*pypesto.objective.MemoryHistory attribute*), 95
__delattr__ (*pypesto.objective.Objective attribute*), 98
__delattr__ (*pypesto.objective.OptimizerHistory attribute*), 101
__delattr__ (*pypesto.optimize.DlibOptimizer attribute*), 121
__delattr__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__delattr__ (*pypesto.optimize.Optimizer attribute*), 125
__delattr__ (*pypesto.optimize.OptimizerResult attribute*), 127
__delattr__ (*pypesto.optimize.PyswarmOptimizer attribute*), 129
__delattr__ (*pypesto.optimize.ScipyOptimizer attribute*), 131
__delattr__ (*pypesto.petab.PetabImporter attribute*), 117
__delattr__ (*pypesto.problem.Iterable attribute*), 105
__delattr__ (*pypesto.problem.List attribute*), 107
__delattr__ (*pypesto.problem.Objective attribute*), 110
__delattr__ (*pypesto.problem.Problem attribute*), 114
__delattr__ (*pypesto.profile.ProfileOptions attribute*), 136
__delattr__ (*pypesto.profile.ProfilerResult attribute*), 139
__delattr__ (*pypesto.result.OptimizeResult attribute*), 171
__delattr__ (*pypesto.result.ProfileResult attribute*), 173
__delattr__ (*pypesto.result.Result attribute*), 175
__delattr__ (*pypesto.result.SampleResult attribute*), 176
__delattr__ (*pypesto.result.Sequence attribute*), 177
__delattr__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 143
__delattr__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 145
__delattr__ (*pypesto.sampling.InternalSampler attribute*), 147
__delattr__ (*pypesto.sampling.McmcPtResult attribute*), 149
__delattr__ (*pypesto.sampling.MetropolisSampler attribute*), 151
__delattr__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 153
__delattr__ (*pypesto.sampling.Sampler attribute*), 155
__delattr__ (*pypesto.visualize.ReferencePoint attribute*), 160
__delitem__ (*pypesto.objective.HistoryOptions attribute*), 92
__delitem__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__delitem__ (*pypesto.optimize.OptimizerResult attribute*), 127
__delitem__ (*pypesto.problem.List attribute*), 107
__delitem__ (*pypesto.profile.ProfileOptions attribute*), 136
__delitem__ (*pypesto.profile.ProfilerResult attribute*), 139
__delitem__ (*pypesto.sampling.McmcPtResult attribute*), 149
__delitem__ (*pypesto.visualize.ReferencePoint attribute*), 160
__dict__ (*pypesto.engine.Engine attribute*), 181
__dict__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__dict__ (*pypesto.engine.MultiThreadEngine attribute*), 184
__dict__ (*pypesto.engine.OptimizerTask attribute*), 185
__dict__ (*pypesto.engine.SingleCoreEngine attribute*), 187
__dict__ (*pypesto.objective.AggregatedObjective attribute*), 71
__dict__ (*pypesto.objective.AmiciCalculator attribute*), 75
__dict__ (*pypesto.objective.AmiciObjectBuilder attribute*), 76
__dict__ (*pypesto.objective.AmiciObjective attribute*), 78

__dict__ (*pypesto.objective.CsvHistory attribute*), 82
__dict__ (*pypesto.objective.Hdf5History attribute*), 85
__dict__ (*pypesto.objective.History attribute*), 87
__dict__ (*pypesto.objective.HistoryBase attribute*), 90
__dict__ (*pypesto.objective.HistoryOptions attribute*), 92
__dict__ (*pypesto.objective.MemoryHistory attribute*), 95
__dict__ (*pypesto.objective.Objective attribute*), 98
__dict__ (*pypesto.objective.OptimizerHistory attribute*), 101
__dict__ (*pypesto.optimize.DlibOptimizer attribute*), 121
__dict__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__dict__ (*pypesto.optimize.Optimizer attribute*), 125
__dict__ (*pypesto.optimize.OptimizerResult attribute*), 127
__dict__ (*pypesto.optimize.PyswarmOptimizer attribute*), 129
__dict__ (*pypesto.optimize.ScipyOptimizer attribute*), 131
__dict__ (*pypesto.petab.PetabImporter attribute*), 117
__dict__ (*pypesto.problem.Objective attribute*), 110
__dict__ (*pypesto.problem.Problem attribute*), 114
__dict__ (*pypesto.profile.ProfileOptions attribute*), 136
__dict__ (*pypesto.profile.ProfilerResult attribute*), 139
__dict__ (*pypesto.result.OptimizeResult attribute*), 171
__dict__ (*pypesto.result.ProfileResult attribute*), 173
__dict__ (*pypesto.result.Result attribute*), 175
__dict__ (*pypesto.result.SampleResult attribute*), 176
__dict__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 143
__dict__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 145
__dict__ (*pypesto.sampling.InternalSampler attribute*), 147
__dict__ (*pypesto.sampling.McmcPtResult attribute*), 150
__dict__ (*pypesto.sampling.MetropolisSampler attribute*), 151
__dict__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 153
__dict__ (*pypesto.sampling.Sampler attribute*), 155
__dict__ (*pypesto.visualize.ReferencePoint attribute*), 160
__dir__() (*pypesto.engine.Engine method*), 181
__dir__() (*pypesto.engine.MultiProcessEngine method*), 183
__dir__() (*pypesto.engine.MultiThreadEngine method*), 184
__dir__() (*pypesto.engine.OptimizerTask method*), 185
__dir__() (*pypesto.engine.SingleCoreEngine method*), 187
__dir__() (*pypesto.objective.AggregatedObjective method*), 72
__dir__() (*pypesto.objective.AmiciCalculator method*), 75
__dir__() (*pypesto.objective.AmiciObjectBuilder method*), 76
__dir__() (*pypesto.objective.AmiciObjective method*), 78
__dir__() (*pypesto.objective.CsvHistory method*), 83
__dir__() (*pypesto.objective.Hdf5History method*), 85
__dir__() (*pypesto.objective.History method*), 87
__dir__() (*pypesto.objective.HistoryBase method*), 90
__dir__() (*pypesto.objective.HistoryOptions method*), 92
__dir__() (*pypesto.objective.MemoryHistory method*), 95
__dir__() (*pypesto.objective.Objective method*), 98
__dir__() (*pypesto.objective.OptimizerHistory method*), 101
__dir__() (*pypesto.optimize.DlibOptimizer method*), 121
__dir__() (*pypesto.optimize.OptimizeOptions method*), 123
__dir__() (*pypesto.optimize.Optimizer method*), 125
__dir__() (*pypesto.optimize.OptimizerResult method*), 128
__dir__() (*pypesto.optimize.PyswarmOptimizer method*), 129
__dir__() (*pypesto.optimize.ScipyOptimizer method*), 131
__dir__() (*pypesto.petab.PetabImporter method*), 117
__dir__() (*pypesto.problem.Iterable method*), 105
__dir__() (*pypesto.problem.List method*), 107
__dir__() (*pypesto.problem.Objective method*), 110
__dir__() (*pypesto.problem.Problem method*), 114
__dir__() (*pypesto.profile.ProfileOptions method*), 136
__dir__() (*pypesto.profile.ProfilerResult method*), 139
__dir__() (*pypesto.result.OptimizeResult method*), 171
__dir__() (*pypesto.result.ProfileResult method*), 173
__dir__() (*pypesto.result.Result method*), 175
__dir__() (*pypesto.result.SampleResult method*), 176
__dir__() (*pypesto.result.Sequence method*), 177
__dir__() (*pypesto.sampling.AdaptiveMetropolisSampler method*), 143
__dir__() (*pypesto.sampling.AdaptiveParallelTemperingSampler method*), 145
__dir__() (*pypesto.sampling.InternalSampler*

method), 147
__dir__() (*pypesto.sampling.McmcPtResult method*), 150
__dir__() (*pypesto.sampling.MetropolisSampler method*), 151
__dir__() (*pypesto.sampling.ParallelTemperingSampler method*), 153
__dir__() (*pypesto.sampling.Sampler method*), 155
__dir__() (*pypesto.visualize.ReferencePoint method*), 160
__eq__ (*pypesto.engine.Engine attribute*), 181
__eq__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__eq__ (*pypesto.engine.MultiThreadEngine attribute*), 184
__eq__ (*pypesto.engine.OptimizerTask attribute*), 186
__eq__ (*pypesto.engine.SingleCoreEngine attribute*), 187
__eq__ (*pypesto.objective.AggregatedObjective attribute*), 72
__eq__ (*pypesto.objective.AmiciCalculator attribute*), 75
__eq__ (*pypesto.objective.AmiciObjectBuilder attribute*), 76
__eq__ (*pypesto.objective.AmiciObjective attribute*), 79
__eq__ (*pypesto.objective.CsvHistory attribute*), 83
__eq__ (*pypesto.objective.Hdf5History attribute*), 85
__eq__ (*pypesto.objective.History attribute*), 87
__eq__ (*pypesto.objective.HistoryBase attribute*), 90
__eq__ (*pypesto.objective.HistoryOptions attribute*), 93
__eq__ (*pypesto.objective.MemoryHistory attribute*), 95
__eq__ (*pypesto.objective.Objective attribute*), 99
__eq__ (*pypesto.objective.OptimizerHistory attribute*), 101
__eq__ (*pypesto.optimize.DlibOptimizer attribute*), 121
__eq__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__eq__ (*pypesto.optimize.Optimizer attribute*), 125
__eq__ (*pypesto.optimize.OptimizerResult attribute*), 128
__eq__ (*pypesto.optimize.PyswarmOptimizer attribute*), 130
__eq__ (*pypesto.optimize.ScipyOptimizer attribute*), 131
__eq__ (*pypesto.petab.PetabImporter attribute*), 117
__eq__ (*pypesto.problem.Iterable attribute*), 105
__eq__ (*pypesto.problem.List attribute*), 107
__eq__ (*pypesto.problem.Objective attribute*), 110
__eq__ (*pypesto.problem.Problem attribute*), 114
__eq__ (*pypesto.profile.ProfileOptions attribute*), 136
__eq__ (*pypesto.profile.ProfilerResult attribute*), 139
__eq__ (*pypesto.result.OptimizeResult attribute*), 171
__eq__ (*pypesto.result.ProfileResult attribute*), 173
__eq__ (*pypesto.result.Result attribute*), 175
__eq__ (*pypesto.result.SampleResult attribute*), 176
__eq__ (*pypesto.result.Sequence attribute*), 177
__eq__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 143
__eq__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 145
__eq__ (*pypesto.sampling.InternalSampler attribute*), 147
__eq__ (*pypesto.sampling.McmcPtResult attribute*), 150
__eq__ (*pypesto.sampling.MetropolisSampler attribute*), 151
__eq__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 154
__eq__ (*pypesto.sampling.Sampler attribute*), 155
__eq__ (*pypesto.visualize.ReferencePoint attribute*), 160
__extra__ (*pypesto.problem.Iterable attribute*), 105
__extra__ (*pypesto.problem.List attribute*), 107
__extra__ (*pypesto.result.Sequence attribute*), 177
__format__() (*pypesto.engine.Engine method*), 181
__format__() (*pypesto.engine.MultiProcessEngine method*), 183
__format__() (*pypesto.engine.MultiThreadEngine method*), 184
__format__() (*pypesto.engine.OptimizerTask method*), 186
__format__() (*pypesto.engine.SingleCoreEngine method*), 187
__format__() (*pypesto.objective.AggregatedObjective method*), 72
__format__() (*pypesto.objective.AmiciCalculator method*), 75
__format__() (*pypesto.objective.AmiciObjectBuilder method*), 76
__format__() (*pypesto.objective.AmiciObjective method*), 79
__format__() (*pypesto.objective.CsvHistory method*), 83
__format__() (*pypesto.objective.Hdf5History method*), 85
__format__() (*pypesto.objective.History method*), 87
__format__() (*pypesto.objective.HistoryBase method*), 90
__format__() (*pypesto.objective.HistoryOptions method*), 93
__format__() (*pypesto.objective.MemoryHistory method*), 95
__format__() (*pypesto.objective.Objective method*), 99
__format__() (*pypesto.objective.OptimizerHistory method*), 101
__format__() (*pypesto.optimize.DlibOptimizer*

method), 121
__format__() (*pypesto.optimize.OptimizeOptions method*), 123
__format__() (*pypesto.optimize.Optimizer method*), 125
__format__() (*pypesto.optimize.OptimizerResult method*), 128
__format__() (*pypesto.optimize.PyswarmOptimizer method*), 130
__format__() (*pypesto.optimize.ScipyOptimizer method*), 131
__format__() (*pypesto.petab.PetabImporter method*), 117
__format__() (*pypesto.problem.Iterable method*), 105
__format__() (*pypesto.problem.List method*), 107
__format__() (*pypesto.problem.Objective method*), 111
__format__() (*pypesto.problem.Problem method*), 114
__format__() (*pypesto.profile.ProfileOptions method*), 136
__format__() (*pypesto.profile.ProfilerResult method*), 139
__format__() (*pypesto.result.OptimizeResult method*), 171
__format__() (*pypesto.result.ProfileResult method*), 173
__format__() (*pypesto.result.Result method*), 175
__format__() (*pypesto.result.SampleResult method*), 176
__format__() (*pypesto.result.Sequence method*), 177
__format__() (*pypesto.sampling.AdaptiveMetropolisSampler method*), 143
__format__() (*pypesto.sampling.AdaptiveParallelTemperingSampler method*), 145
__format__() (*pypesto.sampling.InternalSampler method*), 147
__format__() (*pypesto.sampling.McmcPtResult method*), 150
__format__() (*pypesto.sampling.MetropolisSampler method*), 152
__format__() (*pypesto.sampling.ParallelTemperingSampler method*), 154
__format__() (*pypesto.sampling.Sampler method*), 156
__format__() (*pypesto.visualize.ReferencePoint method*), 160
__ge__ (*pypesto.engine.Engine attribute*), 181
__ge__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__ge__ (*pypesto.engine.MultiThreadEngine attribute*), 184
__ge__ (*pypesto.engine.OptimizerTask attribute*), 186
__ge__ (*pypesto.engine.SingleCoreEngine attribute*), 187
__ge__ (*pypesto.objective.AggregatedObjective attribute*), 72
__ge__ (*pypesto.objective.AmiciCalculator attribute*), 75
__ge__ (*pypesto.objective.AmiciObjectBuilder attribute*), 77
__ge__ (*pypesto.objective.AmiciObjective attribute*), 79
__ge__ (*pypesto.objective.CsvHistory attribute*), 83
__ge__ (*pypesto.objective.Hdf5History attribute*), 85
__ge__ (*pypesto.objective.History attribute*), 87
__ge__ (*pypesto.objective.HistoryBase attribute*), 90
__ge__ (*pypesto.objective.HistoryOptions attribute*), 93
__ge__ (*pypesto.objective.MemoryHistory attribute*), 95
__ge__ (*pypesto.objective.Objective attribute*), 99
__ge__ (*pypesto.objective.OptimizerHistory attribute*), 102
__ge__ (*pypesto.optimize.DlibOptimizer attribute*), 121
__ge__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__ge__ (*pypesto.optimize.Optimizer attribute*), 125
__ge__ (*pypesto.optimize.OptimizerResult attribute*), 128
__ge__ (*pypesto.optimize.PyswarmOptimizer attribute*), 130
__ge__ (*pypesto.optimize.ScipyOptimizer attribute*), 131
__ge__ (*pypesto.petab.PetabImporter attribute*), 117
__ge__ (*pypesto.problem.Iterable attribute*), 105
__ge__ (*pypesto.problem.List attribute*), 107
__ge__ (*pypesto.problem.Objective attribute*), 111
__ge__ (*pypesto.problem.Problem attribute*), 114
__ge__ (*pypesto.profile.ProfileOptions attribute*), 136
__ge__ (*pypesto.profile.ProfilerResult attribute*), 139
__ge__ (*pypesto.result.OptimizeResult attribute*), 171
__ge__ (*pypesto.result.ProfileResult attribute*), 173
__ge__ (*pypesto.result.Result attribute*), 175
__ge__ (*pypesto.result.SampleResult attribute*), 176
__ge__ (*pypesto.result.Sequence attribute*), 177
__ge__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 143
__ge__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 145
__ge__ (*pypesto.sampling.InternalSampler attribute*), 147
__ge__ (*pypesto.sampling.McmcPtResult attribute*), 150
__ge__ (*pypesto.sampling.MetropolisSampler attribute*), 152
__ge__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 154
__ge__ (*pypesto.sampling.Sampler attribute*), 156

__ge__ (*pypesto.visualize.ReferencePoint attribute*), 160
__getattr__() (*pypesto.objective.HistoryOptions method*), 93
__getattr__() (*pypesto.optimize.OptimizeOptions method*), 123
__getattr__() (*pypesto.optimize.OptimizerResult method*), 128
__getattr__() (*pypesto.profile.ProfileOptions method*), 136
__getattr__() (*pypesto.profile.ProfilerResult method*), 139
__getattr__() (*pypesto.sampling.McmcPtResult method*), 150
__getattr__() (*pypesto.visualize.ReferencePoint method*), 160
__getattribute__ (*pypesto.engine.Engine attribute*), 181
__getattribute__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__getattribute__ (*pypesto.engine.MultiThreadEngine attribute*), 184
__getattribute__ (*pypesto.engine.OptimizerTask attribute*), 186
__getattribute__ (*pypesto.engine.SingleCoreEngine attribute*), 187
__getattribute__ (*pypesto.objective.AggregatedObjective attribute*), 72
__getattribute__ (*pypesto.objective.AmiciCalculator attribute*), 75
__getattribute__ (*pypesto.objective.AmiciObjectBuilder attribute*), 77
__getattribute__ (*pypesto.objective.AmiciObjective attribute*), 79
__getattribute__ (*pypesto.objective.CsvHistory attribute*), 83
__getattribute__ (*pypesto.objective.Hdf5History attribute*), 85
__getattribute__ (*pypesto.objective.History attribute*), 87
__getattribute__ (*pypesto.objective.HistoryBase attribute*), 90
__getattribute__ (*pypesto.objective.HistoryOptions attribute*), 93
__getattribute__ (*pypesto.objective.MemoryHistory attribute*), 95
__getattribute__ (*pypesto.objective.Objective attribute*), 99
__getattribute__ (*pypesto.objective.OptimizerHistory attribute*), 102
__getattribute__ (*pypesto.optimize.DlibOptimizer attribute*), 121
__getattribute__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__getattribute__ (*pypesto.optimize.Optimizer attribute*), 125
__getattribute__ (*pypesto.optimize.OptimizerResult attribute*), 128
__getattribute__ (*pypesto.optimize.PyswarmOptimizer attribute*), 130
__getattribute__ (*pypesto.optimize.ScipyOptimizer attribute*), 131
__getattribute__ (*pypesto.petab.PetabImporter attribute*), 117
__getattribute__ (*pypesto.problem.Iterable attribute*), 105
__getattribute__ (*pypesto.problem.List attribute*), 107
__getattribute__ (*pypesto.problem.Objective attribute*), 111
__getattribute__ (*pypesto.problem.Problem attribute*), 114
__getattribute__ (*pypesto.profile.ProfileOptions attribute*), 136
__getattribute__ (*pypesto.profile.ProfilerResult attribute*), 139
__getattribute__ (*pypesto.result.OptimizeResult attribute*), 171
__getattribute__ (*pypesto.result.ProfileResult attribute*), 173
__getattribute__ (*pypesto.result.Result attribute*), 175
__getattribute__ (*pypesto.result.SampleResult attribute*), 176
__getattribute__ (*pypesto.result.Sequence attribute*), 177
__getattribute__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 143
__getattribute__ (*pypesto.sampling.AdaptiveParallelTemperingSamp attribute*), 145
__getattribute__ (*pypesto.sampling.InternalSampler attribute*), 147
__getattribute__ (*pypesto.sampling.McmcPtResult attribute*), 150
__getattribute__ (*pypesto.sampling.MetropolisSampler attribute*), 152
__getattribute__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 154
__getattribute__ (*pypesto.sampling.Sampler attribute*), 156
__getattribute__ (*pypesto.visualize.ReferencePoint attribute*), 160
__getitem__() (*pypesto.objective.HistoryOptions method*), 93
__getitem__() (*pypesto.optimize.OptimizeOptions method*), 123
__getitem__() (*pypesto.optimize.OptimizerResult method*), 128

__getitem__() (*pypesto.problem.List method*), 107
__getitem__() (*pypesto.profile.ProfileOptions method*), 136
__getitem__() (*pypesto.profile.ProfilerResult method*), 139
__getitem__() (*pypesto.result.Sequence method*), 177
__getitem__() (*pypesto.sampling.McmcPtResult method*), 150
__getitem__() (*pypesto.visualize.ReferencePoint method*), 160
__getstate__() (*pypesto.objective.AmiciObjective method*), 79
__gt__ (*pypesto.engine.Engine attribute*), 181
__gt__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__gt__ (*pypesto.engine.MultiThreadEngine attribute*), 184
__gt__ (*pypesto.engine.OptimizerTask attribute*), 186
__gt__ (*pypesto.engine.SingleCoreEngine attribute*), 187
__gt__ (*pypesto.objective.AggregatedObjective attribute*), 72
__gt__ (*pypesto.objective.AmiciCalculator attribute*), 75
__gt__ (*pypesto.objective.AmiciObjectBuilder attribute*), 77
__gt__ (*pypesto.objective.AmiciObjective attribute*), 79
__gt__ (*pypesto.objective.CsvHistory attribute*), 83
__gt__ (*pypesto.objective.Hdf5History attribute*), 85
__gt__ (*pypesto.objective.History attribute*), 87
__gt__ (*pypesto.objective.HistoryBase attribute*), 90
__gt__ (*pypesto.objective.HistoryOptions attribute*), 93
__gt__ (*pypesto.objective.MemoryHistory attribute*), 95
__gt__ (*pypesto.objective.Objective attribute*), 99
__gt__ (*pypesto.objective.OptimizerHistory attribute*), 102
__gt__ (*pypesto.optimize.DlibOptimizer attribute*), 121
__gt__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__gt__ (*pypesto.optimize.Optimizer attribute*), 125
__gt__ (*pypesto.optimize.OptimizerResult attribute*), 128
__gt__ (*pypesto.optimize.PyswarmOptimizer attribute*), 130
__gt__ (*pypesto.optimize.ScipyOptimizer attribute*), 131
__gt__ (*pypesto.petab.PetabImporter attribute*), 117
__gt__ (*pypesto.problem.Iterable attribute*), 105
__gt__ (*pypesto.problem.List attribute*), 107
__gt__ (*pypesto.problem.Objective attribute*), 111
__gt__ (*pypesto.problem.Problem attribute*), 114
__gt__ (*pypesto.profile.ProfileOptions attribute*), 136
__gt__ (*pypesto.profile.ProfilerResult attribute*), 139
__gt__ (*pypesto.result.OptimizeResult attribute*), 171
__gt__ (*pypesto.result.ProfileResult attribute*), 173
__gt__ (*pypesto.result.Result attribute*), 175
__gt__ (*pypesto.result.SampleResult attribute*), 176
__gt__ (*pypesto.result.Sequence attribute*), 177
__gt__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 143
__gt__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 145
__gt__ (*pypesto.sampling.InternalSampler attribute*), 147
__gt__ (*pypesto.sampling.McmcPtResult attribute*), 150
__gt__ (*pypesto.sampling.MetropolisSampler attribute*), 152
__gt__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 154
__gt__ (*pypesto.sampling.Sampler attribute*), 156
__gt__ (*pypesto.visualize.ReferencePoint attribute*), 160
__hash__ (*pypesto.engine.Engine attribute*), 181
__hash__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__hash__ (*pypesto.engine.MultiThreadEngine attribute*), 184
__hash__ (*pypesto.engine.OptimizerTask attribute*), 186
__hash__ (*pypesto.engine.SingleCoreEngine attribute*), 187
__hash__ (*pypesto.objective.AggregatedObjective attribute*), 72
__hash__ (*pypesto.objective.AmiciCalculator attribute*), 75
__hash__ (*pypesto.objective.AmiciObjectBuilder attribute*), 77
__hash__ (*pypesto.objective.AmiciObjective attribute*), 79
__hash__ (*pypesto.objective.CsvHistory attribute*), 83
__hash__ (*pypesto.objective.Hdf5History attribute*), 85
__hash__ (*pypesto.objective.History attribute*), 88
__hash__ (*pypesto.objective.HistoryBase attribute*), 90
__hash__ (*pypesto.objective.HistoryOptions attribute*), 93
__hash__ (*pypesto.objective.MemoryHistory attribute*), 95
__hash__ (*pypesto.objective.Objective attribute*), 99
__hash__ (*pypesto.objective.OptimizerHistory attribute*), 102
__hash__ (*pypesto.optimize.DlibOptimizer attribute*), 121
__hash__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__hash__ (*pypesto.optimize.Optimizer attribute*), 125

__hash__ (*pypesto.optimize.OptimizerResult attribute*), 128
__hash__ (*pypesto.optimize.PyswarmOptimizer attribute*), 130
__hash__ (*pypesto.optimize.ScipyOptimizer attribute*), 131
__hash__ (*pypesto.petab.PetabImporter attribute*), 117
__hash__ (*pypesto.problem.Iterable attribute*), 105
__hash__ (*pypesto.problem.List attribute*), 107
__hash__ (*pypesto.problem.Objective attribute*), 111
__hash__ (*pypesto.problem.Problem attribute*), 114
__hash__ (*pypesto.profile.ProfileOptions attribute*), 136
__hash__ (*pypesto.profile.ProfilerResult attribute*), 139
__hash__ (*pypesto.result.OptimizeResult attribute*), 171
__hash__ (*pypesto.result.ProfileResult attribute*), 173
__hash__ (*pypesto.result.Result attribute*), 175
__hash__ (*pypesto.result.SampleResult attribute*), 176
__hash__ (*pypesto.result.Sequence attribute*), 178
__hash__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 143
__hash__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 146
__hash__ (*pypesto.sampling.InternalSampler attribute*), 147
__hash__ (*pypesto.sampling.McmcPtResult attribute*), 150
__hash__ (*pypesto.sampling.MetropolisSampler attribute*), 152
__hash__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 154
__hash__ (*pypesto.sampling.Sampler attribute*), 156
__hash__ (*pypesto.visualize.ReferencePoint attribute*), 160
__iadd__ (*pypesto.problem.List attribute*), 107
__imul__ (*pypesto.problem.List attribute*), 107
__init__ (*pypesto.objective.AmiciCalculator attribute*), 75
__init__ (*pypesto.objective.AmiciObjectBuilder attribute*), 77
__init__ (*pypesto.objective.HistoryBase attribute*), 90
__init__ (*pypesto.problem.Iterable attribute*), 105
__init__ (*pypesto.problem.List attribute*), 107
__init__ (*pypesto.result.Sequence attribute*), 178
__init__() (*pypesto.engine.Engine method*), 181
__init__() (*pypesto.engine.MultiProcessEngine method*), 183
__init__() (*pypesto.engine.MultiThreadEngine method*), 184
__init__() (*pypesto.engine.OptimizerTask method*), 186
__init__() (*pypesto.engine.SingleCoreEngine method*), 187
__init__() (*pypesto.objective.AggregatedObjective method*), 72
__init__() (*pypesto.objective.AmiciObjective method*), 79
__init__() (*pypesto.objective.CsvHistory method*), 83
__init__() (*pypesto.objective.Hdf5History method*), 85
__init__() (*pypesto.objective.History method*), 88
__init__() (*pypesto.objective.HistoryOptions method*), 93
__init__() (*pypesto.objective.MemoryHistory method*), 95
__init__() (*pypesto.objective.Objective method*), 99
__init__() (*pypesto.objective.OptimizerHistory method*), 102
__init__() (*pypesto.optimize.DlibOptimizer method*), 121
__init__() (*pypesto.optimize.OptimizeOptions method*), 123
__init__() (*pypesto.optimize.Optimizer method*), 125
__init__() (*pypesto.optimize.OptimizerResult method*), 128
__init__() (*pypesto.optimize.PyswarmOptimizer method*), 130
__init__() (*pypesto.optimize.ScipyOptimizer method*), 131
__init__() (*pypesto.petab.PetabImporter method*), 117
__init__() (*pypesto.problem.Objective method*), 111
__init__() (*pypesto.problem.Problem method*), 114
__init__() (*pypesto.profile.ProfileOptions method*), 136
__init__() (*pypesto.profile.ProfilerResult method*), 139
__init__() (*pypesto.result.OptimizeResult method*), 171
__init__() (*pypesto.result.ProfileResult method*), 173
__init__() (*pypesto.result.Result method*), 175
__init__() (*pypesto.result.SampleResult method*), 176
__init__() (*pypesto.sampling.AdaptiveMetropolisSampler method*), 143
__init__() (*pypesto.sampling.AdaptiveParallelTemperingSampler method*), 146
__init__() (*pypesto.sampling.InternalSampler method*), 148
__init__() (*pypesto.sampling.McmcPtResult method*), 150
__init__() (*pypesto.sampling.MetropolisSampler method*), 152
__init__() (*pypesto.sampling.ParallelTemperingSampler method*), 154
__init__() (*pypesto.sampling.Sampler method*), 156

__init__() (*pypesto.visualize.ReferencePoint method*), 160
__init_subclass__() (*pypesto.engine.Engine method*), 182
__init_subclass__() (*pypesto.engine.MultiProcessEngine method*), 183
__init_subclass__() (*pypesto.engine.MultiThreadEngine method*), 184
__init_subclass__() (*pypesto.engine.OptimizerTask method*), 186
__init_subclass__() (*pypesto.engine.SingleCoreEngine method*), 187
__init_subclass__() (*pypesto.objective.AggregatedObjective method*), 72
__init_subclass__() (*pypesto.objective.AmiciCalculator method*), 75
__init_subclass__() (*pypesto.objective.AmiciObjectBuilder method*), 77
__init_subclass__() (*pypesto.objective.AmiciObjective method*), 79
__init_subclass__() (*pypesto.objective.CsvHistory method*), 83
__init_subclass__() (*pypesto.objective.Hdf5History method*), 85
__init_subclass__() (*pypesto.objective.History method*), 88
__init_subclass__() (*pypesto.objective.HistoryBase method*), 90
__init_subclass__() (*pypesto.objective.HistoryOptions method*), 93
__init_subclass__() (*pypesto.objective.MemoryHistory method*), 95
__init_subclass__() (*pypesto.objective.Objective method*), 99
__init_subclass__() (*pypesto.objective.OptimizerHistory method*), 102
__init_subclass__() (*pypesto.optimize.DlibOptimizer method*), 122
__init_subclass__() (*pypesto.optimize.OptimizeOptions method*), 123
__init_subclass__() (*pypesto.optimize.Optimizer method*), 125
__init_subclass__() (*pypesto.optimize.OptimizerResult method*), 128
__init_subclass__() (*pypesto.optimize.PyswarmOptimizer method*), 130
__init_subclass__() (*pypesto.optimize.ScipyOptimizer method*), 131
__init_subclass__() (*pypesto.petab.PetabImporter method*), 118
__init_subclass__() (*pypesto.problem.Iterable method*), 106
__init_subclass__() (*pypesto.problem.List method*), 107
__init_subclass__() (*pypesto.problem.Objective method*), 111
__init_subclass__() (*pypesto.problem.Problem method*), 114
__init_subclass__() (*pypesto.profile.ProfileOptions method*), 136
__init_subclass__() (*pypesto.profile.ProfilerResult method*), 139
__init_subclass__() (*pypesto.result.OptimizeResult method*), 172
__init_subclass__() (*pypesto.result.ProfileResult method*), 173
__init_subclass__() (*pypesto.result.Result method*), 175
__init_subclass__() (*pypesto.result.SampleResult method*), 176
__init_subclass__() (*pypesto.result.Sequence method*), 178
__init_subclass__() (*pypesto.sampling.AdaptiveMetropolisSampler method*), 144
__init_subclass__() (*pypesto.sampling.AdaptiveParallelTemperingSampler method*), 146
__init_subclass__() (*pypesto.sampling.InternalSampler method*), 148
__init_subclass__() (*pypesto.sampling.McmcPtResult method*), 150
__init_subclass__() (*pypesto.sampling.MetropolisSampler method*), 152
__init_subclass__() (*pypesto.sampling.ParallelTemperingSampler method*), 154
__init_subclass__() (*pypesto.sampling.Sampler*

method), 156
__init_subclass__() (*pypesto.visualize.ReferencePoint method*), 160
__iter__ (*pypesto.objective.HistoryOptions attribute*), 93
__iter__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__iter__ (*pypesto.optimize.OptimizerResult attribute*), 128
__iter__ (*pypesto.problem.List attribute*), 107
__iter__ (*pypesto.profile.ProfileOptions attribute*), 136
__iter__ (*pypesto.profile.ProfilerResult attribute*), 139
__iter__ (*pypesto.sampling.McmcPtResult attribute*), 150
__iter__ (*pypesto.visualize.ReferencePoint attribute*), 160
__iter__() (*pypesto.problem.Iterable method*), 106
__iter__() (*pypesto.result.Sequence method*), 178
__le__ (*pypesto.engine.Engine attribute*), 182
__le__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__le__ (*pypesto.engine.MultiThreadEngine attribute*), 184
__le__ (*pypesto.engine.OptimizerTask attribute*), 186
__le__ (*pypesto.engine.SingleCoreEngine attribute*), 188
__le__ (*pypesto.objective.AggregatedObjective attribute*), 72
__le__ (*pypesto.objective.AmiciCalculator attribute*), 75
__le__ (*pypesto.objective.AmiciObjectBuilder attribute*), 77
__le__ (*pypesto.objective.AmiciObjective attribute*), 80
__le__ (*pypesto.objective.CsvHistory attribute*), 83
__le__ (*pypesto.objective.Hdf5History attribute*), 85
__le__ (*pypesto.objective.History attribute*), 88
__le__ (*pypesto.objective.HistoryBase attribute*), 90
__le__ (*pypesto.objective.HistoryOptions attribute*), 93
__le__ (*pypesto.objective.MemoryHistory attribute*), 95
__le__ (*pypesto.objective.Objective attribute*), 99
__le__ (*pypesto.objective.OptimizerHistory attribute*), 102
__le__ (*pypesto.optimize.DlibOptimizer attribute*), 122
__le__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__le__ (*pypesto.optimize.Optimizer attribute*), 125
__le__ (*pypesto.optimize.OptimizerResult attribute*), 128
__le__ (*pypesto.optimize.PyswarmOptimizer attribute*), 130
__le__ (*pypesto.optimize.ScipyOptimizer attribute*), 131
__le__ (*pypesto.petab.PetabImporter attribute*), 118
__le__ (*pypesto.problem.Iterable attribute*), 106
__le__ (*pypesto.problem.List attribute*), 107
__le__ (*pypesto.problem.Objective attribute*), 111
__le__ (*pypesto.problem.Problem attribute*), 115
__le__ (*pypesto.profile.ProfileOptions attribute*), 136
__le__ (*pypesto.profile.ProfilerResult attribute*), 139
__le__ (*pypesto.result.OptimizeResult attribute*), 172
__le__ (*pypesto.result.ProfileResult attribute*), 173
__le__ (*pypesto.result.Result attribute*), 175
__le__ (*pypesto.result.SampleResult attribute*), 176
__le__ (*pypesto.result.Sequence attribute*), 178
__le__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 144
__le__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 146
__le__ (*pypesto.sampling.InternalSampler attribute*), 148
__le__ (*pypesto.sampling.McmcPtResult attribute*), 150
__le__ (*pypesto.sampling.MetropolisSampler attribute*), 152
__le__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 154
__le__ (*pypesto.sampling.Sampler attribute*), 156
__le__ (*pypesto.visualize.ReferencePoint attribute*), 160
__len__ (*pypesto.objective.HistoryOptions attribute*), 93
__len__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__len__ (*pypesto.optimize.OptimizerResult attribute*), 128
__len__ (*pypesto.problem.List attribute*), 107
__len__ (*pypesto.profile.ProfileOptions attribute*), 136
__len__ (*pypesto.profile.ProfilerResult attribute*), 139
__len__ (*pypesto.sampling.McmcPtResult attribute*), 150
__len__ (*pypesto.visualize.ReferencePoint attribute*), 160
__len__() (*pypesto.result.Sequence method*), 178
__lt__ (*pypesto.engine.Engine attribute*), 182
__lt__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__lt__ (*pypesto.engine.MultiThreadEngine attribute*), 185
__lt__ (*pypesto.engine.OptimizerTask attribute*), 186
__lt__ (*pypesto.engine.SingleCoreEngine attribute*), 188
__lt__ (*pypesto.objective.AggregatedObjective attribute*), 72
__lt__ (*pypesto.objective.AmiciCalculator attribute*), 75

__lt__ (*pypesto.objective.AmiciObjectBuilder attribute*), 77
__lt__ (*pypesto.objective.AmiciObjective attribute*), 80
__lt__ (*pypesto.objective.CsvHistory attribute*), 83
__lt__ (*pypesto.objective.Hdf5History attribute*), 85
__lt__ (*pypesto.objective.History attribute*), 88
__lt__ (*pypesto.objective.HistoryBase attribute*), 90
__lt__ (*pypesto.objective.HistoryOptions attribute*), 93
__lt__ (*pypesto.objective.MemoryHistory attribute*), 95
__lt__ (*pypesto.objective.Objective attribute*), 99
__lt__ (*pypesto.objective.OptimizerHistory attribute*), 102
__lt__ (*pypesto.optimize.DlibOptimizer attribute*), 122
__lt__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__lt__ (*pypesto.optimize.Optimizer attribute*), 125
__lt__ (*pypesto.optimize.OptimizerResult attribute*), 128
__lt__ (*pypesto.optimize.PyswarmOptimizer attribute*), 130
__lt__ (*pypesto.optimize.ScipyOptimizer attribute*), 131
__lt__ (*pypesto.petab.PetabImporter attribute*), 118
__lt__ (*pypesto.problem.Iterable attribute*), 106
__lt__ (*pypesto.problem.List attribute*), 107
__lt__ (*pypesto.problem.Objective attribute*), 111
__lt__ (*pypesto.problem.Problem attribute*), 115
__lt__ (*pypesto.profile.ProfileOptions attribute*), 136
__lt__ (*pypesto.profile.ProfilerResult attribute*), 139
__lt__ (*pypesto.result.OptimizeResult attribute*), 172
__lt__ (*pypesto.result.ProfileResult attribute*), 173
__lt__ (*pypesto.result.Result attribute*), 175
__lt__ (*pypesto.result.SampleResult attribute*), 176
__lt__ (*pypesto.result.Sequence attribute*), 178
__lt__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 144
__lt__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 146
__lt__ (*pypesto.sampling.InternalSampler attribute*), 148
__lt__ (*pypesto.sampling.McmcPtResult attribute*), 150
__lt__ (*pypesto.sampling.MetropolisSampler attribute*), 152
__lt__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 154
__lt__ (*pypesto.sampling.Sampler attribute*), 156
__lt__ (*pypesto.visualize.ReferencePoint attribute*), 160
__module__ (*pypesto.engine.Engine attribute*), 182
__module__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__module__ (*pypesto.engine.MultiThreadEngine attribute*), 185
__module__ (*pypesto.engine.OptimizerTask attribute*), 186
__module__ (*pypesto.engine.SingleCoreEngine attribute*), 188
__module__ (*pypesto.objective.AggregatedObjective attribute*), 72
__module__ (*pypesto.objective.AmiciCalculator attribute*), 76
__module__ (*pypesto.objective.AmiciObjectBuilder attribute*), 77
__module__ (*pypesto.objective.AmiciObjective attribute*), 80
__module__ (*pypesto.objective.CsvHistory attribute*), 83
__module__ (*pypesto.objective.Hdf5History attribute*), 85
__module__ (*pypesto.objective.History attribute*), 88
__module__ (*pypesto.objective.HistoryBase attribute*), 90
__module__ (*pypesto.objective.HistoryOptions attribute*), 93
__module__ (*pypesto.objective.MemoryHistory attribute*), 95
__module__ (*pypesto.objective.Objective attribute*), 99
__module__ (*pypesto.objective.OptimizerHistory attribute*), 102
__module__ (*pypesto.optimize.DlibOptimizer attribute*), 122
__module__ (*pypesto.optimize.OptimizeOptions attribute*), 123
__module__ (*pypesto.optimize.Optimizer attribute*), 125
__module__ (*pypesto.optimize.OptimizerResult attribute*), 128
__module__ (*pypesto.optimize.PyswarmOptimizer attribute*), 130
__module__ (*pypesto.optimize.ScipyOptimizer attribute*), 131
__module__ (*pypesto.petab.PetabImporter attribute*), 118
__module__ (*pypesto.problem.Iterable attribute*), 106
__module__ (*pypesto.problem.List attribute*), 107
__module__ (*pypesto.problem.Objective attribute*), 111
__module__ (*pypesto.problem.Problem attribute*), 115
__module__ (*pypesto.profile.ProfileOptions attribute*), 136
__module__ (*pypesto.profile.ProfilerResult attribute*), 139
__module__ (*pypesto.result.OptimizeResult attribute*), 172
__module__ (*pypesto.result.ProfileResult attribute*),

173
__module__ (*pypesto.result.Result attribute*), 175
__module__ (*pypesto.result.SampleResult attribute*), 176
__module__ (*pypesto.result.Sequence attribute*), 178
__module__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 144
__module__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 146
__module__ (*pypesto.sampling.InternalSampler attribute*), 148
__module__ (*pypesto.sampling.McmcPtResult attribute*), 150
__module__ (*pypesto.sampling.MetropolisSampler attribute*), 152
__module__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 154
__module__ (*pypesto.sampling.Sampler attribute*), 156
__module__ (*pypesto.visualize.ReferencePoint attribute*), 160
__mul__ (*pypesto.problem.List attribute*), 107
__ne__ (*pypesto.engine.Engine attribute*), 182
__ne__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__ne__ (*pypesto.engine.MultiThreadEngine attribute*), 185
__ne__ (*pypesto.engine.OptimizerTask attribute*), 186
__ne__ (*pypesto.engine.SingleCoreEngine attribute*), 188
__ne__ (*pypesto.objective.AggregatedObjective attribute*), 72
__ne__ (*pypesto.objective.AmiciCalculator attribute*), 76
__ne__ (*pypesto.objective.AmiciObjectBuilder attribute*), 77
__ne__ (*pypesto.objective.AmiciObjective attribute*), 80
__ne__ (*pypesto.objective.CsvHistory attribute*), 83
__ne__ (*pypesto.objective.Hdf5History attribute*), 85
__ne__ (*pypesto.objective.History attribute*), 88
__ne__ (*pypesto.objective.HistoryBase attribute*), 90
__ne__ (*pypesto.objective.HistoryOptions attribute*), 93
__ne__ (*pypesto.objective.MemoryHistory attribute*), 95
__ne__ (*pypesto.objective.Objective attribute*), 99
__ne__ (*pypesto.objective.OptimizerHistory attribute*), 102
__ne__ (*pypesto.optimize.DlibOptimizer attribute*), 122
__ne__ (*pypesto.optimize.OptimizeOptions attribute*), 124
__ne__ (*pypesto.optimize.Optimizer attribute*), 125
__ne__ (*pypesto.optimize.OptimizerResult attribute*), 128
__ne__ (*pypesto.optimize.PyswarmOptimizer attribute*), 130
__ne__ (*pypesto.optimize.ScipyOptimizer attribute*), 131
__ne__ (*pypesto.petab.PetabImporter attribute*), 118
__ne__ (*pypesto.problem.Iterable attribute*), 106
__ne__ (*pypesto.problem.List attribute*), 108
__ne__ (*pypesto.problem.Objective attribute*), 111
__ne__ (*pypesto.problem.Problem attribute*), 115
__ne__ (*pypesto.profile.ProfileOptions attribute*), 136
__ne__ (*pypesto.profile.ProfilerResult attribute*), 140
__ne__ (*pypesto.result.OptimizeResult attribute*), 172
__ne__ (*pypesto.result.ProfileResult attribute*), 173
__ne__ (*pypesto.result.Result attribute*), 175
__ne__ (*pypesto.result.SampleResult attribute*), 176
__ne__ (*pypesto.result.Sequence attribute*), 178
__ne__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 144
__ne__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 146
__ne__ (*pypesto.sampling.InternalSampler attribute*), 148
__ne__ (*pypesto.sampling.McmcPtResult attribute*), 150
__ne__ (*pypesto.sampling.MetropolisSampler attribute*), 152
__ne__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 154
__ne__ (*pypesto.sampling.Sampler attribute*), 156
__ne__ (*pypesto.visualize.ReferencePoint attribute*), 160
__new__() (*pypesto.engine.Engine method*), 182
__new__() (*pypesto.engine.MultiProcessEngine method*), 183
__new__() (*pypesto.engine.MultiThreadEngine method*), 185
__new__() (*pypesto.engine.OptimizerTask method*), 186
__new__() (*pypesto.engine.SingleCoreEngine method*), 188
__new__() (*pypesto.objective.AggregatedObjective method*), 72
__new__() (*pypesto.objective.AmiciCalculator method*), 76
__new__() (*pypesto.objective.AmiciObjectBuilder method*), 77
__new__() (*pypesto.objective.AmiciObjective method*), 80
__new__() (*pypesto.objective.CsvHistory method*), 83
__new__() (*pypesto.objective.Hdf5History method*), 86
__new__() (*pypesto.objective.History method*), 88
__new__() (*pypesto.objective.HistoryBase method*), 90
__new__() (*pypesto.objective.HistoryOptions method*), 93
__new__() (*pypesto.objective.MemoryHistory*

method), 95
__new__() (*pypesto.objective.Objective method*), 99
__new__() (*pypesto.objective.OptimizerHistory method*), 102
__new__() (*pypesto.optimize.DlibOptimizer method*), 122
__new__() (*pypesto.optimize.OptimizeOptions method*), 124
__new__() (*pypesto.optimize.Optimizer method*), 125
__new__() (*pypesto.optimize.OptimizerResult method*), 128
__new__() (*pypesto.optimize.PyswarmOptimizer method*), 130
__new__() (*pypesto.optimize.ScipyOptimizer method*), 132
__new__() (*pypesto.petab.PetabImporter method*), 118
__new__() (*pypesto.problem.Iterable static method*), 106
__new__() (*pypesto.problem.List static method*), 108
__new__() (*pypesto.problem.Objective method*), 111
__new__() (*pypesto.problem.Problem method*), 115
__new__() (*pypesto.profile.ProfileOptions method*), 136
__new__() (*pypesto.profile.ProfilerResult method*), 140
__new__() (*pypesto.result.OptimizeResult method*), 172
__new__() (*pypesto.result.ProfileResult method*), 173
__new__() (*pypesto.result.Result method*), 175
__new__() (*pypesto.result.SampleResult method*), 176
__new__() (*pypesto.result.Sequence static method*), 178
__new__() (*pypesto.sampling.AdaptiveMetropolisSampler method*), 144
__new__() (*pypesto.sampling.AdaptiveParallelTemperingSampler method*), 146
__new__() (*pypesto.sampling.InternalSampler method*), 148
__new__() (*pypesto.sampling.McmcPtResult method*), 150
__new__() (*pypesto.sampling.MetropolisSampler method*), 152
__new__() (*pypesto.sampling.ParallelTemperingSampler method*), 154
__new__() (*pypesto.sampling.Sampler method*), 156
__new__() (*pypesto.visualize.ReferencePoint method*), 160
__next_in_mro__ (*pypesto.problem.Iterable attribute*), 106
__next_in_mro__ (*pypesto.problem.List attribute*), 108
__next_in_mro__ (*pypesto.result.Sequence attribute*), 178
__orig_bases__ (*pypesto.problem.Iterable attribute*), 106
__orig_bases__ (*pypesto.problem.List attribute*), 108
__orig_bases__ (*pypesto.result.Sequence attribute*), 178
__origin__ (*pypesto.problem.Iterable attribute*), 106
__origin__ (*pypesto.problem.List attribute*), 108
__origin__ (*pypesto.result.Sequence attribute*), 178
__parameters__ (*pypesto.problem.Iterable attribute*), 106
__parameters__ (*pypesto.problem.List attribute*), 108
__parameters__ (*pypesto.result.Sequence attribute*), 178
__reduce__() (*pypesto.engine.Engine method*), 182
__reduce__() (*pypesto.engine.MultiProcessEngine method*), 183
__reduce__() (*pypesto.engine.MultiThreadEngine method*), 185
__reduce__() (*pypesto.engine.OptimizerTask method*), 186
__reduce__() (*pypesto.engine.SingleCoreEngine method*), 188
__reduce__() (*pypesto.objective.AggregatedObjective method*), 72
__reduce__() (*pypesto.objective.AmiciCalculator method*), 76
__reduce__() (*pypesto.objective.AmiciObjectBuilder method*), 77
__reduce__() (*pypesto.objective.AmiciObjective method*), 80
__reduce__() (*pypesto.objective.CsvHistory method*), 83
__reduce__() (*pypesto.objective.Hdf5History method*), 86
__reduce__() (*pypesto.objective.History method*), 88
__reduce__() (*pypesto.objective.HistoryBase method*), 90
__reduce__() (*pypesto.objective.HistoryOptions method*), 93
__reduce__() (*pypesto.objective.MemoryHistory method*), 95
__reduce__() (*pypesto.objective.Objective method*), 99
__reduce__() (*pypesto.objective.OptimizerHistory method*), 102
__reduce__() (*pypesto.optimize.DlibOptimizer method*), 122
__reduce__() (*pypesto.optimize.OptimizeOptions method*), 124
__reduce__() (*pypesto.optimize.Optimizer method*), 126
__reduce__() (*pypesto.optimize.OptimizerResult method*), 128

__reduce__() (pypesto.optimize.PyswarmOptimizer method), 130
__reduce__() (pypesto.optimize.ScipyOptimizer method), 132
__reduce__() (pypesto.petab.PetabImporter method), 118
__reduce__() (pypesto.problem.Iterable method), 106
__reduce__() (pypesto.problem.List method), 108
__reduce__() (pypesto.problem.Objective method), 111
__reduce__() (pypesto.problem.Problem method), 115
__reduce__() (pypesto.profile.ProfileOptions method), 137
__reduce__() (pypesto.profile.ProfilerResult method), 140
__reduce__() (pypesto.result.OptimizeResult method), 172
__reduce__() (pypesto.result.ProfileResult method), 173
__reduce__() (pypesto.result.Result method), 175
__reduce__() (pypesto.result.SampleResult method), 177
__reduce__() (pypesto.result.Sequence method), 178
__reduce__() (pypesto.sampling.AdaptiveMetropolisSampler method), 144
__reduce__() (pypesto.sampling.AdaptiveParallelTemperingSampler method), 146
__reduce__() (pypesto.sampling.InternalSampler method), 148
__reduce__() (pypesto.sampling.McmcPtResult method), 150
__reduce__() (pypesto.sampling.MetropolisSampler method), 152
__reduce__() (pypesto.sampling.ParallelTemperingSampler method), 154
__reduce__() (pypesto.sampling.Sampler method), 156
__reduce__() (pypesto.visualize.ReferencePoint method), 160
__reduce_ex__() (pypesto.engine.Engine method), 182
__reduce_ex__() (pypesto.engine.MultiProcessEngine method), 183
__reduce_ex__() (pypesto.engine.MultiThreadEngine method), 185
__reduce_ex__() (pypesto.engine.OptimizerTask method), 186
__reduce_ex__() (pypesto.engine.SingleCoreEngine method), 188
__reduce_ex__() (pypesto.objective.AggregatedObjective method), 72
__reduce_ex__() (pypesto.objective.AmiciCalculator method), 76
__reduce_ex__() (pypesto.objective.AmiciObjectBuilder method), 77
__reduce_ex__() (pypesto.objective.AmiciObjective method), 80
__reduce_ex__() (pypesto.objective.CsvHistory method), 83
__reduce_ex__() (pypesto.objective.Hdf5History method), 86
__reduce_ex__() (pypesto.objective.History method), 88
__reduce_ex__() (pypesto.objective.HistoryBase method), 90
__reduce_ex__() (pypesto.objective.HistoryOptions method), 93
__reduce_ex__() (pypesto.objective.MemoryHistory method), 95
__reduce_ex__() (pypesto.objective.Objective method), 99
__reduce_ex__() (pypesto.objective.OptimizerHistory method), 102
__reduce_ex__() (pypesto.optimize.DlibOptimizer method), 122
__reduce_ex__() (pypesto.optimize.OptimizeOptions method), 124
__reduce_ex__() (pypesto.optimize.Optimizer method), 126
__reduce_ex__() (pypesto.optimize.OptimizerResult method), 128
__reduce_ex__() (pypesto.optimize.PyswarmOptimizer method), 130
__reduce_ex__() (pypesto.optimize.ScipyOptimizer method), 132
__reduce_ex__() (pypesto.petab.PetabImporter method), 118
__reduce_ex__() (pypesto.problem.Iterable method), 106
__reduce_ex__() (pypesto.problem.List method), 108
__reduce_ex__() (pypesto.problem.Objective method), 111
__reduce_ex__() (pypesto.problem.Problem method), 115
__reduce_ex__() (pypesto.profile.ProfileOptions method), 137
__reduce_ex__() (pypesto.profile.ProfilerResult method), 140
__reduce_ex__() (pypesto.result.OptimizeResult method), 172
__reduce_ex__() (pypesto.result.ProfileResult method), 173
__reduce_ex__() (pypesto.result.Result method), 175
__reduce_ex__() (pypesto.result.SampleResult

method), 177
__reduce_ex__() (*pypesto.result.Sequence method*), 178
__reduce_ex__() (*pypesto.sampling.AdaptiveMetropolisSampler method*), 144
__reduce_ex__() (*pypesto.sampling.AdaptiveParallelTemperingSampler method*), 146
__reduce_ex__() (*pypesto.sampling.InternalSampler method*), 148
__reduce_ex__() (*pypesto.sampling.McmcPtResult method*), 150
__reduce_ex__() (*pypesto.sampling.MetropolisSampler method*), 152
__reduce_ex__() (*pypesto.sampling.ParallelTemperingSampler method*), 154
__reduce_ex__() (*pypesto.sampling.Sampler method*), 156
__reduce_ex__() (*pypesto.visualize.ReferencePoint method*), 161
__repr__ (*pypesto.engine.Engine attribute*), 182
__repr__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__repr__ (*pypesto.engine.MultiThreadEngine attribute*), 185
__repr__ (*pypesto.engine.OptimizerTask attribute*), 186
__repr__ (*pypesto.engine.SingleCoreEngine attribute*), 188
__repr__ (*pypesto.objective.AggregatedObjective attribute*), 72
__repr__ (*pypesto.objective.AmiciCalculator attribute*), 76
__repr__ (*pypesto.objective.AmiciObjectBuilder attribute*), 77
__repr__ (*pypesto.objective.AmiciObjective attribute*), 80
__repr__ (*pypesto.objective.CsvHistory attribute*), 83
__repr__ (*pypesto.objective.Hdf5History attribute*), 86
__repr__ (*pypesto.objective.History attribute*), 88
__repr__ (*pypesto.objective.HistoryBase attribute*), 90
__repr__ (*pypesto.objective.HistoryOptions attribute*), 93
__repr__ (*pypesto.objective.MemoryHistory attribute*), 95
__repr__ (*pypesto.objective.Objective attribute*), 99
__repr__ (*pypesto.objective.OptimizerHistory attribute*), 102
__repr__ (*pypesto.optimize.DlibOptimizer attribute*), 122
__repr__ (*pypesto.optimize.OptimizeOptions attribute*), 124
__repr__ (*pypesto.optimize.Optimizer attribute*), 126
__repr__ (*pypesto.optimize.OptimizerResult attribute*), 129
__repr__ (*pypesto.optimize.PyswarmOptimizer attribute*), 130
__repr__ (*pypesto.optimize.ScipyOptimizer attribute*), 132
__repr__ (*pypesto.petab.PetabImporter attribute*), 118
__repr__ (*pypesto.problem.Iterable attribute*), 106
__repr__ (*pypesto.problem.List attribute*), 108
__repr__ (*pypesto.problem.Objective attribute*), 111
__repr__ (*pypesto.problem.Problem attribute*), 115
__repr__ (*pypesto.profile.ProfileOptions attribute*), 137
__repr__ (*pypesto.profile.ProfilerResult attribute*), 140
__repr__ (*pypesto.result.OptimizeResult attribute*), 172
__repr__ (*pypesto.result.ProfileResult attribute*), 173
__repr__ (*pypesto.result.Result attribute*), 175
__repr__ (*pypesto.result.SampleResult attribute*), 177
__repr__ (*pypesto.result.Sequence attribute*), 178
__repr__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 144
__repr__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 146
__repr__ (*pypesto.sampling.InternalSampler attribute*), 148
__repr__ (*pypesto.sampling.McmcPtResult attribute*), 150
__repr__ (*pypesto.sampling.MetropolisSampler attribute*), 152
__repr__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 154
__repr__ (*pypesto.sampling.Sampler attribute*), 156
__repr__ (*pypesto.visualize.ReferencePoint attribute*), 161
__reversed__() (*pypesto.problem.List method*), 108
__reversed__() (*pypesto.result.Sequence method*), 178
__rmul__ (*pypesto.problem.List attribute*), 108
__setattr__ (*pypesto.engine.Engine attribute*), 182
__setattr__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__setattr__ (*pypesto.engine.MultiThreadEngine attribute*), 185
__setattr__ (*pypesto.engine.OptimizerTask attribute*), 187
__setattr__ (*pypesto.engine.SingleCoreEngine attribute*), 188
__setattr__ (*pypesto.objective.AggregatedObjective attribute*), 72
__setattr__ (*pypesto.objective.AmiciCalculator attribute*), 76
__setattr__ (*pypesto.objective.AmiciObjectBuilder attribute*), 77
__setattr__ (*pypesto.objective.AmiciObjective attribute*), 80

__setattr__ (*pypesto.objective.CsvHistory attribute*), 83
__setattr__ (*pypesto.objective.Hdf5History attribute*), 86
__setattr__ (*pypesto.objective.History attribute*), 88
__setattr__ (*pypesto.objective.HistoryBase attribute*), 90
__setattr__ (*pypesto.objective.HistoryOptions attribute*), 93
__setattr__ (*pypesto.objective.MemoryHistory attribute*), 95
__setattr__ (*pypesto.objective.Objective attribute*), 99
__setattr__ (*pypesto.objective.OptimizerHistory attribute*), 102
__setattr__ (*pypesto.optimize.DlibOptimizer attribute*), 122
__setattr__ (*pypesto.optimize.OptimizeOptions attribute*), 124
__setattr__ (*pypesto.optimize.Optimizer attribute*), 126
__setattr__ (*pypesto.optimize.OptimizerResult attribute*), 129
__setattr__ (*pypesto.optimize.PyswarmOptimizer attribute*), 130
__setattr__ (*pypesto.optimize.ScipyOptimizer attribute*), 132
__setattr__ (*pypesto.petab.PetabImporter attribute*), 118
__setattr__ (*pypesto.problem.Iterable attribute*), 106
__setattr__ (*pypesto.problem.List attribute*), 108
__setattr__ (*pypesto.problem.Objective attribute*), 111
__setattr__ (*pypesto.problem.Problem attribute*), 115
__setattr__ (*pypesto.profile.ProfileOptions attribute*), 137
__setattr__ (*pypesto.profile.ProfilerResult attribute*), 140
__setattr__ (*pypesto.result.OptimizeResult attribute*), 172
__setattr__ (*pypesto.result.ProfileResult attribute*), 173
__setattr__ (*pypesto.result.Result attribute*), 175
__setattr__ (*pypesto.result.SampleResult attribute*), 177
__setattr__ (*pypesto.result.Sequence attribute*), 178
__setattr__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 144
__setattr__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 146
__setattr__ (*pypesto.sampling.InternalSampler attribute*), 148
__setattr__ (*pypesto.sampling.McmcPtResult attribute*), 150
__setattr__ (*pypesto.sampling.MetropolisSampler attribute*), 152
__setattr__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 154
__setattr__ (*pypesto.sampling.Sampler attribute*), 156
__setattr__ (*pypesto.visualize.ReferencePoint attribute*), 161
__setitem__ (*pypesto.objective.HistoryOptions attribute*), 93
__setitem__ (*pypesto.optimize.OptimizeOptions attribute*), 124
__setitem__ (*pypesto.optimize.OptimizerResult attribute*), 129
__setitem__ (*pypesto.problem.List attribute*), 108
__setitem__ (*pypesto.profile.ProfileOptions attribute*), 137
__setitem__ (*pypesto.profile.ProfilerResult attribute*), 140
__setitem__ (*pypesto.sampling.McmcPtResult attribute*), 151
__setitem__ (*pypesto.visualize.ReferencePoint attribute*), 161
__setstate__() (*pypesto.objective.AmiciObjective method*), 80
__sizeof__() (*pypesto.engine.Engine method*), 182
__sizeof__() (*pypesto.engine.MultiProcessEngine method*), 183
__sizeof__() (*pypesto.engine.MultiThreadEngine method*), 185
__sizeof__() (*pypesto.engine.OptimizerTask method*), 187
__sizeof__() (*pypesto.engine.SingleCoreEngine method*), 188
__sizeof__() (*pypesto.objective.AggregatedObjective method*), 72
__sizeof__() (*pypesto.objective.AmiciCalculator method*), 76
__sizeof__() (*pypesto.objective.AmiciObjectBuilder method*), 77
__sizeof__() (*pypesto.objective.AmiciObjective method*), 80
__sizeof__() (*pypesto.objective.CsvHistory method*), 83
__sizeof__() (*pypesto.objective.Hdf5History method*), 86
__sizeof__() (*pypesto.objective.History method*), 88
__sizeof__() (*pypesto.objective.HistoryBase method*), 90
__sizeof__() (*pypesto.objective.HistoryOptions method*), 94
__sizeof__() (*pypesto.objective.MemoryHistory*

method), 95
__sizeof__() (*pypesto.objective.Objective method*), 99
__sizeof__() (*pypesto.objective.OptimizerHistory method*), 102
__sizeof__() (*pypesto.optimize.DlibOptimizer method*), 122
__sizeof__() (*pypesto.optimize.OptimizeOptions method*), 124
__sizeof__() (*pypesto.optimize.Optimizer method*), 126
__sizeof__() (*pypesto.optimize.OptimizerResult method*), 129
__sizeof__() (*pypesto.optimize.PyswarmOptimizer method*), 130
__sizeof__() (*pypesto.optimize.ScipyOptimizer method*), 132
__sizeof__() (*pypesto.petab.PetabImporter method*), 118
__sizeof__() (*pypesto.problem.Iterable method*), 106
__sizeof__() (*pypesto.problem.List method*), 108
__sizeof__() (*pypesto.problem.Objective method*), 111
__sizeof__() (*pypesto.problem.Problem method*), 115
__sizeof__() (*pypesto.profile.ProfileOptions method*), 137
__sizeof__() (*pypesto.profile.ProfilerResult method*), 140
__sizeof__() (*pypesto.result.OptimizeResult method*), 172
__sizeof__() (*pypesto.result.ProfileResult method*), 174
__sizeof__() (*pypesto.result.Result method*), 175
__sizeof__() (*pypesto.result.SampleResult method*), 177
__sizeof__() (*pypesto.result.Sequence method*), 178
__sizeof__() (*pypesto.sampling.AdaptiveMetropolisSampler method*), 144
__sizeof__() (*pypesto.sampling.AdaptiveParallelTemperingSampler method*), 146
__sizeof__() (*pypesto.sampling.InternalSampler method*), 148
__sizeof__() (*pypesto.sampling.McmcPtResult method*), 151
__sizeof__() (*pypesto.sampling.MetropolisSampler method*), 152
__sizeof__() (*pypesto.sampling.ParallelTemperingSampler method*), 154
__sizeof__() (*pypesto.sampling.Sampler method*), 156
__sizeof__() (*pypesto.visualize.ReferencePoint method*), 161
__slots__ (*pypesto.problem.Iterable attribute*), 106
__slots__ (*pypesto.problem.List attribute*), 108
__slots__ (*pypesto.result.Sequence attribute*), 178
__str__ (*pypesto.engine.Engine attribute*), 182
__str__ (*pypesto.engine.MultiProcessEngine attribute*), 183
__str__ (*pypesto.engine.MultiThreadEngine attribute*), 185
__str__ (*pypesto.engine.OptimizerTask attribute*), 187
__str__ (*pypesto.engine.SingleCoreEngine attribute*), 188
__str__ (*pypesto.objective.AggregatedObjective attribute*), 72
__str__ (*pypesto.objective.AmiciCalculator attribute*), 76
__str__ (*pypesto.objective.AmiciObjectBuilder attribute*), 77
__str__ (*pypesto.objective.AmiciObjective attribute*), 80
__str__ (*pypesto.objective.CsvHistory attribute*), 83
__str__ (*pypesto.objective.Hdf5History attribute*), 86
__str__ (*pypesto.objective.History attribute*), 88
__str__ (*pypesto.objective.HistoryBase attribute*), 90
__str__ (*pypesto.objective.HistoryOptions attribute*), 94
__str__ (*pypesto.objective.MemoryHistory attribute*), 96
__str__ (*pypesto.objective.Objective attribute*), 99
__str__ (*pypesto.objective.OptimizerHistory attribute*), 102
__str__ (*pypesto.optimize.DlibOptimizer attribute*), 122
__str__ (*pypesto.optimize.OptimizeOptions attribute*), 124
__str__ (*pypesto.optimize.Optimizer attribute*), 126
__str__ (*pypesto.optimize.OptimizerResult attribute*), 129
__str__ (*pypesto.optimize.PyswarmOptimizer attribute*), 130
__str__ (*pypesto.optimize.ScipyOptimizer attribute*), 132
__str__ (*pypesto.petab.PetabImporter attribute*), 118
__str__ (*pypesto.problem.Iterable attribute*), 106
__str__ (*pypesto.problem.List attribute*), 108
__str__ (*pypesto.problem.Objective attribute*), 111
__str__ (*pypesto.problem.Problem attribute*), 115
__str__ (*pypesto.profile.ProfileOptions attribute*), 137
__str__ (*pypesto.profile.ProfilerResult attribute*), 140
__str__ (*pypesto.result.OptimizeResult attribute*), 172
__str__ (*pypesto.result.ProfileResult attribute*), 174
__str__ (*pypesto.result.Result attribute*), 175
__str__ (*pypesto.result.SampleResult attribute*), 177
__str__ (*pypesto.result.Sequence attribute*), 178
__str__ (*pypesto.sampling.AdaptiveMetropolisSampler*

attribute), 144
__str__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 146
__str__ (*pypesto.sampling.InternalSampler attribute*), 148
__str__ (*pypesto.sampling.McmcPtResult attribute*), 151
__str__ (*pypesto.sampling.MetropolisSampler attribute*), 152
__str__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 154
__str__ (*pypesto.sampling.Sampler attribute*), 156
__str__ (*pypesto.visualize.ReferencePoint attribute*), 161
__subclasshook__() (*pypesto.engine.Engine method*), 182
__subclasshook__() (*pypesto.engine.MultiProcessEngine method*), 184
__subclasshook__() (*pypesto.engine.MultiThreadEngine method*), 185
__subclasshook__() (*pypesto.engine.OptimizerTask method*), 187
__subclasshook__() (*pypesto.engine.SingleCoreEngine method*), 188
__subclasshook__() (*pypesto.objective.AggregatedObjective method*), 72
__subclasshook__() (*pypesto.objective.AmiciCalculator method*), 76
__subclasshook__() (*pypesto.objective.AmiciObjectBuilder method*), 77
__subclasshook__() (*pypesto.objective.AmiciObjective method*), 80
__subclasshook__() (*pypesto.objective.CsvHistory method*), 83
__subclasshook__() (*pypesto.objective.Hdf5History method*), 86
__subclasshook__() (*pypesto.objective.History method*), 88
__subclasshook__() (*pypesto.objective.HistoryBase method*), 90
__subclasshook__() (*pypesto.objective.HistoryOptions method*), 94
__subclasshook__() (*pypesto.objective.MemoryHistory method*), 96
__subclasshook__() (*pypesto.objective.Objective method*), 99
__subclasshook__() (*pypesto.objective.OptimizerHistory method*), 102
__subclasshook__() (*pypesto.optimize.DlibOptimizer method*), 122
__subclasshook__() (*pypesto.optimize.OptimizeOptions method*), 124
__subclasshook__() (*pypesto.optimize.Optimizer method*), 126
__subclasshook__() (*pypesto.optimize.OptimizerResult method*), 129
__subclasshook__() (*pypesto.optimize.PyswarmOptimizer method*), 130
__subclasshook__() (*pypesto.optimize.ScipyOptimizer method*), 132
__subclasshook__() (*pypesto.petab.PetabImporter method*), 118
__subclasshook__() (*pypesto.problem.Iterable method*), 106
__subclasshook__() (*pypesto.problem.List method*), 108
__subclasshook__() (*pypesto.problem.Objective method*), 111
__subclasshook__() (*pypesto.problem.Problem method*), 115
__subclasshook__() (*pypesto.profile.ProfileOptions method*), 137
__subclasshook__() (*pypesto.profile.ProfilerResult method*), 140
__subclasshook__() (*pypesto.result.OptimizeResult method*), 172
__subclasshook__() (*pypesto.result.ProfileResult method*), 174
__subclasshook__() (*pypesto.result.Result method*), 176
__subclasshook__() (*pypesto.result.SampleResult method*), 177
__subclasshook__() (*pypesto.result.Sequence method*), 178
__subclasshook__() (*pypesto.sampling.AdaptiveMetropolisSampler method*), 144
__subclasshook__() (*pypesto.sampling.AdaptiveParallelTemperingSampler method*), 146
__subclasshook__()

(*pypesto.sampling.InternalSampler method*), 148
__subclasshook__() (*pypesto.sampling.McmcPtResult method*), 151
__subclasshook__() (*pypesto.sampling.MetropolisSampler method*), 152
__subclasshook__() (*pypesto.sampling.ParallelTemperingSampler method*), 154
__subclasshook__() (*pypesto.sampling.Sampler method*), 156
__subclasshook__() (*pypesto.visualize.ReferencePoint method*), 161
__tree_hash__ (*pypesto.problem.Iterable attribute*), 106
__tree_hash__ (*pypesto.problem.List attribute*), 108
__tree_hash__ (*pypesto.result.Sequence attribute*), 178
__weakref__ (*pypesto.engine.Engine attribute*), 182
__weakref__ (*pypesto.engine.MultiProcessEngine attribute*), 184
__weakref__ (*pypesto.engine.MultiThreadEngine attribute*), 185
__weakref__ (*pypesto.engine.OptimizerTask attribute*), 187
__weakref__ (*pypesto.engine.SingleCoreEngine attribute*), 188
__weakref__ (*pypesto.objective.AggregatedObjective attribute*), 73
__weakref__ (*pypesto.objective.AmiciCalculator attribute*), 76
__weakref__ (*pypesto.objective.AmiciObjectBuilder attribute*), 77
__weakref__ (*pypesto.objective.AmiciObjective attribute*), 80
__weakref__ (*pypesto.objective.CsvHistory attribute*), 84
__weakref__ (*pypesto.objective.Hdf5History attribute*), 86
__weakref__ (*pypesto.objective.History attribute*), 88
__weakref__ (*pypesto.objective.HistoryBase attribute*), 91
__weakref__ (*pypesto.objective.HistoryOptions attribute*), 94
__weakref__ (*pypesto.objective.MemoryHistory attribute*), 96
__weakref__ (*pypesto.objective.Objective attribute*), 100
__weakref__ (*pypesto.objective.OptimizerHistory attribute*), 102
__weakref__ (*pypesto.optimize.DlibOptimizer attribute*), 122
__weakref__ (*pypesto.optimize.OptimizeOptions attribute*), 124
__weakref__ (*pypesto.optimize.Optimizer attribute*), 126
__weakref__ (*pypesto.optimize.OptimizerResult attribute*), 129
__weakref__ (*pypesto.optimize.PyswarmOptimizer attribute*), 131
__weakref__ (*pypesto.optimize.ScipyOptimizer attribute*), 132
__weakref__ (*pypesto.petab.PetabImporter attribute*), 118
__weakref__ (*pypesto.problem.Objective attribute*), 112
__weakref__ (*pypesto.problem.Problem attribute*), 115
__weakref__ (*pypesto.profile.ProfileOptions attribute*), 137
__weakref__ (*pypesto.profile.ProfilerResult attribute*), 140
__weakref__ (*pypesto.result.OptimizeResult attribute*), 172
__weakref__ (*pypesto.result.ProfileResult attribute*), 174
__weakref__ (*pypesto.result.Result attribute*), 176
__weakref__ (*pypesto.result.SampleResult attribute*), 177
__weakref__ (*pypesto.sampling.AdaptiveMetropolisSampler attribute*), 144
__weakref__ (*pypesto.sampling.AdaptiveParallelTemperingSampler attribute*), 146
__weakref__ (*pypesto.sampling.InternalSampler attribute*), 148
__weakref__ (*pypesto.sampling.McmcPtResult attribute*), 151
__weakref__ (*pypesto.sampling.MetropolisSampler attribute*), 152
__weakref__ (*pypesto.sampling.ParallelTemperingSampler attribute*), 155
__weakref__ (*pypesto.sampling.Sampler attribute*), 156
__weakref__ (*pypesto.visualize.ReferencePoint attribute*), 161

A

AdaptiveMetropolisSampler (*class in pypesto.sampling*), 143
AdaptiveParallelTemperingSampler (*class in pypesto.sampling*), 145
add_profile() (*pypesto.result.ProfileResult method*), 174
adjust_betas() (*pypesto.sampling.AdaptiveParallelTemperingSampler method*), 146

adjust_betas() (*pypesto.sampling.ParallelTemperingSampler method*), 155
aggregate_fun() (*pypesto.objective.AggregatedObjective method*), 73
aggregate_fun_sensi_orders() (*pypesto.objective.AggregatedObjective method*), 73
aggregate_grad() (*pypesto.objective.AggregatedObjective method*), 73
aggregate_hess() (*pypesto.objective.AggregatedObjective method*), 73
aggregate_hessp() (*pypesto.objective.AggregatedObjective method*), 73
aggregate_res() (*pypesto.objective.AggregatedObjective method*), 73
aggregate_res_sensi_orders() (*pypesto.objective.AggregatedObjective method*), 73
aggregate_sres() (*pypesto.objective.AggregatedObjective method*), 73
AggregatedObjective (*class in pypesto.objective*), 71
AmiciCalculator (*class in pypesto.objective*), 74
AmiciObjectBuilder (*class in pypesto.objective*), 76
AmiciObjective (*class in pypesto.objective*), 78
append() (*pypesto.problem.List method*), 108
append() (*pypesto.result.OptimizeResult method*), 172
append_profile_point() (*pypesto.profile.ProfilerResult method*), 140
apply_steadystate_guess() (*pypesto.objective.AmiciObjective method*), 80
as_dataframe() (*pypesto.result.OptimizeResult method*), 172
as_list() (*pypesto.result.OptimizeResult method*), 172
assert_instance() (*pypesto.objective.HistoryOptions static method*), 94
assert_instance() (*pypesto.optimize.OptimizeOptions static method*), 124
assign_clustered_colors() (*in module pypesto.visualize*), 161
assign_clusters() (*in module pypesto.visualize*), 162
assign_colors() (*in module pypesto.visualize*), 162
assign_startpoints() (*in module pypesto.startpoint*), 189
auto_color (*pypesto.visualize.ReferencePoint attribute*), 159

C

check_grad() (*pypesto.objective.AggregatedObjective method*), 73
check_grad() (*pypesto.objective.AmiciObjective method*), 80
check_grad() (*pypesto.objective.Objective method*), 100
check_grad() (*pypesto.problem.Objective method*), 112
check_sensi_orders() (*pypesto.objective.AggregatedObjective method*), 73
check_sensi_orders() (*pypesto.objective.AmiciObjective method*), 81
check_sensi_orders() (*pypesto.objective.Objective method*), 100
check_sensi_orders() (*pypesto.problem.Objective method*), 112
clear() (*pypesto.objective.HistoryOptions method*), 94
clear() (*pypesto.optimize.OptimizeOptions method*), 124
clear() (*pypesto.optimize.OptimizerResult method*), 129
clear() (*pypesto.problem.List method*), 108
clear() (*pypesto.profile.ProfileOptions method*), 137
clear() (*pypesto.profile.ProfilerResult method*), 140
clear() (*pypesto.sampling.McmcPtResult method*), 151
clear() (*pypesto.visualize.ReferencePoint method*), 161
color (*pypesto.visualize.ReferencePoint attribute*), 159
compile_model() (*pypesto.petab.PetabImporter method*), 118
copy() (*pypesto.objective.HistoryOptions method*), 94
copy() (*pypesto.optimize.OptimizeOptions method*), 124
copy() (*pypesto.optimize.OptimizerResult method*), 129
copy() (*pypesto.problem.List method*), 108
copy() (*pypesto.profile.ProfileOptions method*), 137
copy() (*pypesto.profile.ProfilerResult method*), 140
copy() (*pypesto.sampling.McmcPtResult method*), 151
copy() (*pypesto.visualize.ReferencePoint method*), 161
count() (*pypesto.problem.List method*), 108
count() (*pypesto.result.Sequence method*), 178
create_edatas() (*pypesto.objective.AmiciObjectBuilder method*), 77
create_edatas() (*pypesto.petab.PetabImporter method*), 118
create_history() (*pypesto.objective.HistoryOptions method*), 94
create_instance() (*pypesto.profile.ProfileOptions static method*), 137

create_model() (*pypesto.objective.AmiciObjectBuilder method*), 78
create_model() (*pypesto.petab.PetabImporter method*), 118
create_new_profile() (*pypesto.result.ProfileResult method*), 174
create_new_profile_list() (*pypesto.result.ProfileResult method*), 174
create_objective() (*pypesto.petab.PetabImporter method*), 119
create_problem() (*pypesto.petab.PetabImporter method*), 119
create_references() (*in module pypesto.visualize*), 162
create_solver() (*pypesto.objective.AmiciObjectBuilder method*), 78
create_solver() (*pypesto.petab.PetabImporter method*), 119
CsvHistory (*class in pypesto.objective*), 82

D

default_options() (*pypesto.sampling.AdaptiveMetropolisSampler class method*), 144
default_options() (*pypesto.sampling.AdaptiveParallelTemperingSampler class method*), 146
default_options() (*pypesto.sampling.InternalSampler class method*), 148
default_options() (*pypesto.sampling.MetropolisSampler class method*), 153
default_options() (*pypesto.sampling.ParallelTemperingSampler class method*), 155
default_options() (*pypesto.sampling.Sampler class method*), 157
delete_nan_inf() (*in module pypesto.visualize*), 162
dim (*pypesto.problem.Problem attribute*), 114
DlibOptimizer (*class in pypesto.optimize*), 121

E

Engine (*class in pypesto.engine*), 181
execute() (*pypesto.engine.Engine method*), 182
execute() (*pypesto.engine.MultiProcessEngine method*), 184
execute() (*pypesto.engine.MultiThreadEngine method*), 185
execute() (*pypesto.engine.OptimizerTask method*), 187
execute() (*pypesto.engine.SingleCoreEngine method*), 188
exitflag (*pypesto.optimize.OptimizerResult attribute*), 127
exitflag_path (*pypesto.profile.ProfilerResult attribute*), 138
extend() (*pypesto.problem.List method*), 108

F

finalize() (*pypesto.objective.CsvHistory method*), 84
finalize() (*pypesto.objective.Hdf5History method*), 86
finalize() (*pypesto.objective.History method*), 88
finalize() (*pypesto.objective.HistoryBase method*), 91
finalize() (*pypesto.objective.MemoryHistory method*), 96
finalize() (*pypesto.objective.OptimizerHistory method*), 102
fix_parameters() (*pypesto.problem.Problem method*), 115
flip_profile() (*pypesto.profile.ProfilerResult method*), 140
from_yaml() (*pypesto.petab.PetabImporter static method*), 119
fromkeys() (*pypesto.objective.HistoryOptions method*), 94
fromkeys() (*pypesto.optimize.OptimizeOptions method*), 124
fromkeys() (*pypesto.optimize.OptimizerResult method*), 129
fromkeys() (*pypesto.profile.ProfileOptions method*), 137
fromkeys() (*pypesto.profile.ProfilerResult method*), 140
fromkeys() (*pypesto.sampling.McmcPtResult method*), 151
fromkeys() (*pypesto.visualize.ReferencePoint method*), 161
fval (*pypesto.optimize.OptimizerResult attribute*), 126
fval (*pypesto.visualize.ReferencePoint attribute*), 159
fval0 (*pypesto.optimize.OptimizerResult attribute*), 127
fval_path (*pypesto.profile.ProfilerResult attribute*), 138

G

get() (*pypesto.objective.HistoryOptions method*), 94
get() (*pypesto.optimize.OptimizeOptions method*), 124
get() (*pypesto.optimize.OptimizerResult method*), 129
get() (*pypesto.profile.ProfileOptions method*), 137
get() (*pypesto.profile.ProfilerResult method*), 140
get() (*pypesto.sampling.McmcPtResult method*), 151
get() (*pypesto.visualize.ReferencePoint method*), 161
get_bound_fun() (*pypesto.objective.AmiciObjective method*), 81

get_bound_res() (*pypesto.objective.AmiciObjective method*), 81
get_chi2_trace() (*pypesto.objective.CsvHistory method*), 84
get_chi2_trace() (*pypesto.objective.Hdf5History method*), 86
get_chi2_trace() (*pypesto.objective.History method*), 88
get_chi2_trace() (*pypesto.objective.HistoryBase method*), 91
get_chi2_trace() (*pypesto.objective.MemoryHistory method*), 96
get_chi2_trace() (*pypesto.objective.OptimizerHistory method*), 103
get_current_profile() (*pypesto.result.ProfileResult method*), 174
get_default_options() (*pypesto.optimize.DlibOptimizer static method*), 122
get_default_options() (*pypesto.optimize.Optimizer static method*), 126
get_default_options() (*pypesto.optimize.PyswarmOptimizer static method*), 131
get_default_options() (*pypesto.optimize.ScipyOptimizer static method*), 132
get_for_key() (*pypesto.result.OptimizeResult method*), 172
get_full_matrix() (*pypesto.problem.Problem method*), 115
get_full_vector() (*pypesto.problem.Problem method*), 115
get_fval() (*pypesto.objective.AggregatedObjective method*), 73
get_fval() (*pypesto.objective.AmiciObjective method*), 81
get_fval() (*pypesto.objective.Objective method*), 100
get_fval() (*pypesto.problem.Objective method*), 112
get_fval_trace() (*pypesto.objective.CsvHistory method*), 84
get_fval_trace() (*pypesto.objective.Hdf5History method*), 86
get_fval_trace() (*pypesto.objective.History method*), 88
get_fval_trace() (*pypesto.objective.HistoryBase method*), 91
get_fval_trace() (*pypesto.objective.MemoryHistory method*), 96
get_fval_trace() (*pypesto.objective.OptimizerHistory method*), 103
get_grad() (*pypesto.objective.AggregatedObjective method*), 73
get_grad() (*pypesto.objective.AmiciObjective method*), 81
get_grad() (*pypesto.objective.Objective method*), 100
get_grad() (*pypesto.problem.Objective method*), 112
get_grad_trace() (*pypesto.objective.CsvHistory method*), 84
get_grad_trace() (*pypesto.objective.Hdf5History method*), 86
get_grad_trace() (*pypesto.objective.History method*), 88
get_grad_trace() (*pypesto.objective.HistoryBase method*), 91
get_grad_trace() (*pypesto.objective.MemoryHistory method*), 96
get_grad_trace() (*pypesto.objective.OptimizerHistory method*), 103
get_hess() (*pypesto.objective.AggregatedObjective method*), 73
get_hess() (*pypesto.objective.AmiciObjective method*), 81
get_hess() (*pypesto.objective.Objective method*), 100
get_hess() (*pypesto.problem.Objective method*), 112
get_hess_trace() (*pypesto.objective.CsvHistory method*), 84
get_hess_trace() (*pypesto.objective.Hdf5History method*), 86
get_hess_trace() (*pypesto.objective.History method*), 89
get_hess_trace() (*pypesto.objective.HistoryBase method*), 91
get_hess_trace() (*pypesto.objective.MemoryHistory method*), 96
get_hess_trace() (*pypesto.objective.OptimizerHistory method*), 103
get_last_sample() (*pypesto.sampling.AdaptiveMetropolisSampler method*), 144
get_last_sample() (*pypesto.sampling.InternalSampler method*), 148
get_last_sample() (*pypesto.sampling.MetropolisSampler method*), 153
get_reduced_matrix() (*pypesto.problem.Problem method*), 115
get_reduced_vector() (*pypesto.problem.Problem method*), 116
get_res() (*pypesto.objective.AggregatedObjective method*), 73
get_res() (*pypesto.objective.AmiciObjective method*), 81
get_res() (*pypesto.objective.Objective method*), 100
get_res() (*pypesto.problem.Objective method*), 112
get_res_trace() (*pypesto.objective.CsvHistory*

method), 84
get_res_trace() (*pypesto.objective.Hdf5History method*), 86
get_res_trace() (*pypesto.objective.History method*), 89
get_res_trace() (*pypesto.objective.HistoryBase method*), 91
get_res_trace() (*pypesto.objective.MemoryHistory method*), 96
get_res_trace() (*pypesto.objective.OptimizerHistory method*), 103
get_samples() (*pypesto.sampling.AdaptiveMetropolisSampler method*), 144
get_samples() (*pypesto.sampling.AdaptiveParallelTemperingSampler method*), 147
get_samples() (*pypesto.sampling.InternalSampler method*), 149
get_samples() (*pypesto.sampling.MetropolisSampler method*), 153
get_samples() (*pypesto.sampling.ParallelTemperingSampler method*), 155
get_samples() (*pypesto.sampling.Sampler method*), 157
get_schi2_trace() (*pypesto.objective.CsvHistory method*), 84
get_schi2_trace() (*pypesto.objective.Hdf5History method*), 86
get_schi2_trace() (*pypesto.objective.History method*), 89
get_schi2_trace() (*pypesto.objective.HistoryBase method*), 91
get_schi2_trace() (*pypesto.objective.MemoryHistory method*), 96
get_schi2_trace() (*pypesto.objective.OptimizerHistory method*), 103
get_sres() (*pypesto.objective.AggregatedObjective method*), 74
get_sres() (*pypesto.objective.AmiciObjective method*), 81
get_sres() (*pypesto.objective.Objective method*), 100
get_sres() (*pypesto.problem.Objective method*), 112
get_sres_trace() (*pypesto.objective.CsvHistory method*), 84
get_sres_trace() (*pypesto.objective.Hdf5History method*), 86
get_sres_trace() (*pypesto.objective.History method*), 89
get_sres_trace() (*pypesto.objective.HistoryBase method*), 91
get_sres_trace() (*pypesto.objective.MemoryHistory method*), 96
get_sres_trace() (*pypesto.objective.OptimizerHistory method*), 103
get_time_trace() (*pypesto.objective.CsvHistory method*), 84
get_time_trace() (*pypesto.objective.Hdf5History method*), 86
get_time_trace() (*pypesto.objective.History method*), 89
get_time_trace() (*pypesto.objective.HistoryBase method*), 91
get_time_trace() (*pypesto.objective.MemoryHistory method*), 96
get_time_trace() (*pypesto.objective.OptimizerHistory method*), 103
get_x_trace() (*pypesto.objective.CsvHistory method*), 84
get_x_trace() (*pypesto.objective.Hdf5History method*), 86
get_x_trace() (*pypesto.objective.History method*), 89
get_x_trace() (*pypesto.objective.HistoryBase method*), 91
get_x_trace() (*pypesto.objective.MemoryHistory method*), 96
get_x_trace() (*pypesto.objective.OptimizerHistory method*), 103
grad (*pypesto.optimize.OptimizerResult attribute*), 126
gradnorm_path (*pypesto.profile.ProfilerResult attribute*), 138

H

has_fun (*pypesto.objective.AggregatedObjective attribute*), 74
has_fun (*pypesto.objective.AmiciObjective attribute*), 81
has_fun (*pypesto.objective.Objective attribute*), 100
has_fun (*pypesto.problem.Objective attribute*), 112
has_grad (*pypesto.objective.AggregatedObjective attribute*), 74
has_grad (*pypesto.objective.AmiciObjective attribute*), 81
has_grad (*pypesto.objective.Objective attribute*), 100
has_grad (*pypesto.problem.Objective attribute*), 112
has_hess (*pypesto.objective.AggregatedObjective attribute*), 74
has_hess (*pypesto.objective.AmiciObjective attribute*), 81
has_hess (*pypesto.objective.Objective attribute*), 100
has_hess (*pypesto.problem.Objective attribute*), 112
has_hessp (*pypesto.objective.AggregatedObjective attribute*), 74
has_hessp (*pypesto.objective.AmiciObjective attribute*), 81
has_hessp (*pypesto.objective.Objective attribute*), 100
has_hessp (*pypesto.problem.Objective attribute*), 112

has_res (*pypesto.objective.AggregatedObjective attribute*), 74
has_res (*pypesto.objective.AmiciObjective attribute*), 81
has_res (*pypesto.objective.Objective attribute*), 100
has_res (*pypesto.problem.Objective attribute*), 112
has_sres (*pypesto.objective.AggregatedObjective attribute*), 74
has_sres (*pypesto.objective.AmiciObjective attribute*), 81
has_sres (*pypesto.objective.Objective attribute*), 100
has_sres (*pypesto.problem.Objective attribute*), 112
Hdf5History (*class in pypesto.objective*), 85
hess (*pypesto.optimize.OptimizerResult attribute*), 127
History (*class in pypesto.objective*), 87
history (*pypesto.objective.Objective attribute*), 98
history (*pypesto.optimize.OptimizerResult attribute*), 127
history (*pypesto.problem.Objective attribute*), 110
HistoryBase (*class in pypesto.objective*), 89
HistoryOptions (*class in pypesto.objective*), 92

I

id (*pypesto.optimize.OptimizerResult attribute*), 126
index() (*pypesto.problem.List method*), 108
index() (*pypesto.result.Sequence method*), 179
initialize() (*pypesto.objective.AggregatedObjective method*), 74
initialize() (*pypesto.objective.AmiciCalculator method*), 76
initialize() (*pypesto.objective.AmiciObjective method*), 81
initialize() (*pypesto.objective.Objective method*), 101
initialize() (*pypesto.problem.Objective method*), 112
initialize() (*pypesto.sampling.AdaptiveMetropolisSampler method*), 145
initialize() (*pypesto.sampling.AdaptiveParallelTemperingSampler method*), 147
initialize() (*pypesto.sampling.InternalSampler method*), 149
initialize() (*pypesto.sampling.MetropolisSampler method*), 153
initialize() (*pypesto.sampling.ParallelTemperingSampler method*), 155
initialize() (*pypesto.sampling.Sampler method*), 157
insert() (*pypesto.problem.List method*), 108
InternalSampler (*class in pypesto.sampling*), 147
is_least_squares() (*pypesto.optimize.DlibOptimizer method*), 122
is_least_squares() (*pypesto.optimize.Optimizer method*), 126
is_least_squares() (*pypesto.optimize.PyswarmOptimizer method*), 131
is_least_squares() (*pypesto.optimize.ScipyOptimizer method*), 132
items() (*pypesto.objective.HistoryOptions method*), 94
items() (*pypesto.optimize.OptimizeOptions method*), 124
items() (*pypesto.optimize.OptimizerResult method*), 129
items() (*pypesto.profile.ProfileOptions method*), 137
items() (*pypesto.profile.ProfilerResult method*), 140
items() (*pypesto.sampling.McmcPtResult method*), 151
items() (*pypesto.visualize.ReferencePoint method*), 161
Iterable (*class in pypesto.problem*), 105

K

keys() (*pypesto.objective.HistoryOptions method*), 94
keys() (*pypesto.optimize.OptimizeOptions method*), 124
keys() (*pypesto.optimize.OptimizerResult method*), 129
keys() (*pypesto.profile.ProfileOptions method*), 137
keys() (*pypesto.profile.ProfilerResult method*), 140
keys() (*pypesto.sampling.McmcPtResult method*), 151
keys() (*pypesto.visualize.ReferencePoint method*), 161

L

latin_hypercube() (*in module pypesto.startpoint*), 189
legend (*pypesto.visualize.ReferencePoint attribute*), 159
List (*class in pypesto.problem*), 106
log_to_console() (*in module pypesto.logging*), 191
log_to_file() (*in module pypesto.logging*), 191

M

McmcPtResult (*class in pypesto.sampling*), 149
MemoryHistory (*class in pypesto.objective*), 94
message (*pypesto.optimize.OptimizerResult attribute*), 127
message (*pypesto.profile.ProfilerResult attribute*), 138
MetropolisSampler (*class in pypesto.sampling*), 151
minimize() (*in module pypesto.optimize*), 132
minimize() (*pypesto.optimize.DlibOptimizer method*), 122
minimize() (*pypesto.optimize.Optimizer method*), 126

minimize() (*pypesto.optimize.PyswarmOptimizer method*), 131
minimize() (*pypesto.optimize.ScipyOptimizer method*), 132
MODEL_BASE_DIR (*pypesto.petab.PetabImporter attribute*), 117
MultiProcessEngine (*class in pypesto.engine*), 182
MultiThreadEngine (*class in pypesto.engine*), 184

N

n_fval (*pypesto.objective.CsvHistory attribute*), 84
n_fval (*pypesto.objective.Hdf5History attribute*), 86
n_fval (*pypesto.objective.History attribute*), 89
n_fval (*pypesto.objective.HistoryBase attribute*), 91
n_fval (*pypesto.objective.MemoryHistory attribute*), 96
n_fval (*pypesto.objective.OptimizerHistory attribute*), 103
n_fval (*pypesto.optimize.OptimizerResult attribute*), 127
n_fval (*pypesto.profile.ProfilerResult attribute*), 138
n_grad (*pypesto.objective.CsvHistory attribute*), 84
n_grad (*pypesto.objective.Hdf5History attribute*), 87
n_grad (*pypesto.objective.History attribute*), 89
n_grad (*pypesto.objective.HistoryBase attribute*), 91
n_grad (*pypesto.objective.MemoryHistory attribute*), 96
n_grad (*pypesto.objective.OptimizerHistory attribute*), 103
n_grad (*pypesto.optimize.OptimizerResult attribute*), 127
n_grad (*pypesto.profile.ProfilerResult attribute*), 138
n_hess (*pypesto.objective.CsvHistory attribute*), 84
n_hess (*pypesto.objective.Hdf5History attribute*), 87
n_hess (*pypesto.objective.History attribute*), 89
n_hess (*pypesto.objective.HistoryBase attribute*), 91
n_hess (*pypesto.objective.MemoryHistory attribute*), 96
n_hess (*pypesto.objective.OptimizerHistory attribute*), 103
n_hess (*pypesto.optimize.OptimizerResult attribute*), 127
n_hess (*pypesto.profile.ProfilerResult attribute*), 138
n_res (*pypesto.objective.CsvHistory attribute*), 84
n_res (*pypesto.objective.Hdf5History attribute*), 87
n_res (*pypesto.objective.History attribute*), 89
n_res (*pypesto.objective.HistoryBase attribute*), 91
n_res (*pypesto.objective.MemoryHistory attribute*), 96
n_res (*pypesto.objective.OptimizerHistory attribute*), 103
n_res (*pypesto.optimize.OptimizerResult attribute*), 127
n_sres (*pypesto.objective.CsvHistory attribute*), 84
n_sres (*pypesto.objective.Hdf5History attribute*), 87
n_sres (*pypesto.objective.History attribute*), 89
n_sres (*pypesto.objective.HistoryBase attribute*), 91
n_sres (*pypesto.objective.MemoryHistory attribute*), 96
n_sres (*pypesto.objective.OptimizerHistory attribute*), 103
n_sres (*pypesto.optimize.OptimizerResult attribute*), 127
normalize_input() (*pypesto.problem.Problem method*), 116

O

Objective (*class in pypesto.objective*), 97
Objective (*class in pypesto.problem*), 109
optimize_result (*pypesto.result.Result attribute*), 174
OptimizeOptions (*class in pypesto.optimize*), 122
Optimizer (*class in pypesto.optimize*), 125
optimizer_history() (*in module pypesto.visualize*), 163
optimizer_history_lowlevel() (*in module pypesto.visualize*), 163
OptimizeResult (*class in pypesto.result*), 171
OptimizerHistory (*class in pypesto.objective*), 101
OptimizerResult (*class in pypesto.optimize*), 126
OptimizerTask (*class in pypesto.engine*), 185
output_to_dict() (*pypesto.objective.AggregatedObjective static method*), 74
output_to_dict() (*pypesto.objective.AmiciObjective static method*), 81
output_to_dict() (*pypesto.objective.Objective static method*), 101
output_to_dict() (*pypesto.problem.Objective static method*), 113
output_to_tuple() (*pypesto.objective.AggregatedObjective static method*), 74
output_to_tuple() (*pypesto.objective.AmiciObjective static method*), 81
output_to_tuple() (*pypesto.objective.Objective static method*), 101
output_to_tuple() (*pypesto.problem.Objective static method*), 113

P

par_arr_to_dct() (*pypesto.objective.AmiciObjective method*), 82
ParallelTemperingSampler (*class in pypesto.sampling*), 153
parameter_profile() (*in module pypesto.profile*), 141
parameters() (*in module pypesto.visualize*), 164
parameters_lowlevel() (*in module pypesto.visualize*), 164

PetabImporter (*class in pypesto.petab*), 117
pop() (*pypesto.objective.HistoryOptions method*), 94
pop() (*pypesto.optimize.OptimizeOptions method*), 124
pop() (*pypesto.optimize.OptimizerResult method*), 129
pop() (*pypesto.problem.List method*), 108
pop() (*pypesto.profile.ProfileOptions method*), 137
pop() (*pypesto.profile.ProfilerResult method*), 140
pop() (*pypesto.sampling.McmcPtResult method*), 151
pop() (*pypesto.visualize.ReferencePoint method*), 161
popitem() (*pypesto.objective.HistoryOptions method*), 94
popitem() (*pypesto.optimize.OptimizeOptions method*), 124
popitem() (*pypesto.optimize.OptimizerResult method*), 129
popitem() (*pypesto.profile.ProfileOptions method*), 137
popitem() (*pypesto.profile.ProfilerResult method*), 140
popitem() (*pypesto.sampling.McmcPtResult method*), 151
popitem() (*pypesto.visualize.ReferencePoint method*), 161
pre_post_processor (*pypesto.objective.Objective attribute*), 98
pre_post_processor (*pypesto.problem.Objective attribute*), 110
print_parameter_summary() (*pypesto.problem.Problem method*), 116
Problem (*class in pypesto.problem*), 113
problem (*pypesto.result.Result attribute*), 174
process_offset_y() (*in module pypesto.visualize*), 165
process_result_list() (*in module pypesto.visualize*), 165
process_y_limits() (*in module pypesto.visualize*), 165
profile_lowlevel() (*in module pypesto.visualize*), 166
profile_result (*pypesto.result.Result attribute*), 174
ProfileOptions (*class in pypesto.profile*), 135
ProfileResult (*class in pypesto.result*), 172
ProfilerResult (*class in pypesto.profile*), 137
profiles() (*in module pypesto.visualize*), 166
profiles_lowlevel() (*in module pypesto.visualize*), 166
pypesto.engine (*module*), 179
pypesto.logging (*module*), 189
pypesto.objective (*module*), 70
pypesto.optimize (*module*), 120
pypesto.petab (*module*), 116
pypesto.problem (*module*), 103
pypesto.profile (*module*), 133
pypesto.result (*module*), 169
pypesto.sampling (*module*), 141
pypesto.startpoint (*module*), 188
pypesto.visualize (*module*), 157
PyswarmOptimizer (*class in pypesto.optimize*), 129

R

ratio_path (*pypesto.profile.ProfilerResult attribute*), 138
rdatas_to_measurement_df() (*pypesto.petab.PetabImporter method*), 119
rdatas_to_simulation_df() (*pypesto.petab.PetabImporter method*), 120
rebind_fun() (*pypesto.objective.AmiciObjective method*), 82
rebind_res() (*pypesto.objective.AmiciObjective method*), 82
ReferencePoint (*class in pypesto.visualize*), 159
remove() (*pypesto.problem.List method*), 109
res (*pypesto.optimize.OptimizerResult attribute*), 127
res_to_chi2() (*in module pypesto.objective*), 103
reset_steadystate_guesses() (*pypesto.objective.AggregatedObjective method*), 74
reset_steadystate_guesses() (*pypesto.objective.AmiciObjective method*), 82
Result (*class in pypesto.result*), 174
reverse() (*pypesto.problem.List method*), 109

S

sample() (*in module pypesto.sampling*), 157
sample() (*pypesto.sampling.AdaptiveMetropolisSampler method*), 145
sample() (*pypesto.sampling.AdaptiveParallelTemperingSampler method*), 147
sample() (*pypesto.sampling.InternalSampler method*), 149
sample() (*pypesto.sampling.MetropolisSampler method*), 153
sample() (*pypesto.sampling.ParallelTemperingSampler method*), 155
sample() (*pypesto.sampling.Sampler method*), 157
sample_result (*pypesto.result.Result attribute*), 174
Sampler (*class in pypesto.sampling*), 155
SampleResult (*class in pypesto.result*), 176
sampling_1d_marginals() (*in module pypesto.visualize*), 167
sampling_fval_trace() (*in module pypesto.visualize*), 167
sampling_parameters_trace() (*in module pypesto.visualize*), 168
sampling_scatter() (*in module pypesto.visualize*), 168
ScipyOptimizer (*class in pypesto.optimize*), 131

Sequence (*class in pypesto.result*), 177
set_last_sample() (*pypesto.sampling.AdaptiveMetropolisSampler method*), 145
set_last_sample() (*pypesto.sampling.InternalSampler method*), 149
set_last_sample() (*pypesto.sampling.MetropolisSampler method*), 153
setdefault() (*pypesto.objective.HistoryOptions method*), 94
setdefault() (*pypesto.optimize.OptimizeOptions method*), 124
setdefault() (*pypesto.optimize.OptimizerResult method*), 129
setdefault() (*pypesto.profile.ProfileOptions method*), 137
setdefault() (*pypesto.profile.ProfilerResult method*), 140
setdefault() (*pypesto.sampling.McmcPtResult method*), 151
setdefault() (*pypesto.visualize.ReferencePoint method*), 161
SingleCoreEngine (*class in pypesto.engine*), 187
sort() (*pypesto.problem.List method*), 109
sort() (*pypesto.result.OptimizeResult method*), 172
sres (*pypesto.optimize.OptimizerResult attribute*), 127
sres_to_schi2() (*in module pypesto.objective*), 103
start_time (*pypesto.objective.CsvHistory attribute*), 84
start_time (*pypesto.objective.Hdf5History attribute*), 87
start_time (*pypesto.objective.History attribute*), 89
start_time (*pypesto.objective.HistoryBase attribute*), 91
start_time (*pypesto.objective.MemoryHistory attribute*), 96
start_time (*pypesto.objective.OptimizerHistory attribute*), 103
store_steadystate_guess() (*pypesto.objective.AmiciObjective method*), 82
swap_samples() (*pypesto.sampling.AdaptiveParallelTemperingSampler method*), 147
swap_samples() (*pypesto.sampling.ParallelTemperingSampler method*), 155

T

time (*pypesto.optimize.OptimizerResult attribute*), 127
time_path (*pypesto.profile.ProfilerResult attribute*), 138
time_total (*pypesto.profile.ProfilerResult attribute*), 138
translate_options() (*pypesto.sampling.AdaptiveMetropolisSampler class method*), 145
translate_options() (*pypesto.sampling.AdaptiveParallelTemperingSampler class method*), 147
translate_options() (*pypesto.sampling.InternalSampler class method*), 149
translate_options() (*pypesto.sampling.MetropolisSampler class method*), 153
translate_options() (*pypesto.sampling.ParallelTemperingSampler class method*), 155
translate_options() (*pypesto.sampling.Sampler class method*), 157

U

unfix_parameters() (*pypesto.problem.Problem method*), 116
uniform() (*in module pypesto.startpoint*), 189
update() (*pypesto.objective.CsvHistory method*), 84
update() (*pypesto.objective.Hdf5History method*), 87
update() (*pypesto.objective.History method*), 89
update() (*pypesto.objective.HistoryBase method*), 91
update() (*pypesto.objective.HistoryOptions method*), 94
update() (*pypesto.objective.MemoryHistory method*), 96
update() (*pypesto.objective.OptimizerHistory method*), 103
update() (*pypesto.optimize.OptimizeOptions method*), 124
update() (*pypesto.optimize.OptimizerResult method*), 129
update() (*pypesto.profile.ProfileOptions method*), 137
update() (*pypesto.profile.ProfilerResult method*), 141
update() (*pypesto.sampling.McmcPtResult method*), 151
update() (*pypesto.visualize.ReferencePoint method*), 161
update_from_problem() (*pypesto.objective.AggregatedObjective method*), 74
update_from_problem() (*pypesto.objective.AmiciObjective method*), 82
update_from_problem() (*pypesto.objective.Objective method*), 101
update_from_problem() (*pypesto.problem.Objective method*), 113

V

values() (*pypesto.objective.HistoryOptions method*),

94
values() (*pypesto.optimize.OptimizeOptions method*), 125
values() (*pypesto.optimize.OptimizerResult method*), 129
values() (*pypesto.profile.ProfileOptions method*), 137
values() (*pypesto.profile.ProfilerResult method*), 141
values() (*pypesto.sampling.McmcPtResult method*), 151
values() (*pypesto.visualize.ReferencePoint method*), 161

W

waterfall() (*in module pypesto.visualize*), 168
waterfall_lowlevel() (*in module pypesto.visualize*), 169

X

x (*pypesto.optimize.OptimizerResult attribute*), 126
x (*pypesto.visualize.ReferencePoint attribute*), 159
x0 (*pypesto.optimize.OptimizerResult attribute*), 127
x_free_indices (*pypesto.problem.Problem attribute*), 114
x_path (*pypesto.profile.ProfilerResult attribute*), 138