pyPESTO Documentation

Release 0.0.12

The pyPESTO developers

Apr 06, 2020

User's guide

Version: 0.0.12
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)

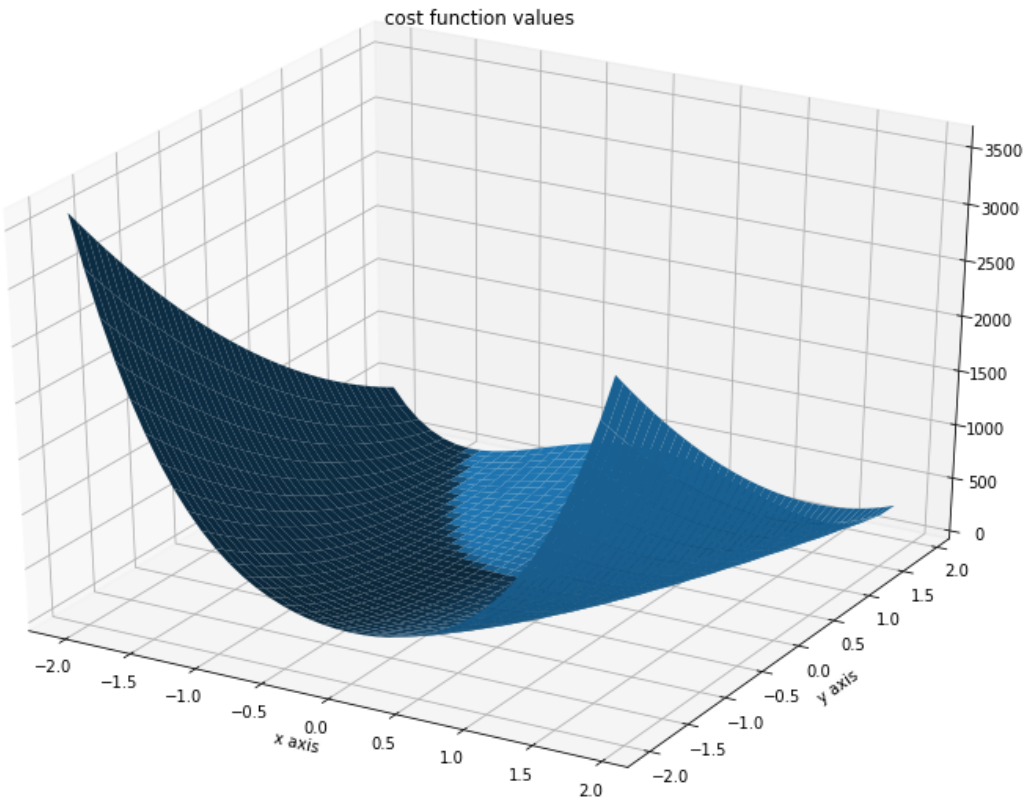
```
problem1 = pypesto.Problem(objective=objective1, lb=lb, ub=ub)
problem2 = pypesto.Problem(objective=objective2, lb=lb, ub=ub)
```

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')
```



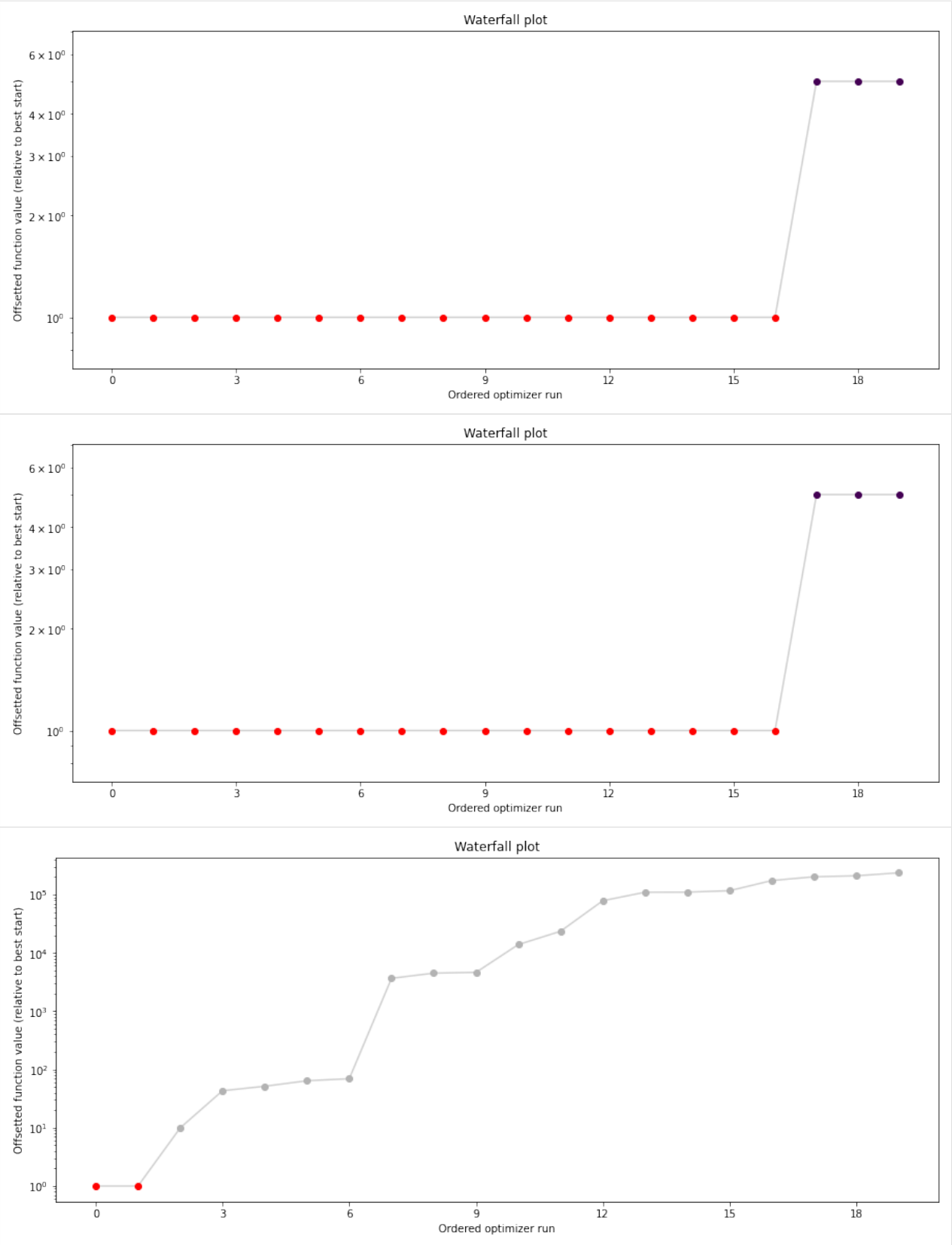
2.1.3 Run optimization

[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>
```

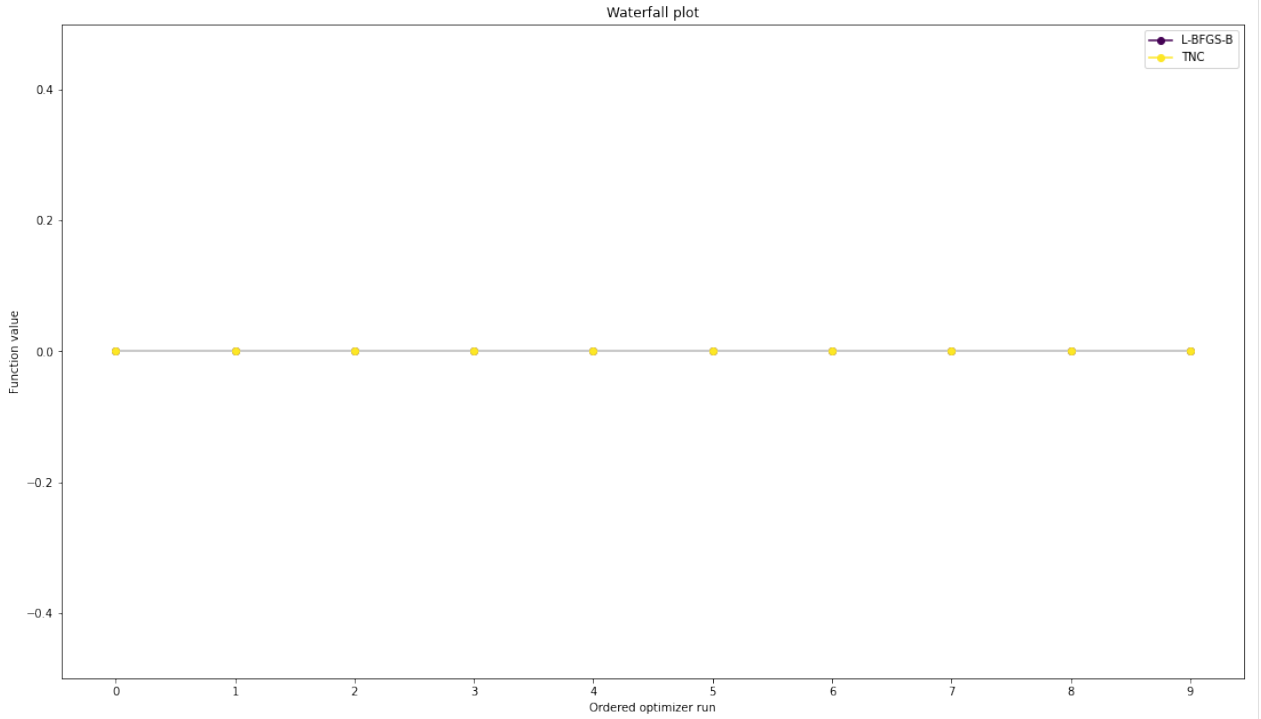


We can now have a closer look, which method perfomred better: Let's first compare bfgs and TNC, since both methods

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')
```

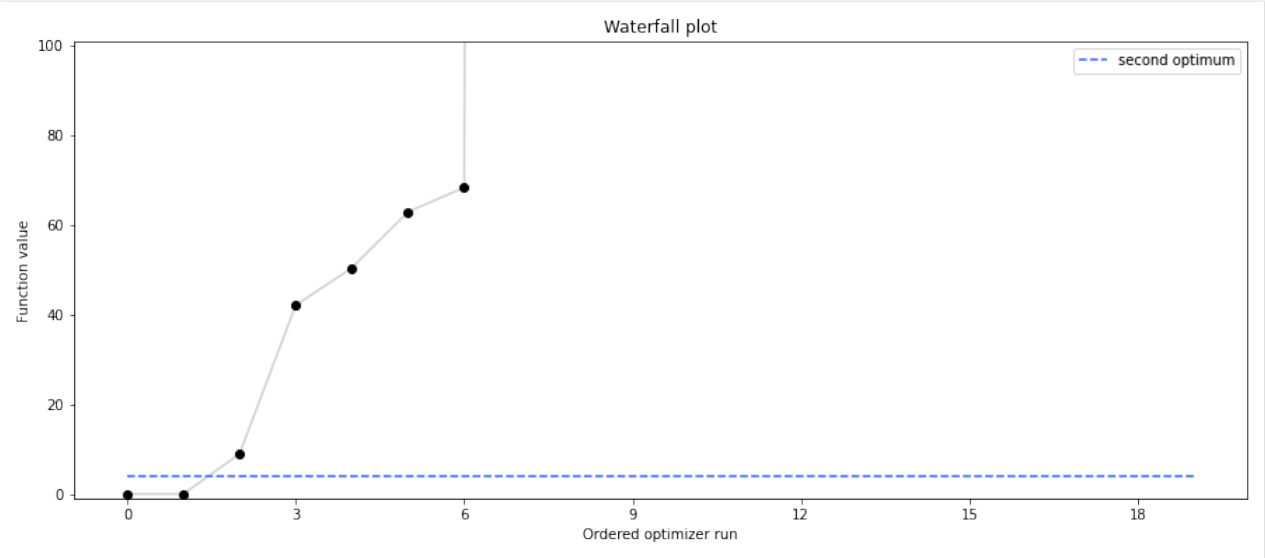
[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd93936a410>
```



[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>
```

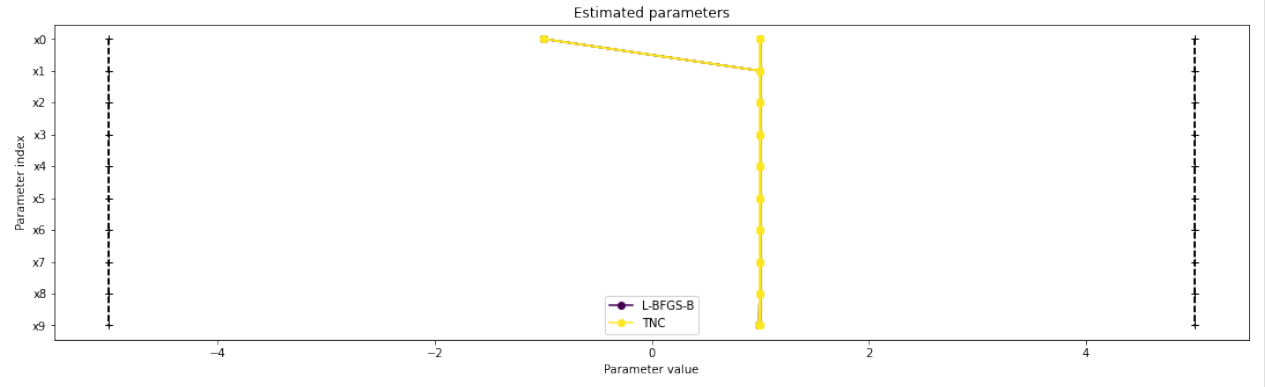


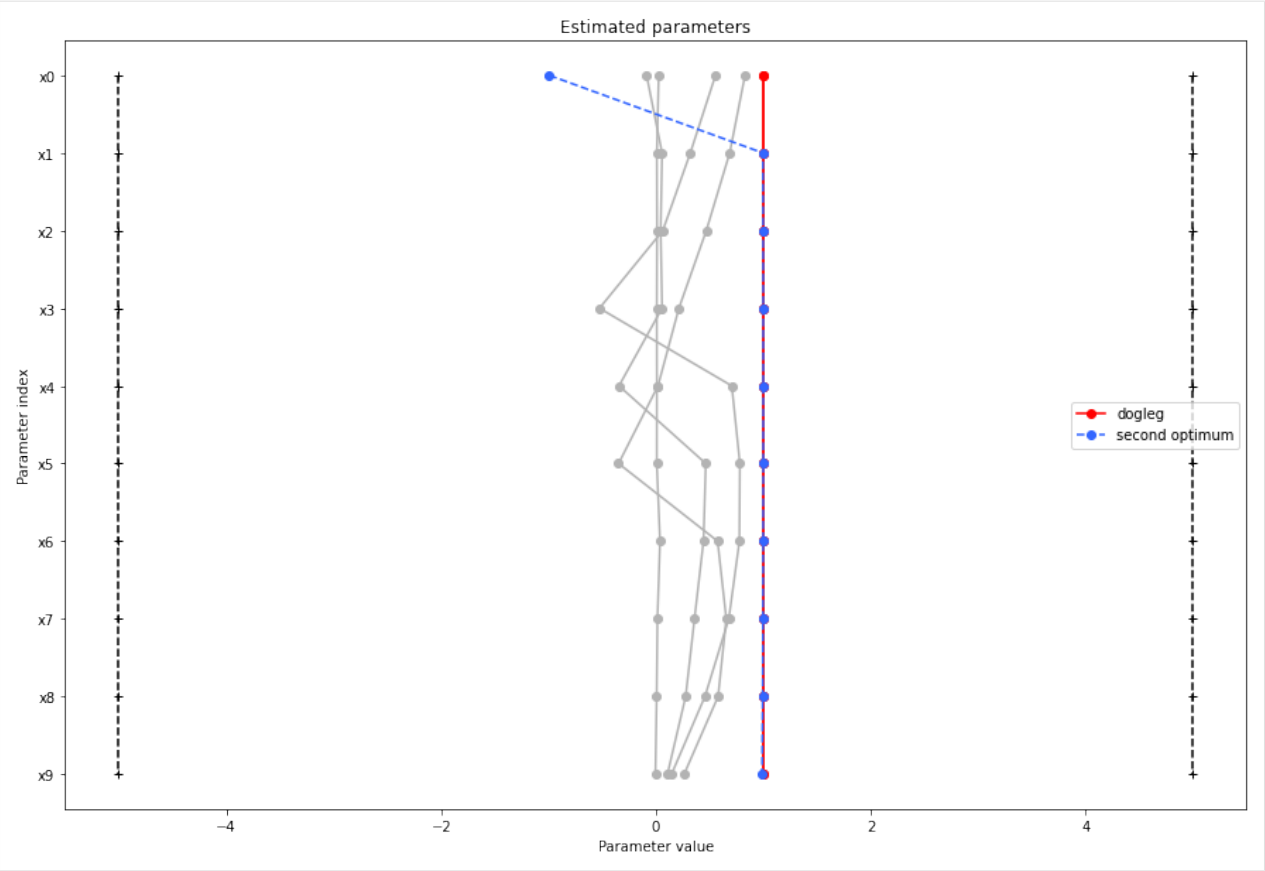
2.1.5 Visualize parameters

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>
```



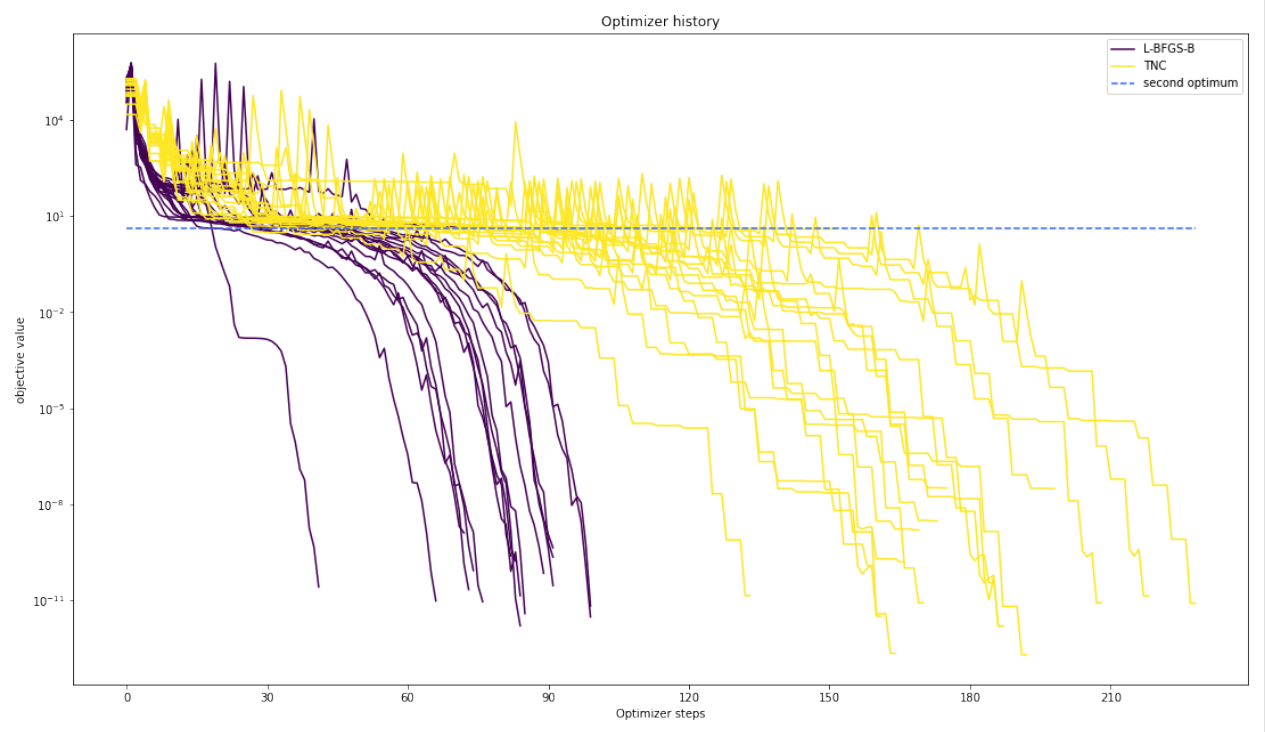


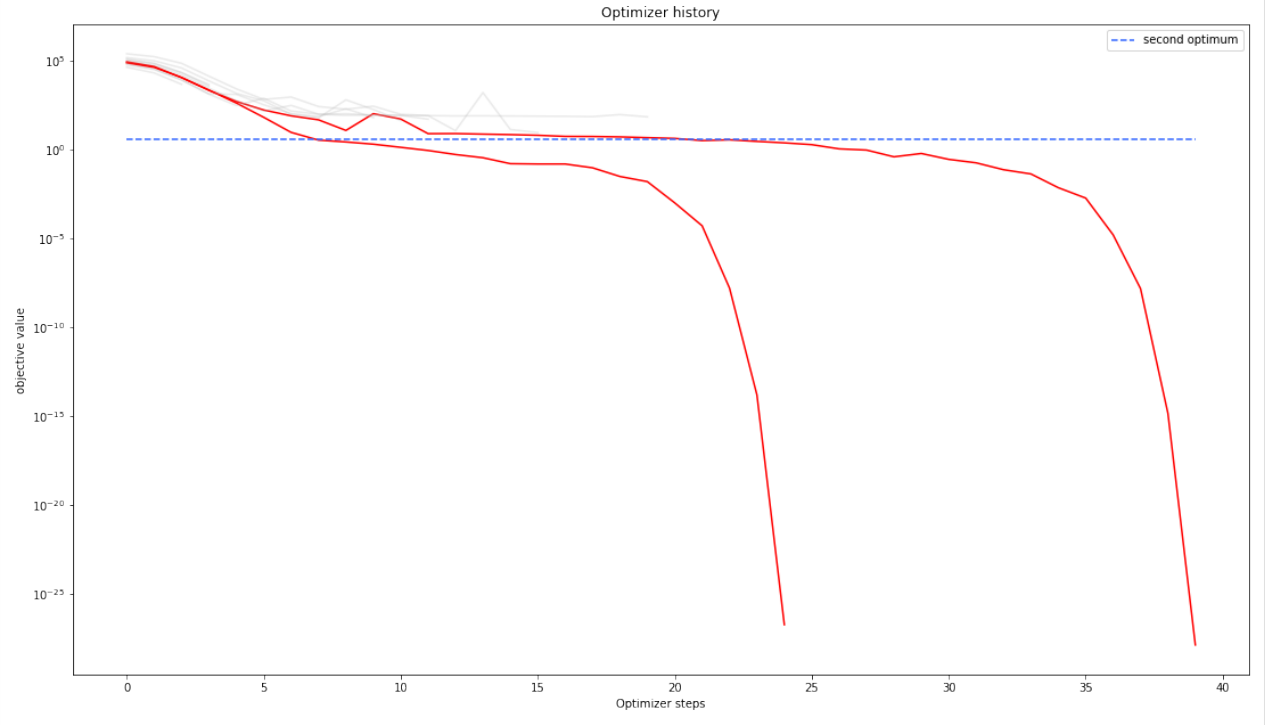
2.1.6 Optimizer history

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. . .

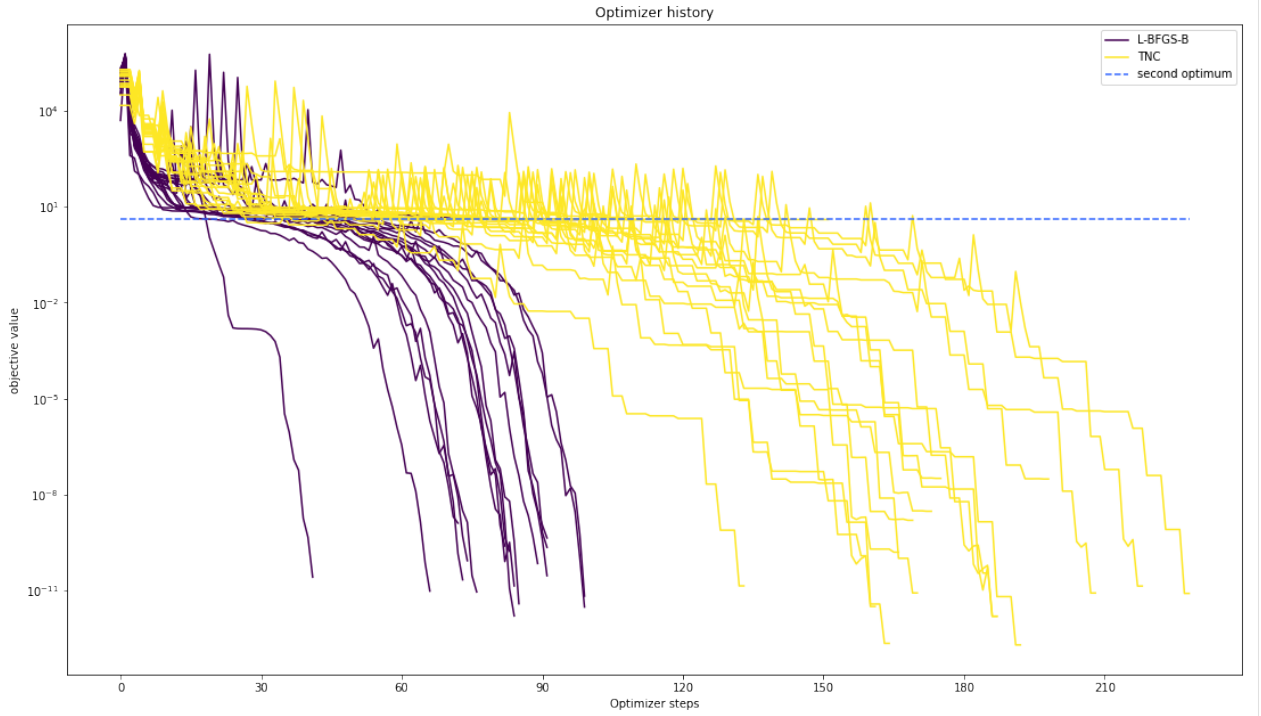
[11]:
```
# plot one list of waterfalls
pypesto.visualize.optimizer_history([result1_bfgs, result1_tnc],
                                    legends=['L-BFGS-B', 'TNC'],
                                    reference=ref,
```

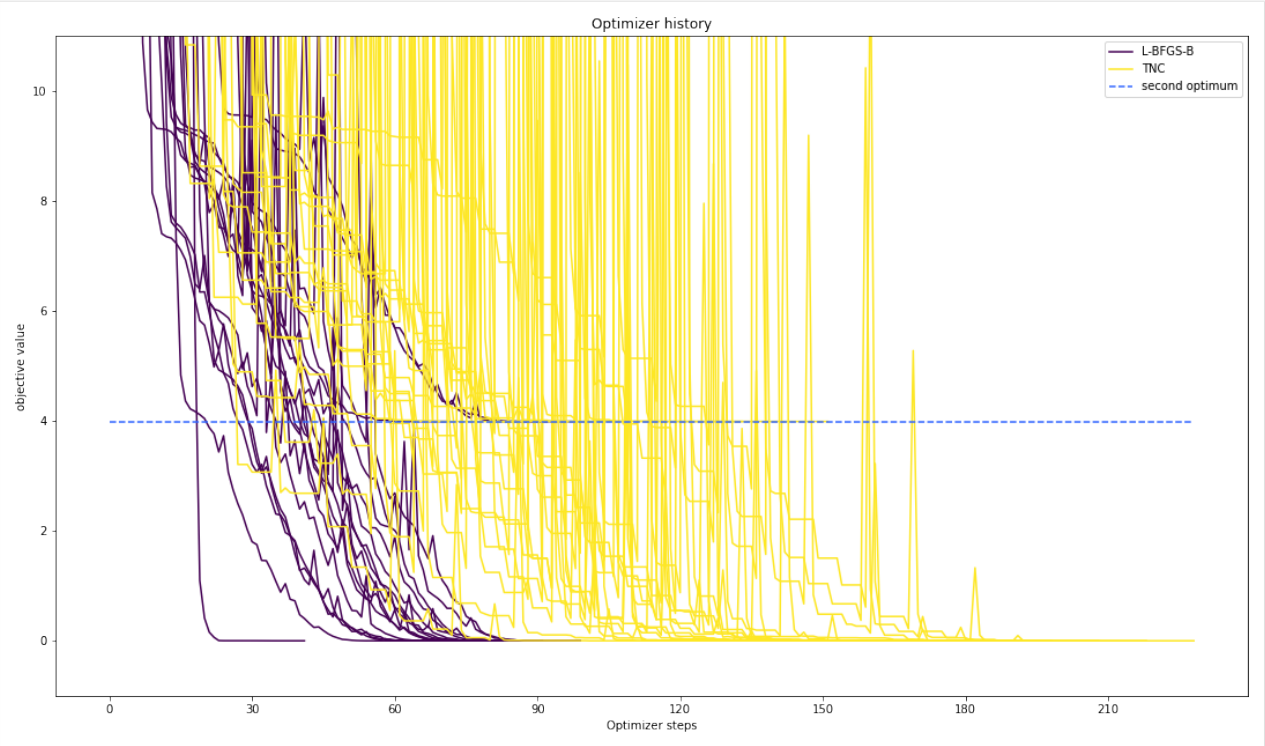
(continues on next page)

(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.

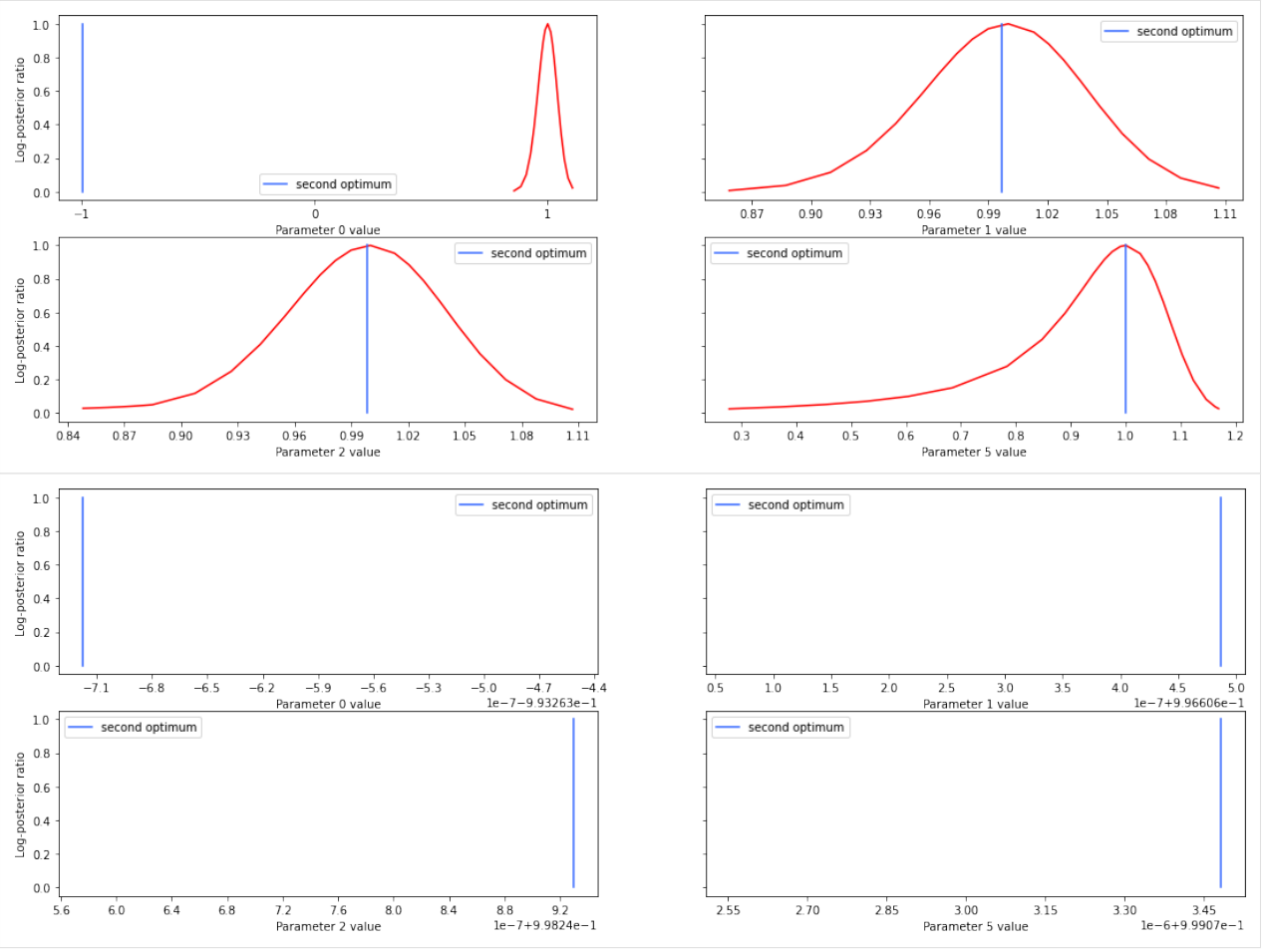
Finally, an integer (result_index) can be passed, in order to specify the local optimum, from which profiling should be started.

[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)
```

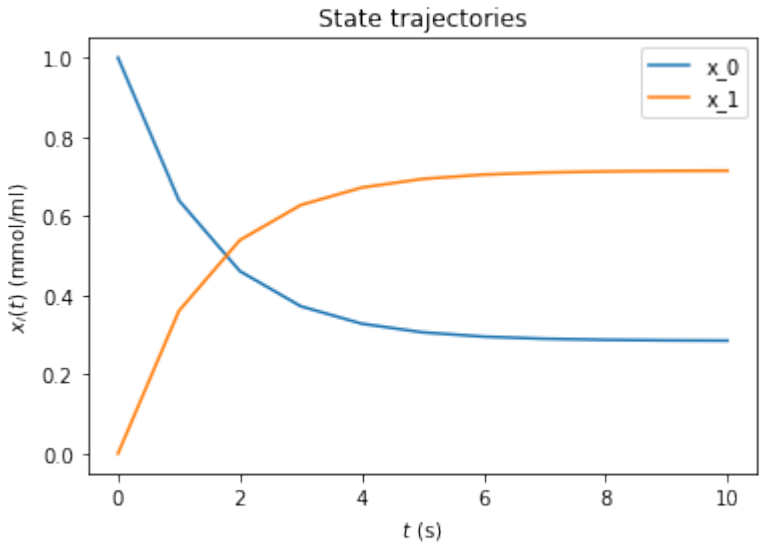
2.2.2 Load AMICI model

[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)

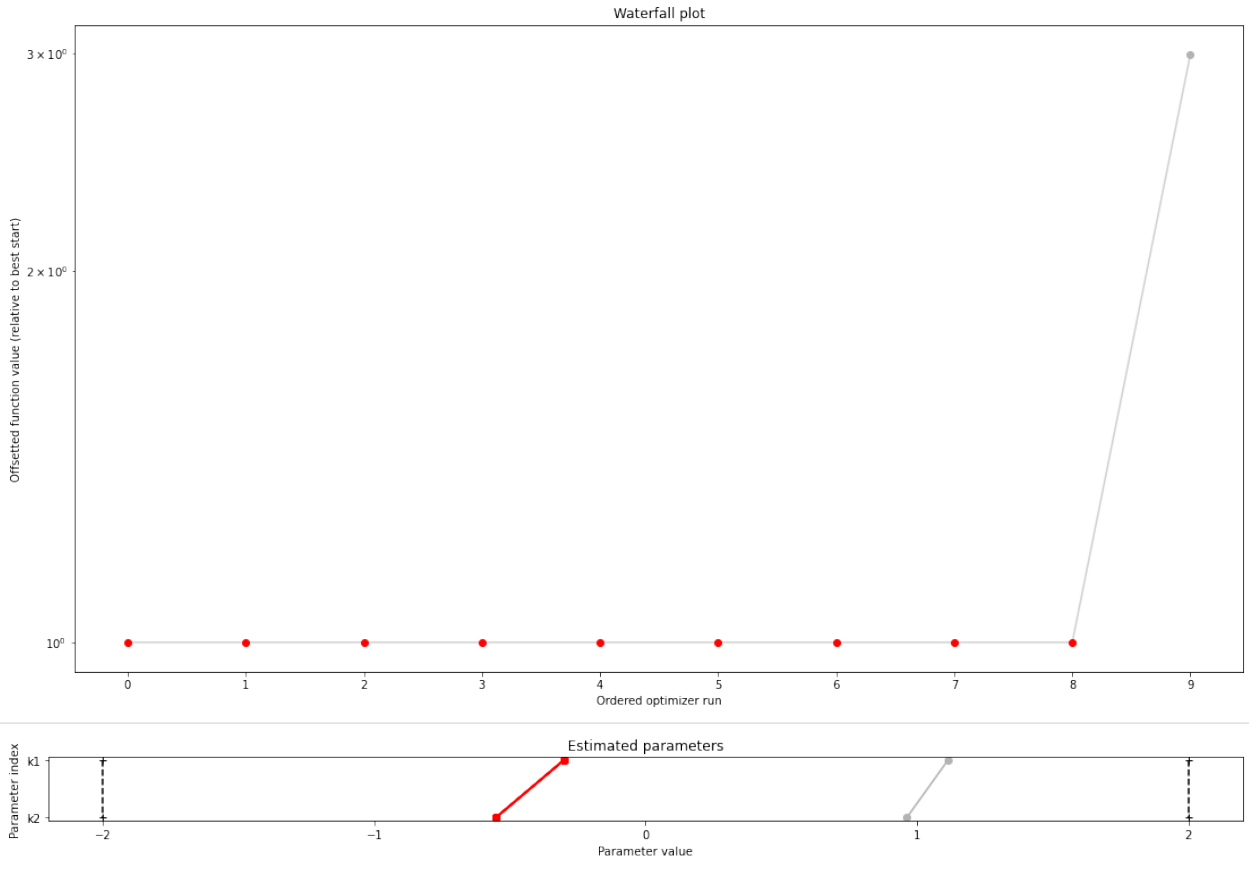
(continued from previous page)

```
# 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)
```

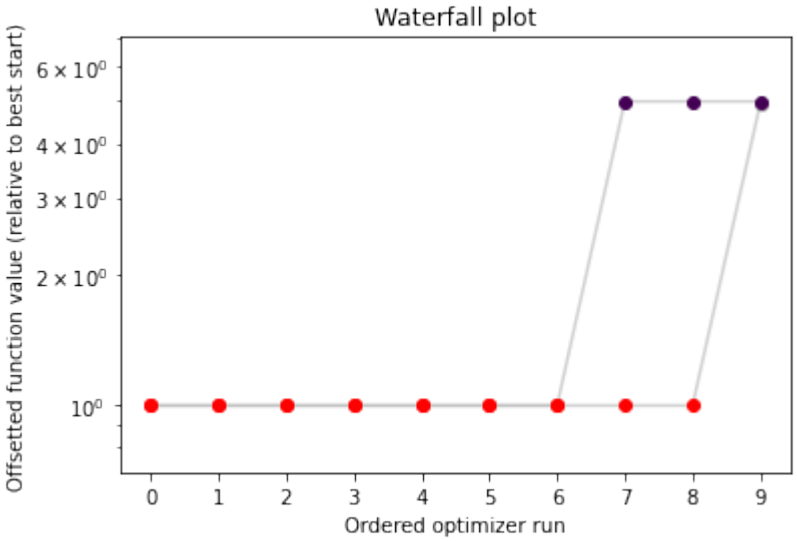
2.3.2 Optimize

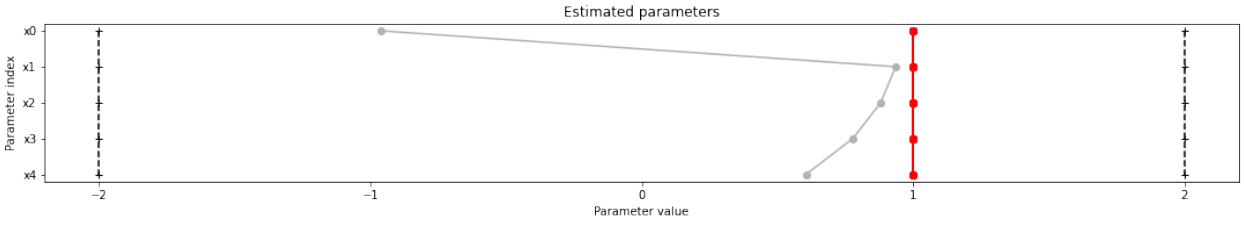
[3]:
```
optimizer = pypesto.ScipyOptimizer()
n_starts = 10

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

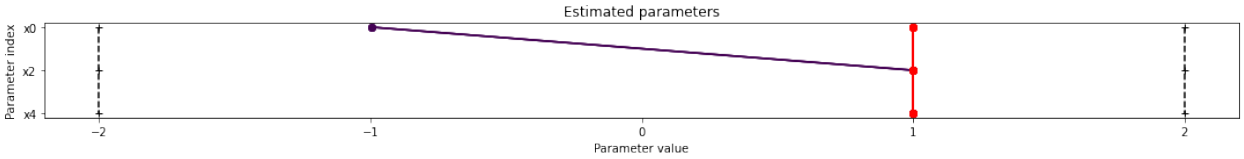
2.3.3 Visualize

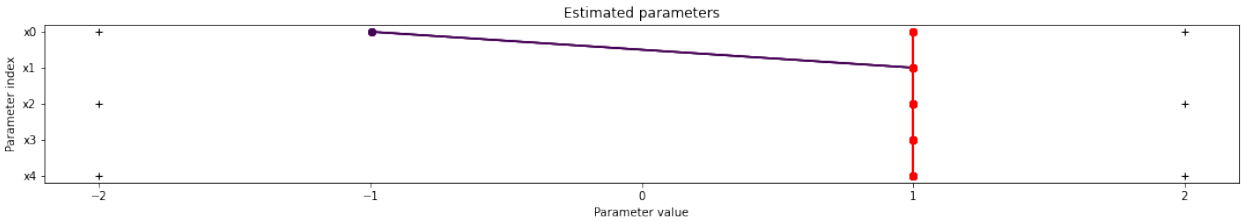
[4]:
```
fig, ax = plt.subplots()
pypesto.visualize.waterfall(result1, ax)
pypesto.visualize.waterfall(result2, ax)
pypesto.visualize.parameters(result1)
pypesto.visualize.parameters(result2)
pypesto.visualize.parameters(result2, free_indices_only=False)
```

[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

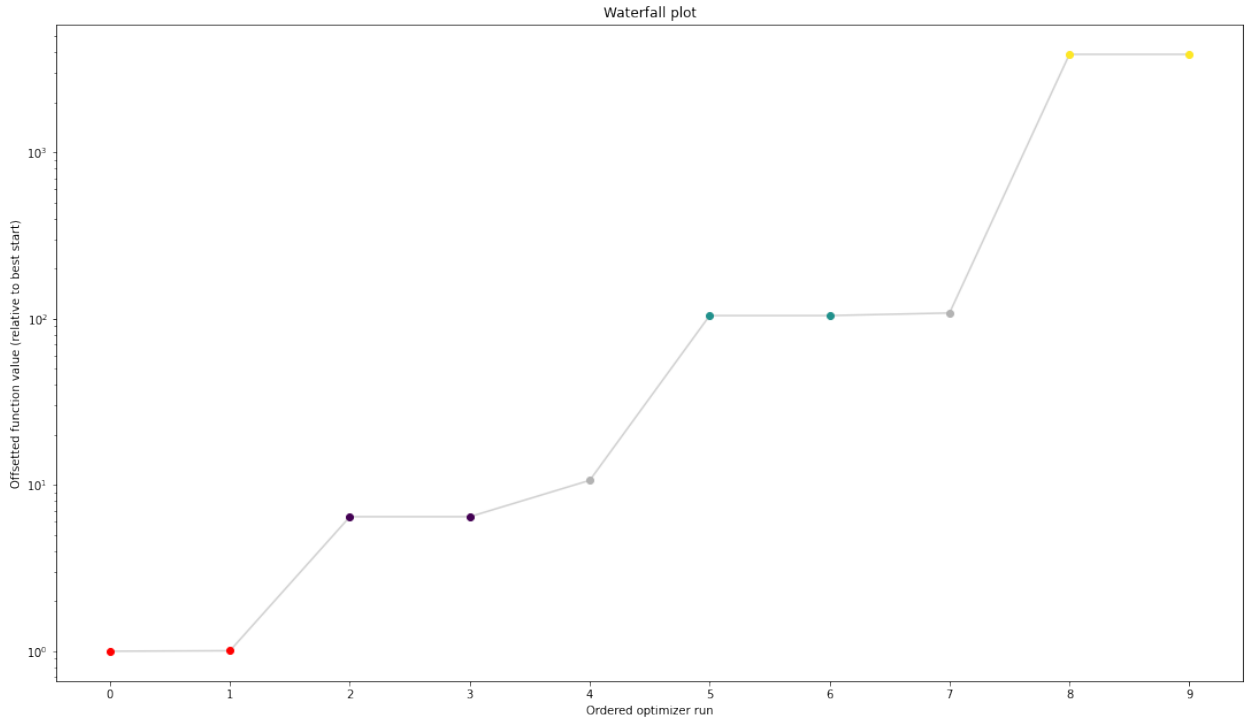
[18]:
```
# waterfall, parameter space,
import pypesto.visualize
```

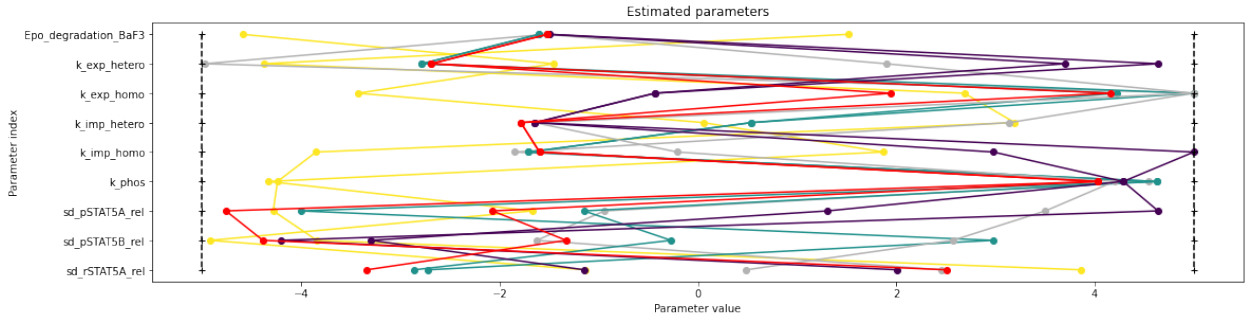
(continues on next page)

(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.

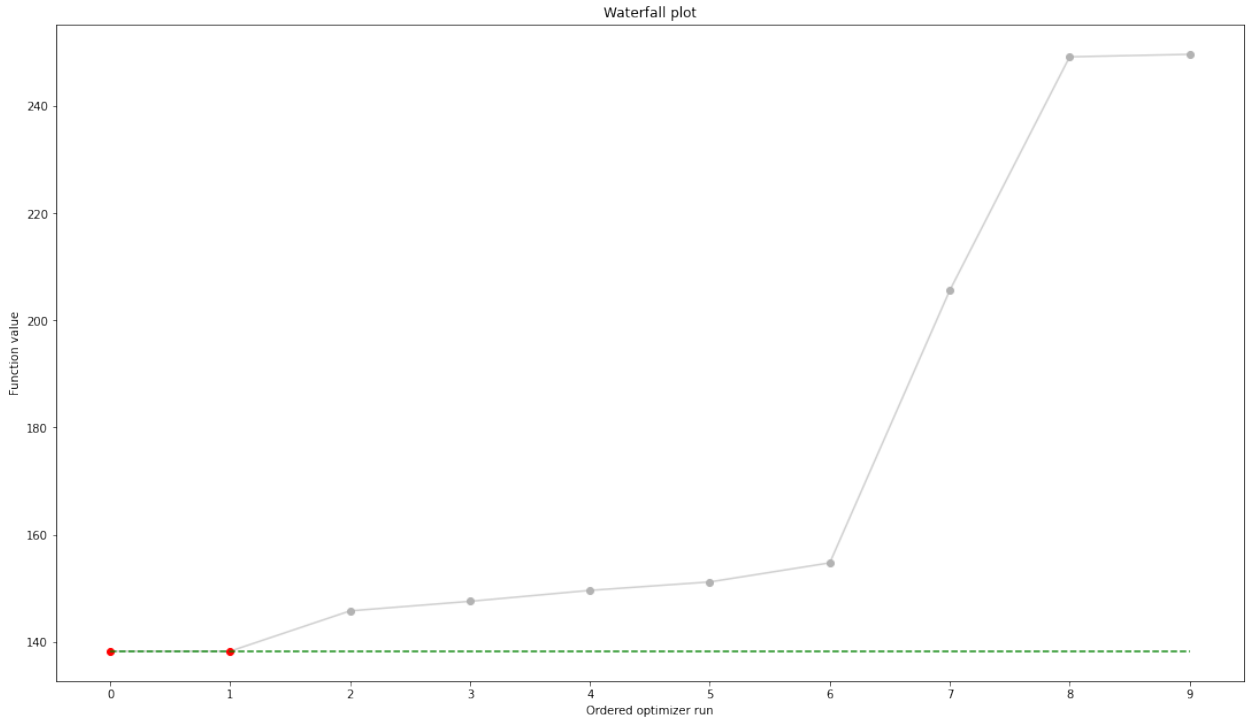
[13]:
```
import pypesto.visualize

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

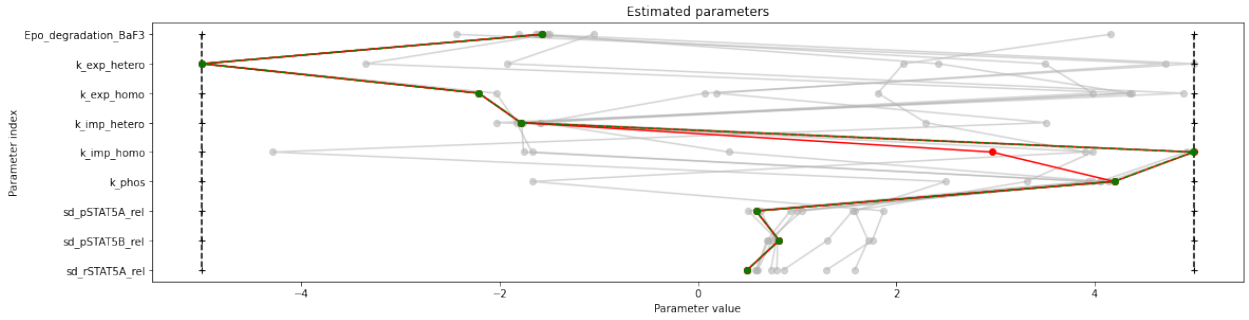
(continues on next page)

(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': 0.013525247573852539,
 '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]),
 '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)

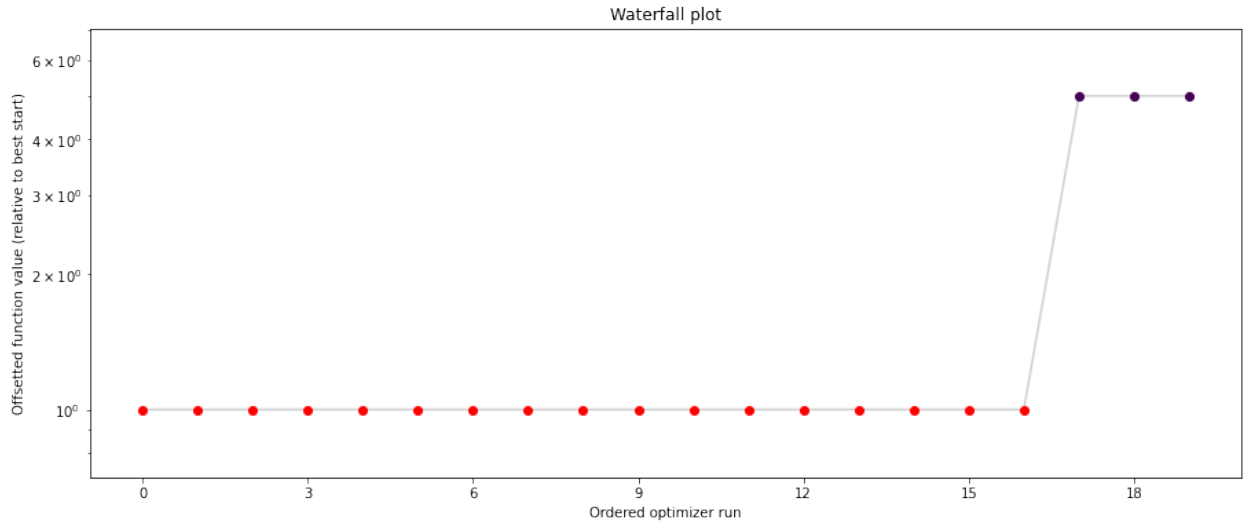
(continued from previous page)

```
  '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)

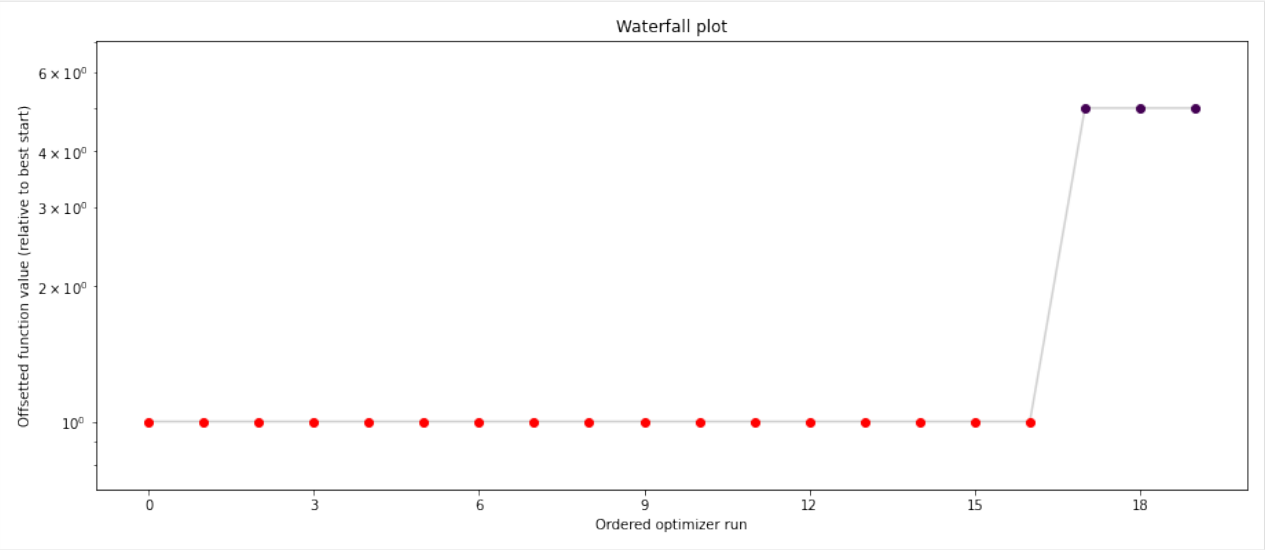
(continued from previous page)

```
 '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

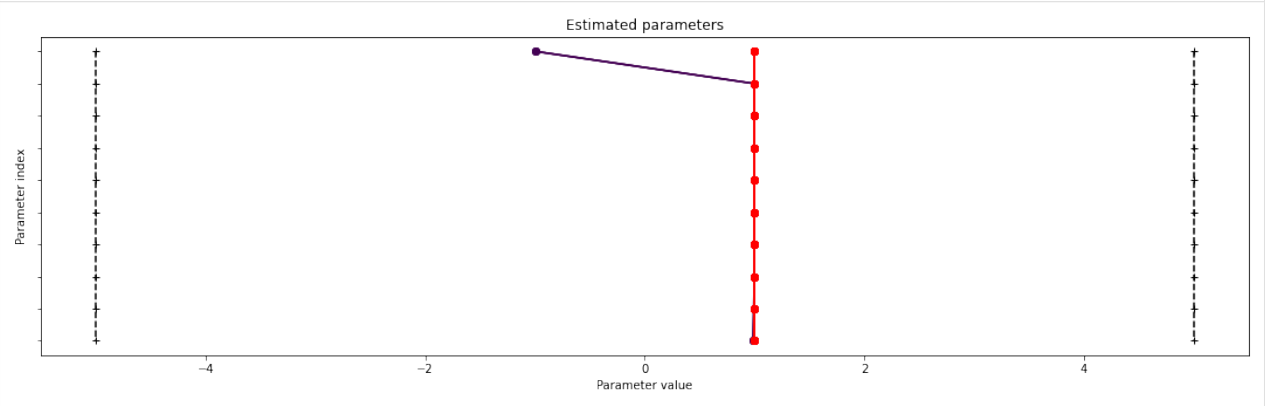
[9]:
```
# plot waterfalls
pypesto.visualize.waterfall(result, size=(15,6))
```

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



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

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



[ ]:

2.7 Download the examples as notebooks

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

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

CHAPTER 6

Problem

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

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

Optimize

CHAPTER 8

Profile

CHAPTER 9

Sample

CHAPTER 10

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)

CHAPTER 11

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.

CHAPTER 12

Visualize

pypesto comes with various visualization routines. To use these, import pypesto.visualize.

CHAPTER 13

Startpoint

Method 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.

CHAPTER 14

Release notes

14.1 0.0 series

14.1.1 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).

14.1.2 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).

14.1.3 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).

14.1.4 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)

14.1.5 0.0.8 (2019-09-01)

- Add logo (#178).
- Fix petab API changes (#179).
- Some minor bugfixes (#168).

14.1.6 0.0.7 (2019-03-21)

- Support noise models in Petab and Amici.
- Minor Petab update bug fixes.

14.1.7 0.0.6 (2019-03-13)

- Several minor error fixes, in particular on tests and steady state.

14.1.8 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.

14.1.9 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)

14.1.10 0.0.3 (2019-01-30)

- Import amici models and the petab data format automatically using pypesto.PetabImporter.
- Basic profiling routines.

14.1.11 0.0.2 (2018-10-18)

- Fix parameter values
- Record trace of function values
- Amici objective to directly handle amici models

14.1.12 0.0.1 (2018-07-25)

- Basic framework and implementation of the optimization

CHAPTER 15

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 16

Contact

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

- Yannik Schälte: yannik.schaelte@gmail.com

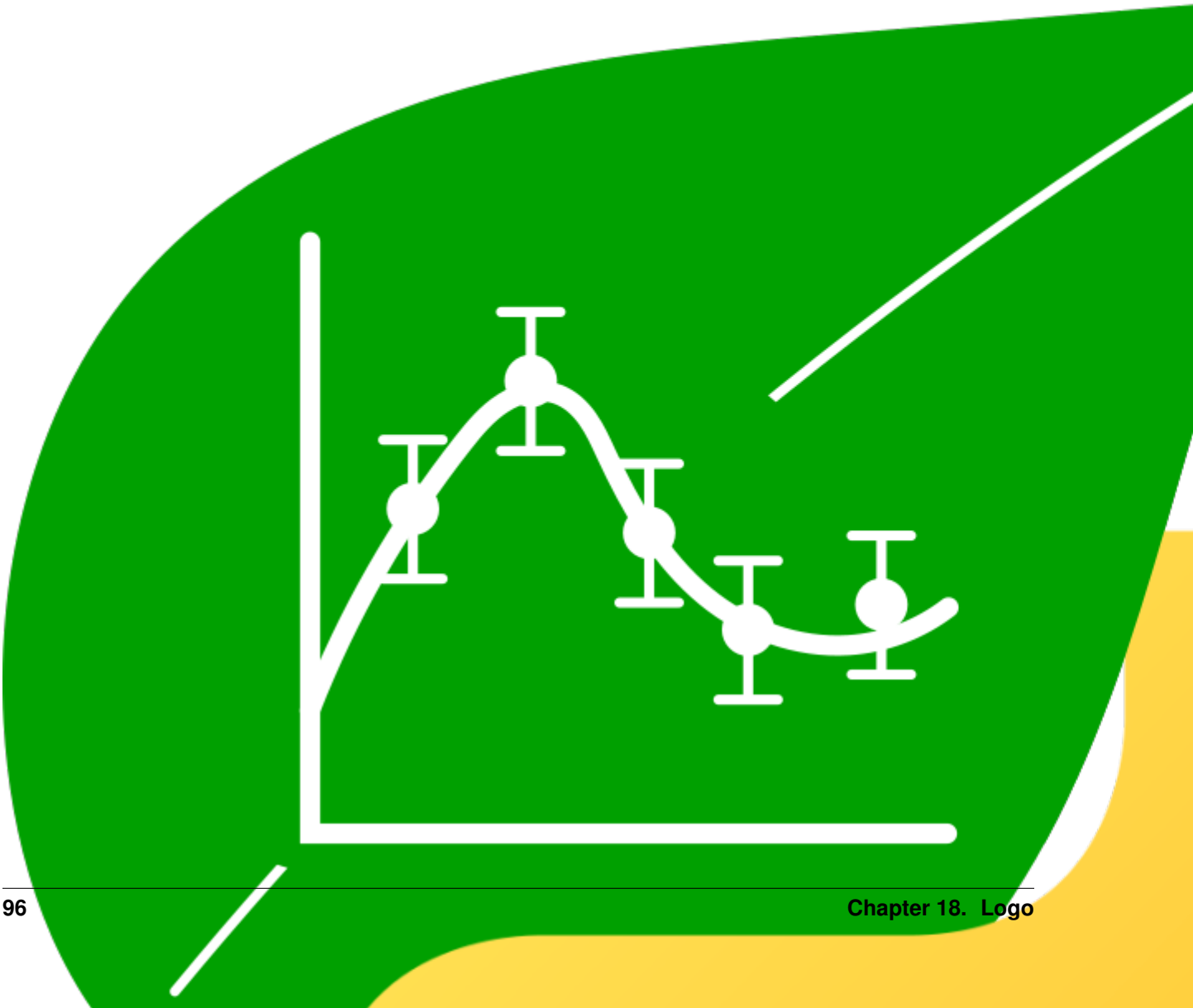
CHAPTER 17

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 18

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 19

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

C

D

F

G

N

O

P

R

S

U

X