pyPESTO Documentation

Release 0.2.5

The pyPESTO developers

May 05, 2021

USER'S GUIDE

Version: 0.2.5
Source code: https://github.com/icb-dcm/pypesto

CHAPTER

ONE

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

TWO

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 pypesto.visualize as visualize
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

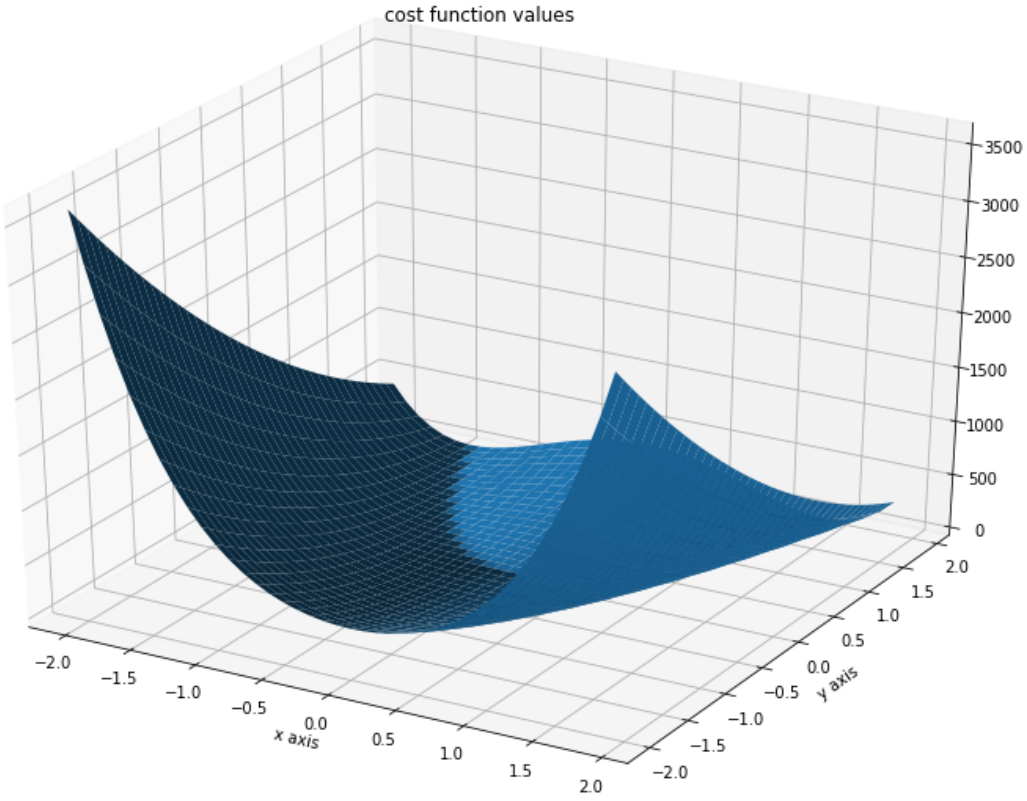
[2]:
```
# first type of objective
objective1 = pypesto.Objective(fun=sp.optimize.rosen,
                               grad=sp.optimize.rosen_der,
                               hess=sp.optimize.rosen_hess)

# 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))

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

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')
```



2.1.2 Run optimization

[5]:
```
import pypesto.optimize as optimize
```

[6]:
```
%%time

# create different optimizers
optimizer_bfgs = optimize.ScipyOptimizer(method='l-bfgs-b')
optimizer_tnc = optimize.ScipyOptimizer(method='TNC')
optimizer_dogleg = optimize.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 = optimize.minimize(
    problem=problem1, optimizer=optimizer_bfgs,
    n_starts=n_starts, history_options=history_options)
result1_tnc = optimize.minimize(
    problem=problem1, optimizer=optimizer_tnc,
    n_starts=n_starts, history_options=history_options)
result1_dogleg = optimize.minimize(
    problem=problem1, optimizer=optimizer_dogleg,
    n_starts=n_starts, history_options=history_options)

# Optimize second type of objective
result2 = optimize.minimize(
    problem=problem2, optimizer=optimizer_tnc, n_starts=n_starts)
```

```
Function values from history and optimizer do not match: 2.685929315976887, 2.
↪9820162443657527
Parameters obtained from history and optimizer do not match: [ 9.81668828e-01  9.
↪70664899e-01  9.44081691e-01  8.86185728e-01
  7.98866760e-01  6.27503392e-01  3.49193984e-01  1.00863999e-01
  1.18119992e-02 -4.83096174e-04], [0.98232811 0.9607169  0.93006041 0.86376905 0.
↪72679074 0.51464422
 0.25715153 0.03390018 0.0134388  0.00224348]
Function values from history and optimizer do not match: 2.932320470464073, 3.
↪104833804292291
Parameters obtained from history and optimizer do not match: [9.85695690e-01 9.
↪66998320e-01 9.34405532e-01 8.72211105e-01
 7.61716907e-01 5.82160864e-01 2.90132686e-01 5.88015713e-02
 1.02493152e-02 1.44786818e-04], [9.76820249e-01 9.49203293e-01 9.03611145e-01 8.
↪32711736e-01
 6.92021069e-01 4.71244784e-01 2.26981271e-01 1.93600745e-02
 9.06285841e-03 3.00716108e-04]
Function values from history and optimizer do not match: 7.128857018893593, 7.
↪737539574292646
Parameters obtained from history and optimizer do not match: [-9.74521002e-01  9.
↪48916364e-01  8.98382180e-01  7.95485807e-01
  6.32334509e-01  3.95389632e-01  1.55262583e-01  2.24615758e-02
  9.92812211e-03  4.70538835e-05], [-0.95137113  0.92287756  0.85600385  0.74220324
↪0.53469862  0.25223695
  0.05388462  0.01175751  0.01035533  0.00121333]
Function values from history and optimizer do not match: 4.047666500407507, 4.
↪8092850089870245
```

(continues on next page)

(continued from previous page)

```
Parameters obtained from history and optimizer do not match: [9.57097378e-01 9.
↪15272882e-01 8.35583627e-01 6.92924153e-01
 4.69156347e-01 1.98916115e-01 2.87951418e-02 1.21495892e-02
 1.14276335e-02 2.48487865e-04], [9.37837181e-01 8.73541886e-01 7.61292462e-01 5.
↪64720865e-01
 2.84119482e-01 6.17767487e-02 1.53662912e-02 1.54992154e-02
 1.49513982e-02 2.98560604e-04]
Function values from history and optimizer do not match: 4.760963749486806, 5.
↪255690010134404
Parameters obtained from history and optimizer do not match: [-0.98990379  0.98886801␣
↪  0.98189121  0.96587616  0.93451723  0.87262109
  0.75889559  0.56883791  0.31364781  0.07883034], [-0.99248035  0.99162316  0.
↪97889433  0.95364865  0.91078502  0.8261375
  0.68236478  0.45820395  0.17444197  0.01383626]
Function values from history and optimizer do not match: 1.8159939922237558, 2.
↪5425135960926237
Parameters obtained from history and optimizer do not match: [9.90583524e-01 9.
↪80917081e-01 9.63072632e-01 9.30325108e-01
 8.61713989e-01 7.40678602e-01 5.38268550e-01 2.71328618e-01
 5.43996813e-02 7.89698144e-04], [9.89162276e-01 9.78043570e-01 9.51094059e-01 9.
↪02211862e-01
 8.07645490e-01 6.35406055e-01 3.75384767e-01 1.11075371e-01
 1.30319964e-02 2.11963742e-04]
Function values from history and optimizer do not match: 2.2546577084566284, 2.
↪988463828057193
Parameters obtained from history and optimizer do not match: [9.86890406e-01 9.
↪73738159e-01 9.51089323e-01 9.02086672e-01
 8.09027663e-01 6.46629154e-01 4.04671023e-01 1.51442890e-01
 1.94187285e-02 2.45054194e-04], [9.81148194e-01 9.60640784e-01 9.21690034e-01 8.
↪55030060e-01
 7.31180374e-01 5.23069013e-01 2.44624625e-01 3.39441804e-02
 1.03741576e-02 2.45306769e-05]
Function values from history and optimizer do not match: 0.3545683077008359, 0.
↪5906121485206447
Parameters obtained from history and optimizer do not match: [0.99668472 0.99262575 0.
↪98945665 0.98129911 0.96532923 0.93081497
 0.86315388 0.74328951 0.53910453 0.2736718 ], [0.9963228  0.99215562 0.98514259 0.
↪97132273 0.94683482 0.89670025
 0.80300196 0.64224614 0.40061592 0.14210795]
Function values from history and optimizer do not match: 1.442951465698237, 2.
↪117657844069939
Parameters obtained from history and optimizer do not match: [0.99253701 0.98698288 0.
↪97438388 0.94864025 0.89249411 0.79343394
 0.62110958 0.37154848 0.12142293 0.00337751], [9.85576055e-01 9.72515609e-01 9.
↪52500598e-01 9.14984751e-01
 8.40282960e-01 7.07108893e-01 4.93844010e-01 2.19299261e-01
 1.80684261e-02 2.39353088e-04]
Function values from history and optimizer do not match: 0.4310215367360306, 0.
↪7200757805862191
Parameters obtained from history and optimizer do not match: [0.99728801 0.99265292 0.
↪98655945 0.97724776 0.95330363 0.91375386
 0.83290125 0.68949822 0.4687524  0.21461214], [0.99666432 0.99530499 0.9871224  0.
↪96976884 0.94230384 0.89383977
 0.79420195 0.62752848 0.3793222  0.11129682]
Function values from history and optimizer do not match: 6.33997905147026, 7.
↪069668392692864
Parameters obtained from history and optimizer do not match: [7.84450616e-01 6.
↪10188497e-01 3.64032562e-01 1.19476022e-01
```

(continues on next page)

(continued from previous page)

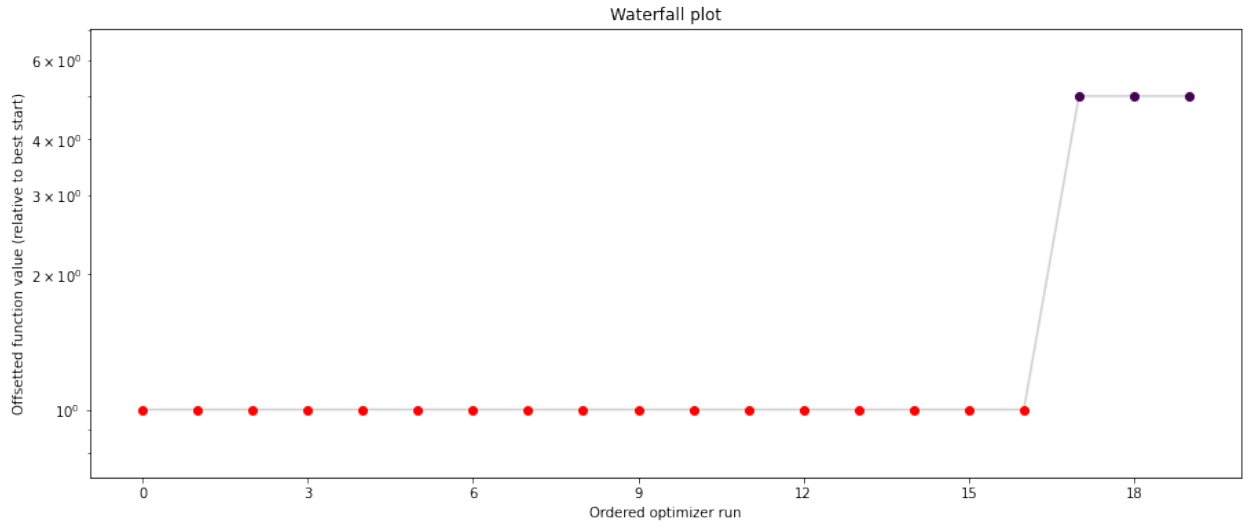
```
 1.25200919e-02 9.74166479e-03 1.00503247e-02 8.51949533e-03
 9.92120699e-03 1.97235559e-04], [ 7.13358486e-01  4.93846146e-01  2.05601150e-01  2.
↪46828697e-02
  1.00531820e-02  8.83759494e-03  9.93584452e-03  1.16356575e-02
  1.00772170e-02 -9.19777874e-05]
Function values from history and optimizer do not match: 1.080010188007246, 1.
↪638996874292531
Parameters obtained from history and optimizer do not match: [0.99354151 0.98796198 0.
↪97743947 0.96147507 0.92290179 0.84825176
 0.71159467 0.49318554 0.223647   0.03035961], [0.99093761 0.98310117 0.96952353 0.
↪94165684 0.88399848 0.77718421
 0.59296742 0.3287277  0.08605952 0.00216266]
Function values from history and optimizer do not match: 6.334069745693479, 7.
↪027921368861192
Parameters obtained from history and optimizer do not match: [-0.98264119  0.97390376␣
↪ 0.94694027  0.8905699   0.79188661  0.62198099
  0.37540054  0.12148722  0.01380672  0.00504649], [-0.97385408  0.95844934  0.
↪9257917   0.85697013  0.71970238  0.49533252
  0.21270446  0.03011495  0.00979574 -0.00651404]
```

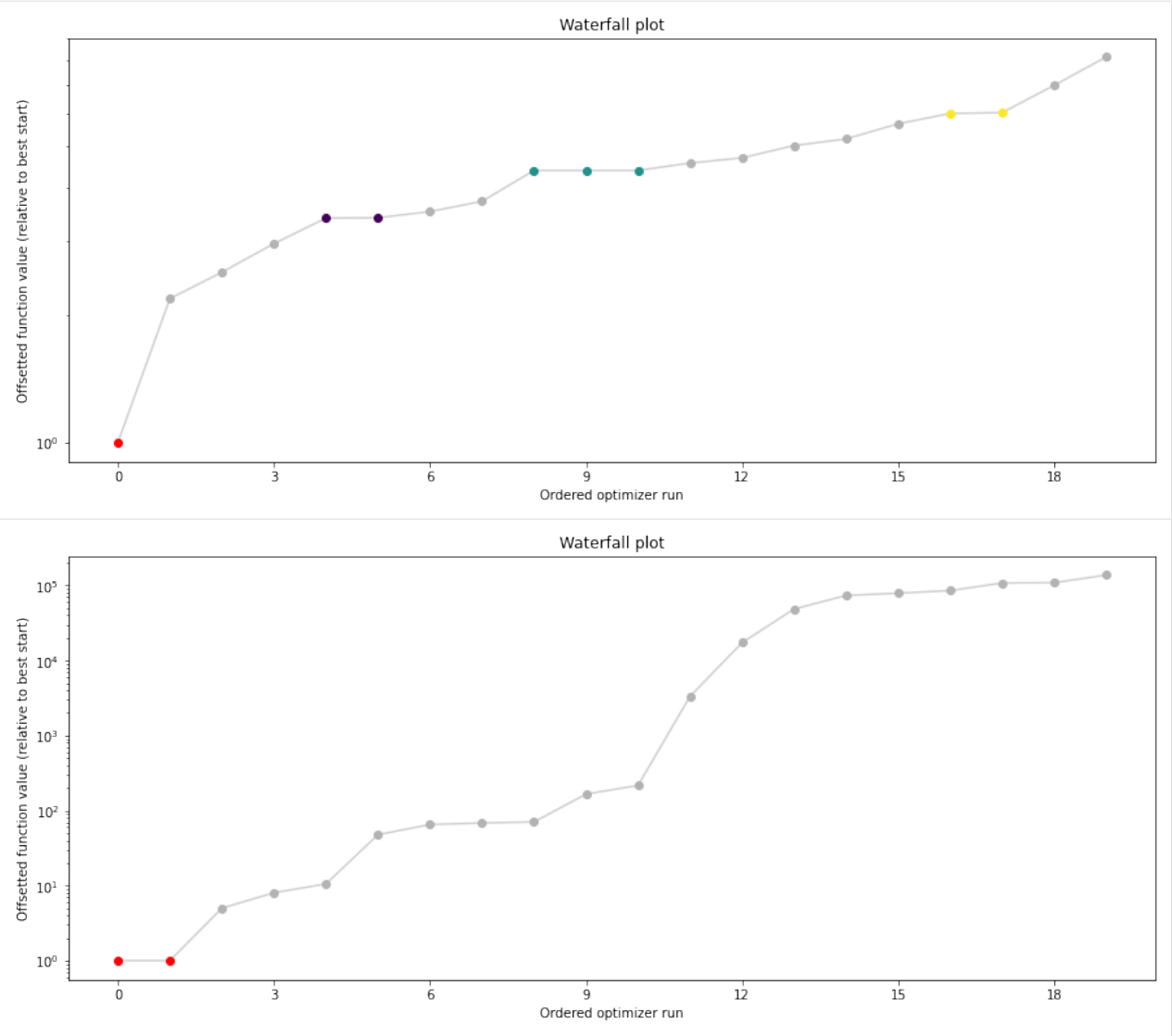
```
CPU times: user 2.74 s, sys: 37.7 ms, total: 2.78 s
Wall time: 2.74 s
```

Visualize and compare optimization results

[7]:
```
# plot separated waterfalls
visualize.waterfall(result1_bfgs, size=(15,6))
visualize.waterfall(result1_tnc, size=(15,6))
visualize.waterfall(result1_dogleg, size=(15,6))
```

[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3be1ba0d0>
```

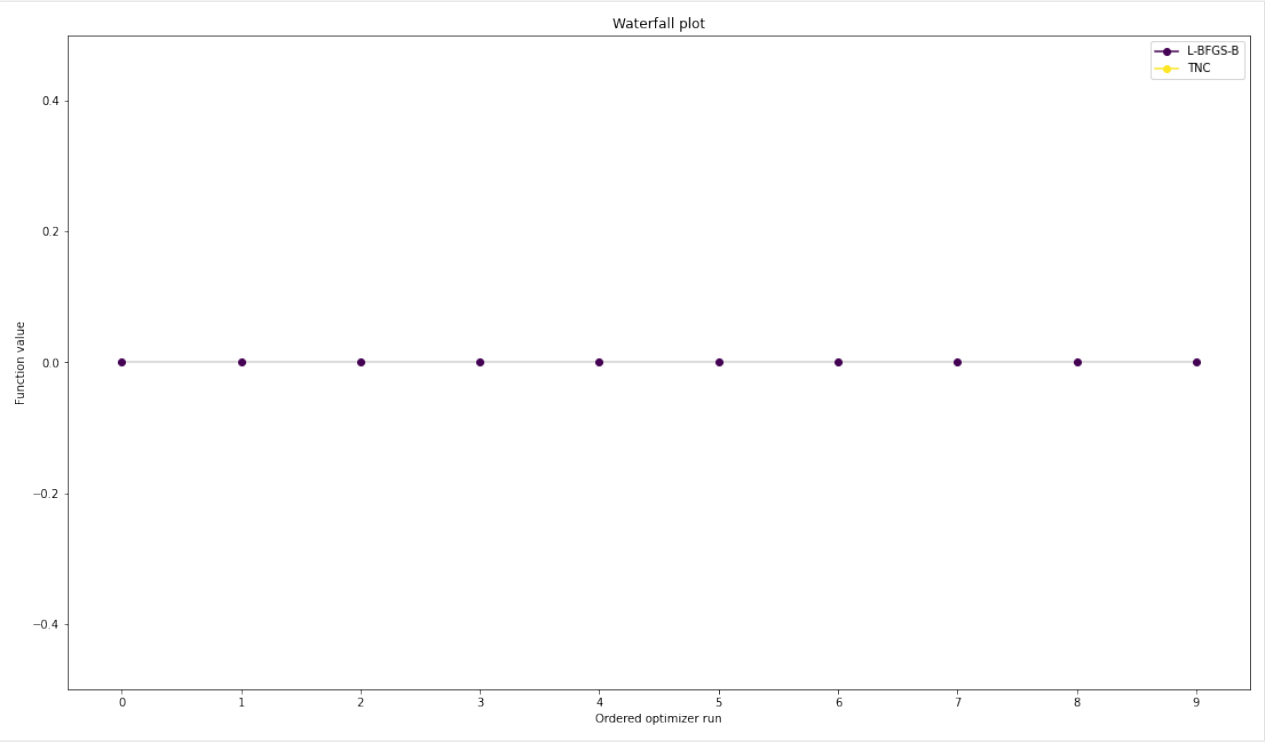




We can now have a closer look, which method perfomred better: Let's first compare bfgs and TNC, since both methods gave good results. How does the fine convergence look like?

[8]:
```
# plot one list of waterfalls
visualize.waterfall([result1_bfgs, result1_tnc],
                    legends=['L-BFGS-B', 'TNC'],
                    start_indices=10,
                    scale_y='lin')
```

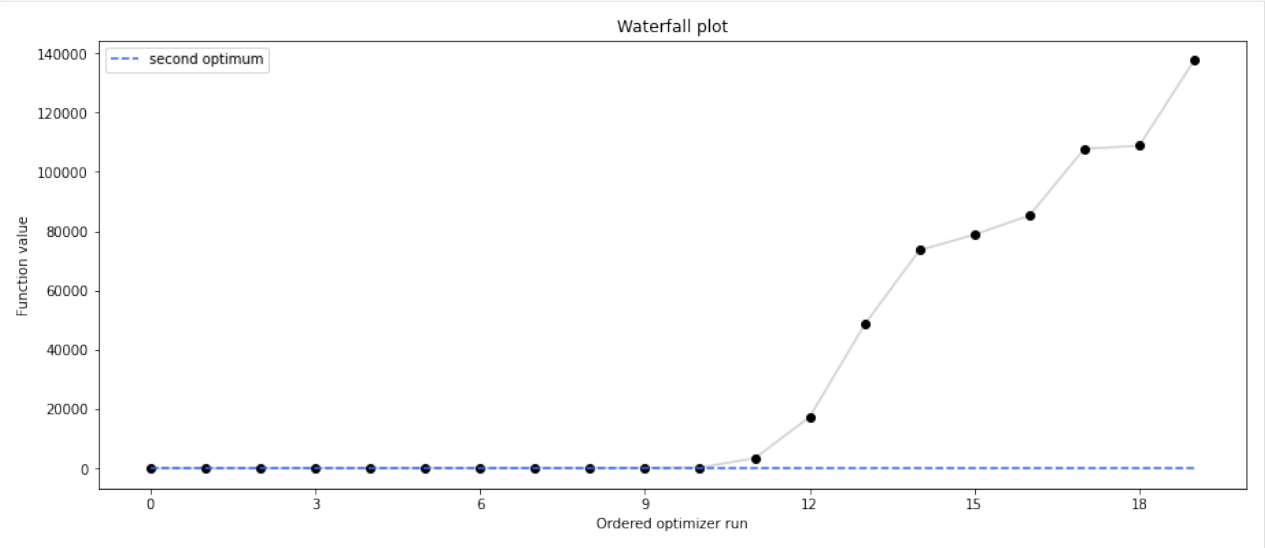
[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3bf463760>
```



[9]:
```
# 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 = visualize.create_references(ref)

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

```
Second optimum at: 3.9865791142048876
```

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3bde83940>
```

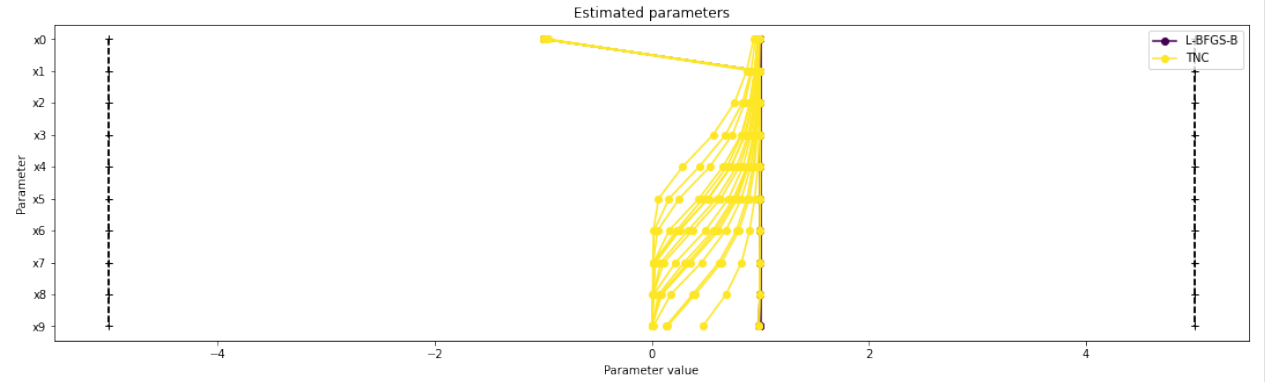


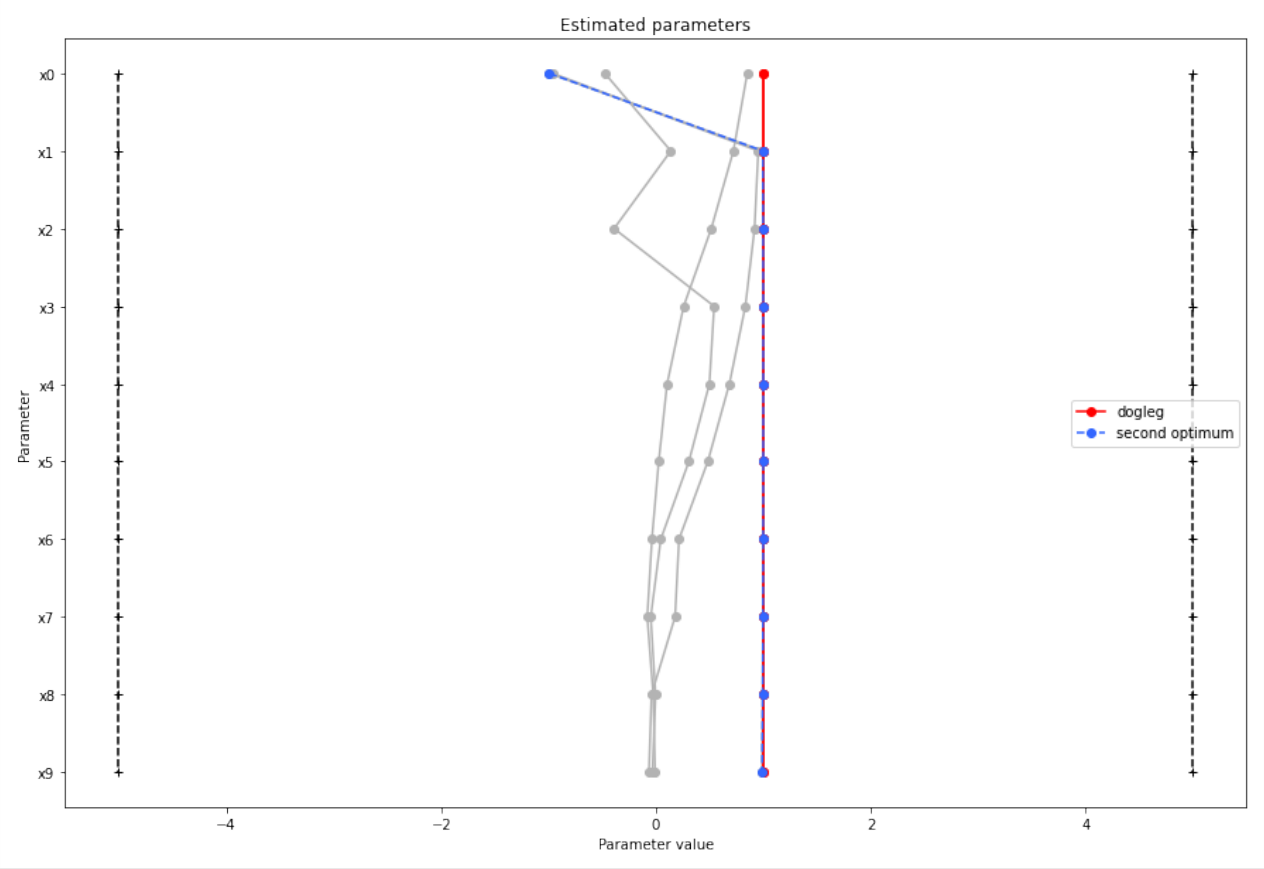
2.1.3 Visualize parameters

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

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

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3bdd5b430>
```





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

[11]:
```
df = result1_tnc.optimize_result.as_dataframe(
    ['fval', 'n_fval', 'n_grad', 'n_hess', 'n_res', 'n_sres', 'time'])
df.head()
```

[11]:

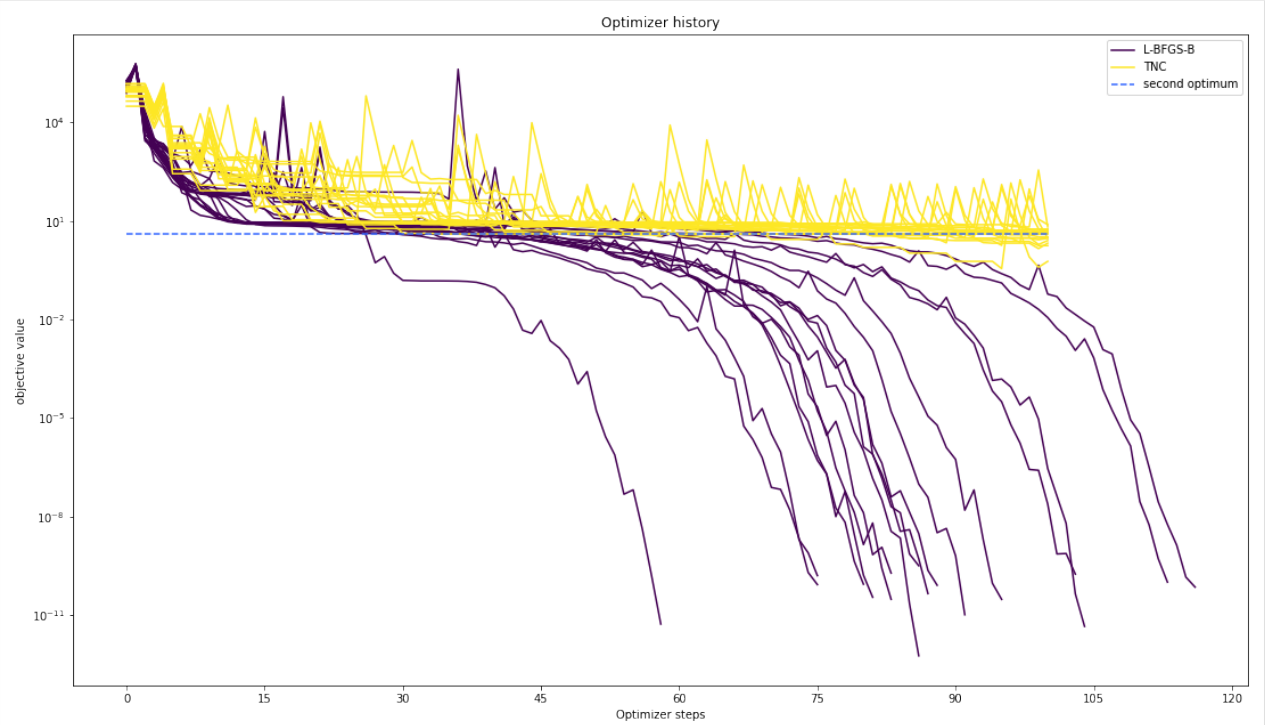
	fval	n_fval	n_grad	n_hess	n_res	n_sres	time
0	0.590612	101	101	0	0	0	0.052775
1	1.779748	101	101	0	0	0	0.049476
2	2.117658	101	101	0	0	0	0.039615
3	2.542514	101	101	0	0	0	0.064188
4	2.982016	101	101	0	0	0	0.024157

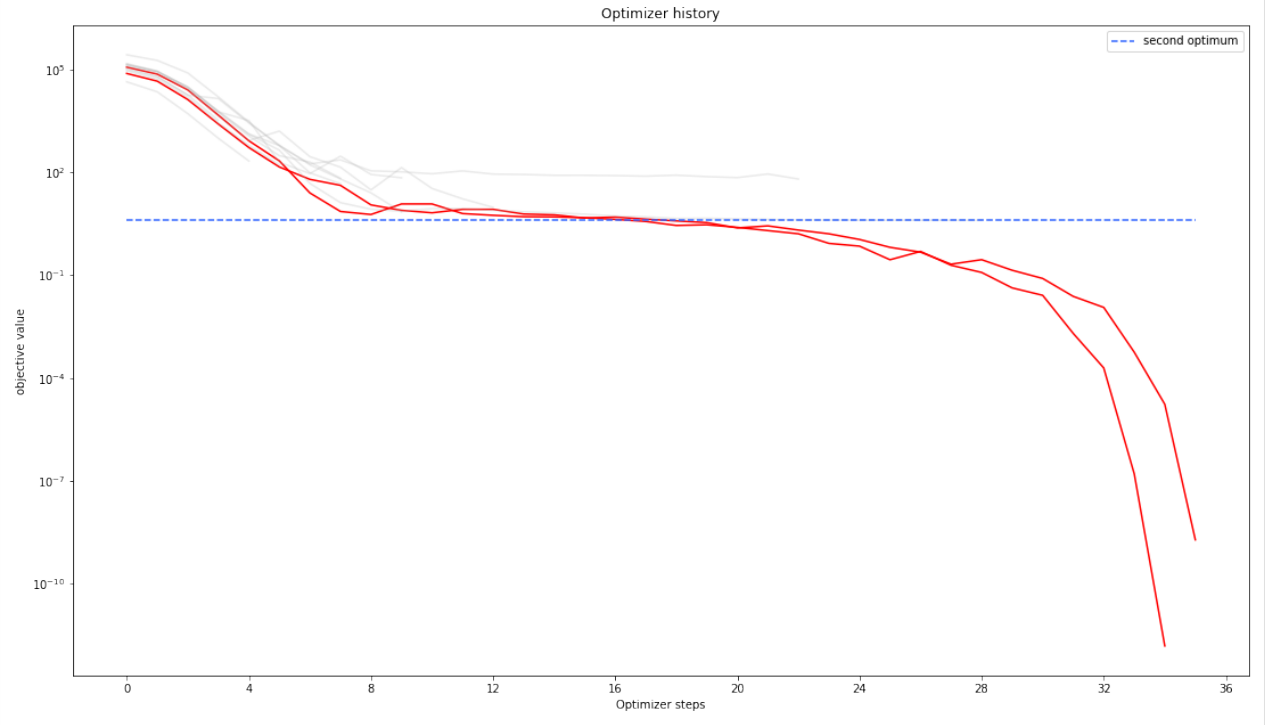
Optimizer history

Let's compare optimzer progress over time.

[12]:
```
# plot one list of waterfalls
visualize.optimizer_history([result1_bfgs, result1_tnc],
                            legends=['L-BFGS-B', 'TNC'],
                            reference=ref)
# plot one list of waterfalls
visualize.optimizer_history(result1_dogleg,
                            reference=ref)
```

[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3bdbcc820>
```





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

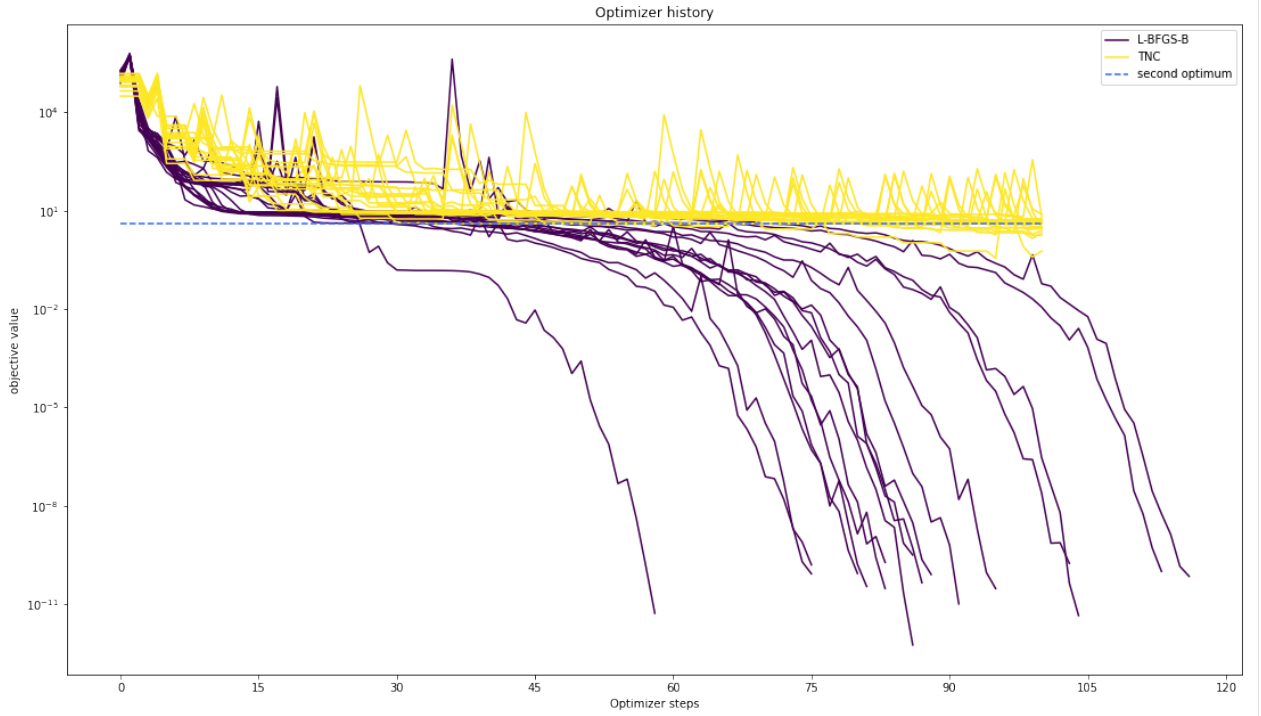
[13]:
```
# plot one list of waterfalls
visualize.optimizer_history([result1_bfgs, result1_tnc],
                            legends=['L-BFGS-B', 'TNC'],
                            reference=ref,
```

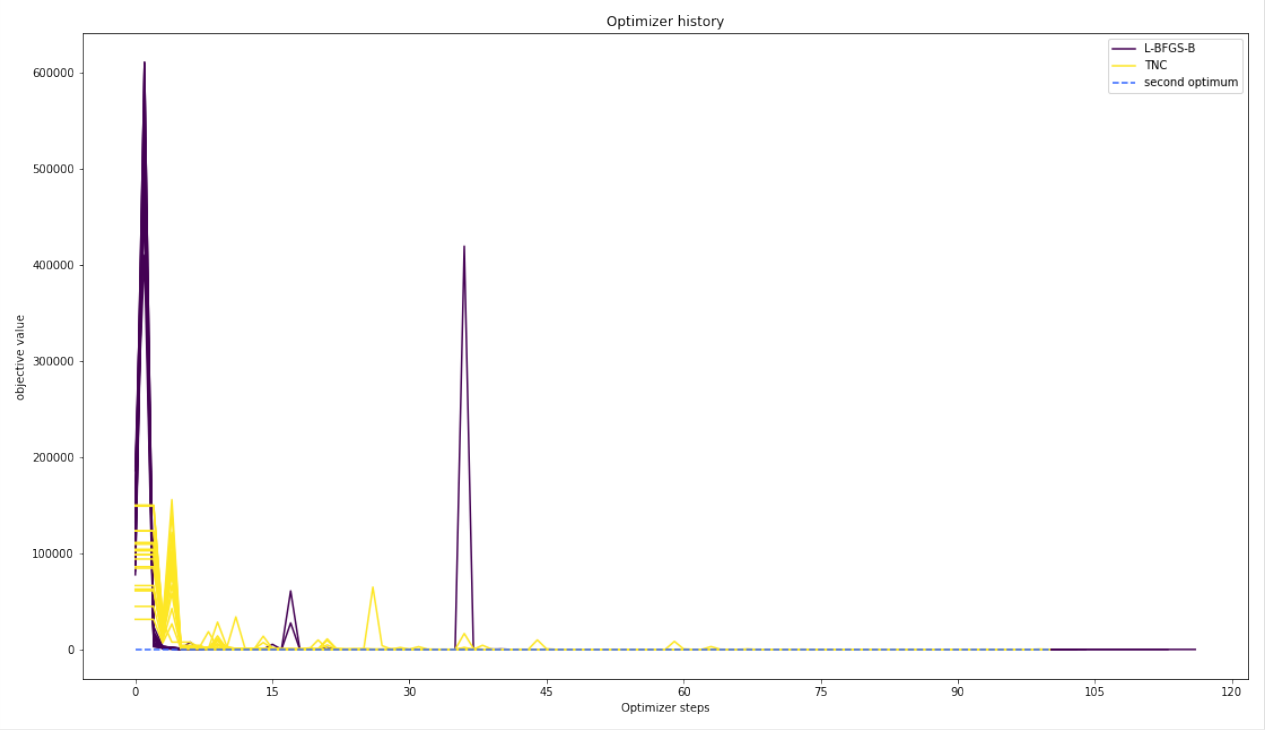
(continues on next page)

(continued from previous page)

```
                              offset_y=0.)

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

[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb3be01d130>





2.1.4 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.

[14]:
```
import pypesto.profile as profile
```

[15]:
```
%%time

# compute profiles
profile_options = profile.ProfileOptions(min_step_size=0.0005,
    delta_ratio_max=0.05,
    default_step_size=0.005,
    ratio_min=0.01)

result1_bfgs = profile.parameter_profile(
    problem=problem1,
    result=result1_bfgs,
    optimizer=optimizer_bfgs,
    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_bfgs = profile.parameter_profile(
    problem=problem1,
```

(continues on next page)

(continued from previous page)

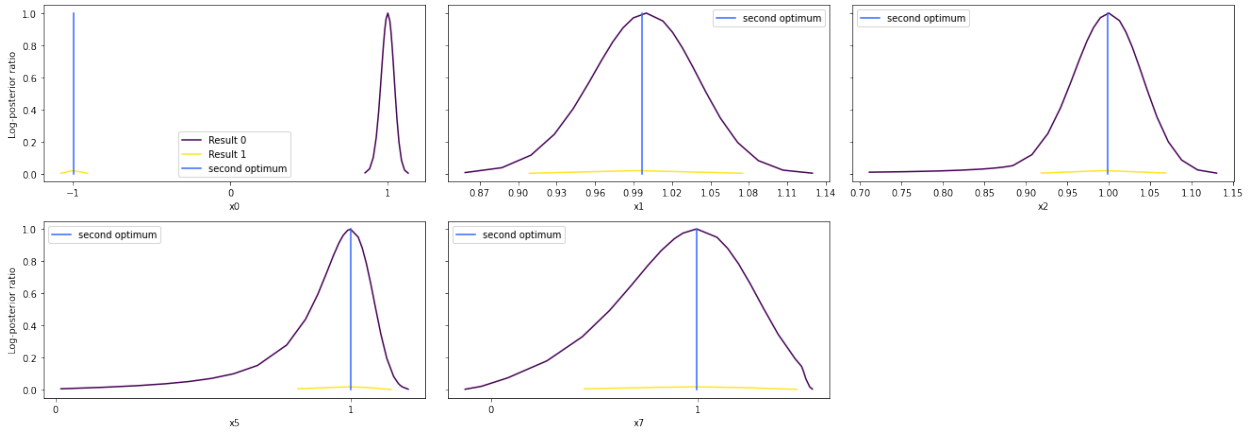
```
    result=result1_bfgs,
    optimizer=optimizer_bfgs,
    profile_index=np.array([1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0]),
    result_index=19,
    profile_options=profile_options)
```

```
CPU times: user 1.31 s, sys: 4.28 ms, total: 1.32 s
Wall time: 1.32 s
```

Visualize and analyze results

pypesto offers easy-to-use visualization routines:

[16]:
```
# specify the parameters, for which profiles should be computed
ax = visualize.profiles(result1_bfgs, profile_indices = [0,1,2,5,7],
                        reference=ref, profile_list_ids=[0, 1])
```



Approximate profiles

When computing the profiles is computationally too demanding, it is possible to employ to at least consider a normal approximation with covariance matrix given by the Hessian or FIM at the optimal parameters.

[17]:
```
%%time

result1_tnc = profile.approximate_parameter_profile(
    problem=problem1,
    result=result1_bfgs,
    profile_index=np.array([1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0]),
    result_index=0,
    n_steps=1000)
```

```
CPU times: user 25 ms, sys: 16.3 ms, total: 41.4 ms
Wall time: 24.2 ms
```

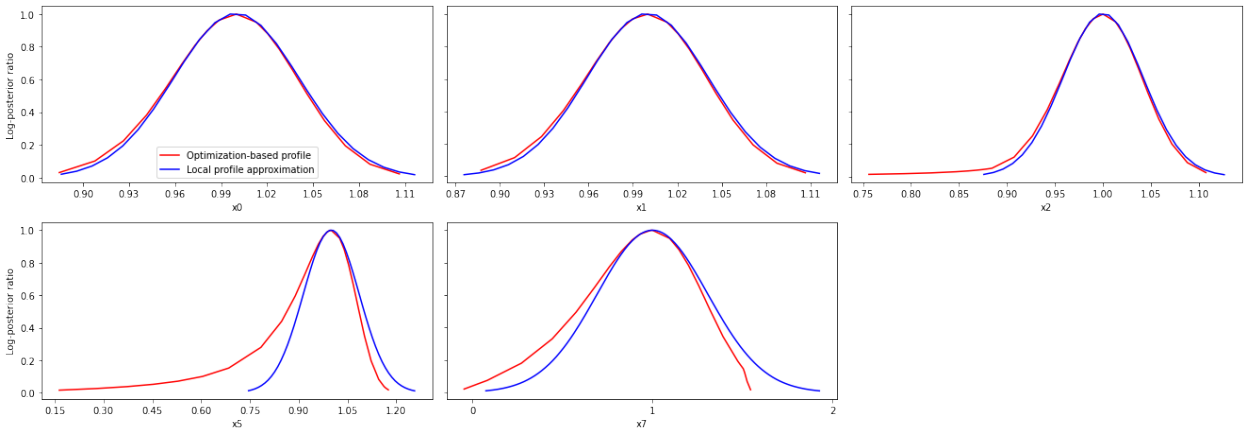
These approximate profiles require at most one additional function evaluation, can however yield substantial approximation errors:

[18]:
```
axes = visualize.profiles(
    result1_bfgs, profile_indices = [0,1,2,5,7], profile_list_ids=[0, 2],
```

(continues on next page)

(continued from previous page)

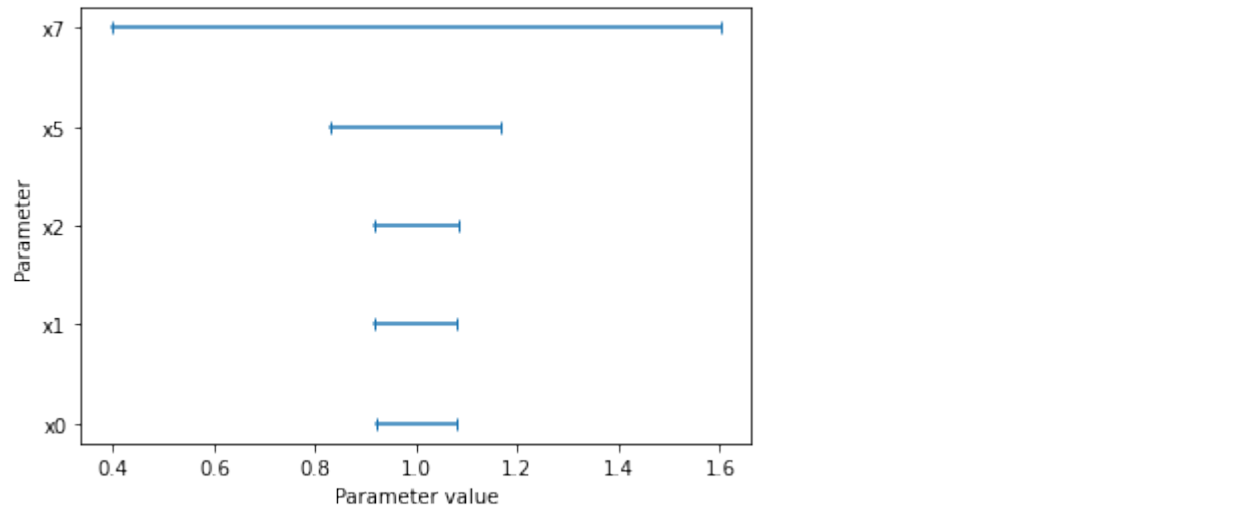
```
    ratio_min=0.01, colors=[(1,0,0,1), (0,0,1,1)],
    legends=["Optimization-based profile", "Local profile approximation"])
```



We can also plot approximate confidence intervals based on profiles:

[19]:
```
visualize.profile_cis(
    result1_bfgs, confidence_level=0.95, profile_list=2)
```

[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3bda13f10>
```



2.2 Conversion reaction

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

(continues on next page)

(continued from previous page)

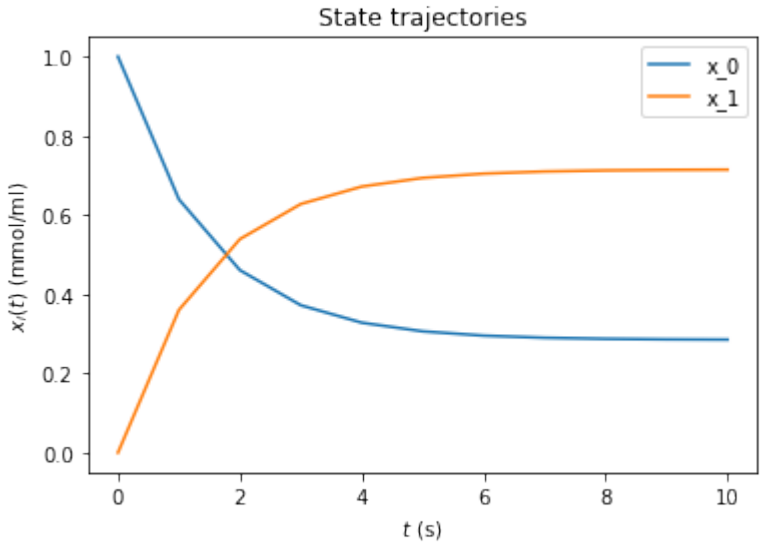
```
import pypesto.visualize as visualize

# 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(np.linspace(0, 10, 11))
model.setParameterScale(amici.ParameterScaling.log10)
model.setParameters([-0.3,-0.7])
solver = model.getSolver()
solver.setSensitivityMethod(amici.SensitivityMethod.forward)
solver.setSensitivityOrder(amici.SensitivityOrder.first)

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



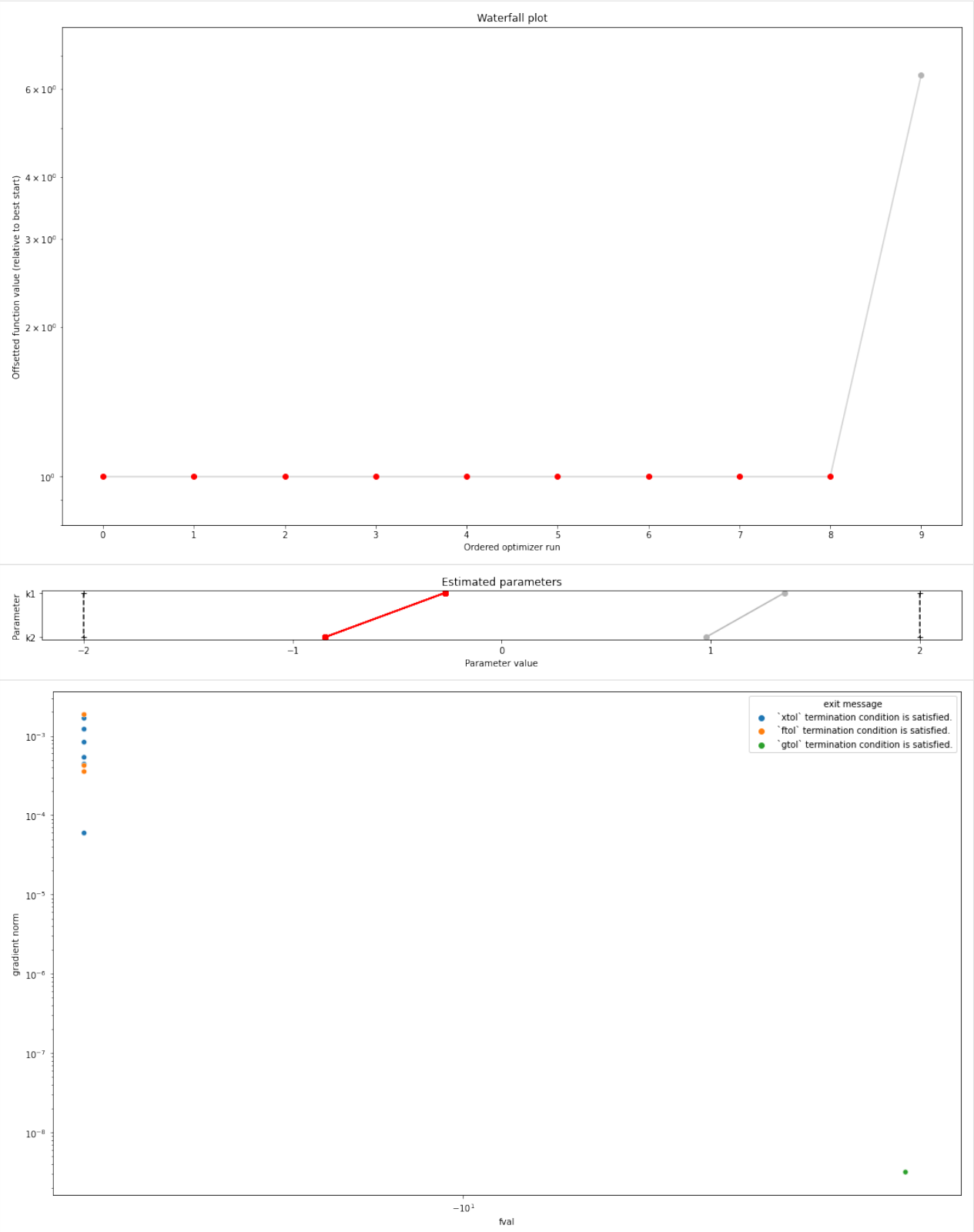
2.2.3 Optimize

[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 = optimize.ScipyOptimizer(method='ls_trf')

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

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

2.2.4 Visualize

[5]:
```
visualize.waterfall(result)
visualize.parameters(result)
visualize.optimizer_convergence(result)
```

[5]:
```
<AxesSubplot:xlabel='fval', ylabel='gradient norm'>
```

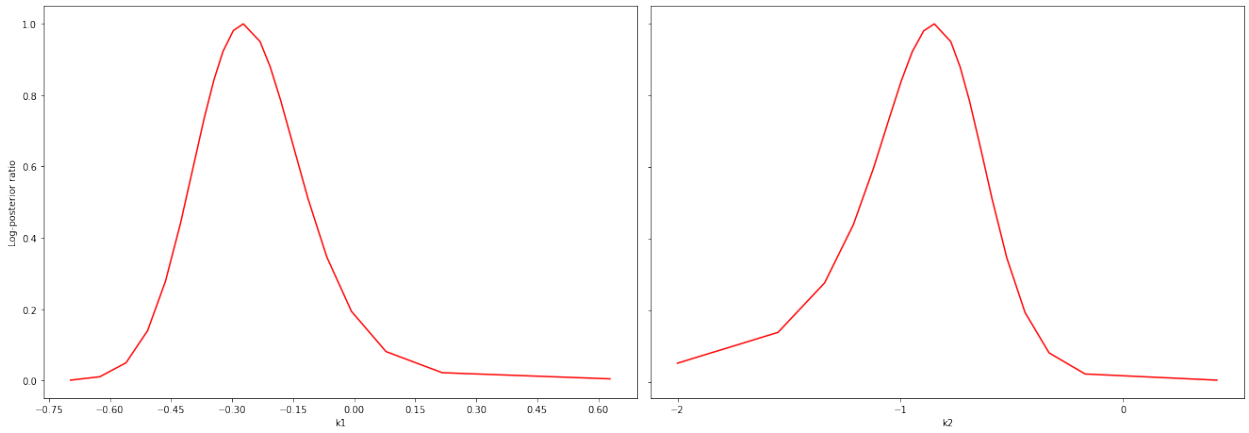

Waterfall plot
Offsetted function value (relative to best start)
Ordered optimizer run
Estimated parameters
Parameter
k1
k2
Parameter value
exit message
`xtol` termination condition is satisfied.
`ftol` termination condition is satisfied.
`gtol` termination condition is satisfied.
gradient norm
fval

2.2.5 Profiles

[6]:
```
import pypesto.profile as profile

profile_options = profile.ProfileOptions(min_step_size=0.0005,
    delta_ratio_max=0.05,
    default_step_size=0.005,
    ratio_min=0.01)

result = profile.parameter_profile(
    problem=problem,
    result=result,
    optimizer=optimizer,
    profile_index=np.array([1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0]),
    result_index=0,
    profile_options=profile_options)
```

```
Parameters obtained from history and optimizer do not match: [-0.17103023], [-0.
↪17102486]
Parameters obtained from history and optimizer do not match: [-0.92272262], [-0.
↪92270649]
```

[7]:
```
# specify the parameters, for which profiles should be computed
ax = visualize.profiles(result)
```



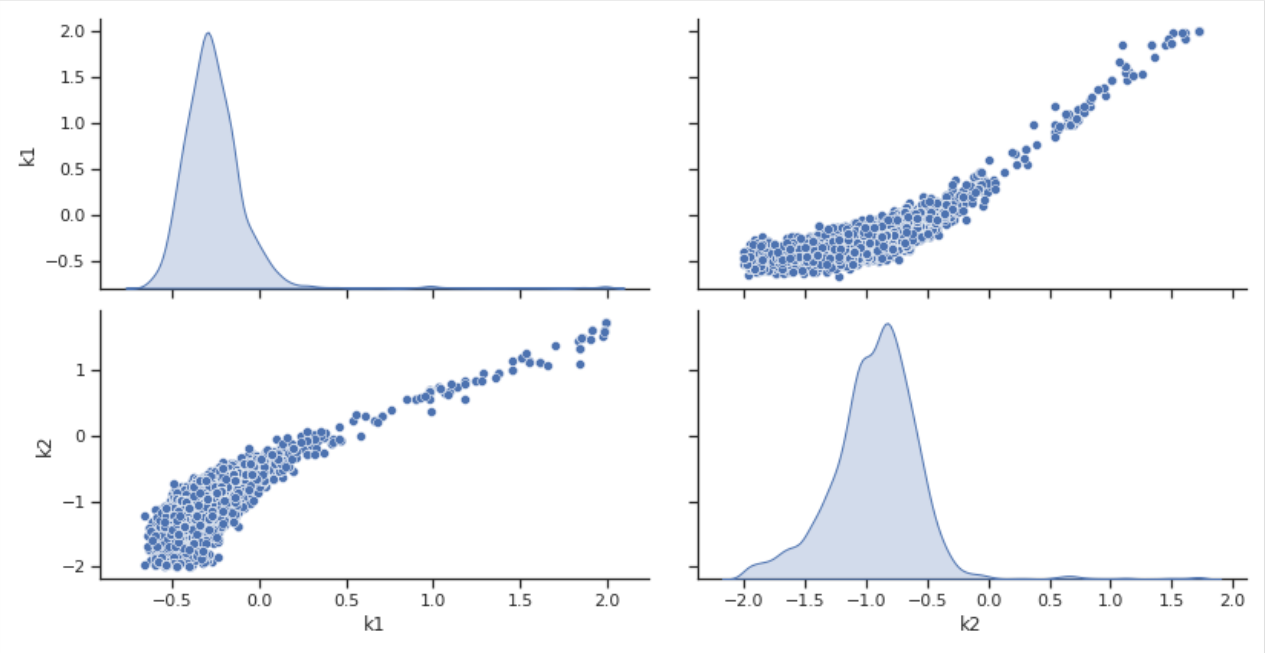
2.2.6 Sampling

[8]:
```
import pypesto.sample as sample

sampler = sample.AdaptiveParallelTemperingSampler(
    internal_sampler=sample.AdaptiveMetropolisSampler(),
    n_chains=3)

result = sample.sample(problem, n_samples=10000, sampler=sampler, result=result)
```

```
100%|| 10000/10000 [00:33<00:00, 295.74it/s]
```

[9]:
```
ax = visualize.sampling_scatter(result, size=[13,6])
```

```
Burn in index not found in the results, the full chain will be shown.
You may want to use, e.g., 'pypesto.sample.geweke_test'.
```



2.2.7 Predict

[10]:
```
# Let's create a function, which predicts the ratio of x_1 and x_0
import pypesto.predict as predict

def ratio_function(amici_output_list):
    # This (optional) function post-processes the results from AMICI and must accept␣
↪one input:
    # a list of dicts, with the fields t, x, y[, sx, sy - if sensi_orders includes 1]
    # We need to specify the simulation condition: here, we only have one, i.e., it's␣
↪[0]
    x = amici_output_list[0]['x']
    ratio = x[:,1] / x[:,0]
    # we need to output also at least one simulation condition
    return [ratio]

# create pypesto prediction function
predictor = predict.AmiciPredictor(objective, post_processor=ratio_function, output_
↪ids=['ratio'])

# run prediction
prediction = predictor(x=model.getUnscaledParameters())
```

[11]:
```
dict(prediction)
```

[11]:
```
{'conditions': [{'timepoints': array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.
↪, 10.]),
   'output_ids': ['ratio'],
   'x_names': ['k1', 'k2'],
   'output': array([0.        , 1.95196396, 2.00246152, 2.00290412, 2.00290796,
          2.00290801, 2.00290801, 2.00290799, 2.002908  , 2.00290801,
          2.002908  ]),
```

(continues on next page)

(continued from previous page)

```
   'output_sensi': None}],
 'condition_ids': ['condition_0'],
 'comment': None,
 'parameter_ids': ['k1', 'k2']}
```

2.2.8 Analyze parameter ensembles

[12]:
```
# We want to use the sample result to create a prediction
from pypesto.ensemble import ensemble

# first collect some vectors from the sampling result
vectors = result.sample_result.trace_x[0, ::20, :]

# create the collection
my_ensemble = ensemble.Ensemble(vectors,
                                x_names=problem.x_names,
                                ensemble_type=ensemble.EnsembleType.sample,
                                lower_bound=problem.lb,
                                upper_bound=problem.ub)

# we can also perform an approximative identifiability analysis
summary = my_ensemble.compute_summary()
identifiability = my_ensemble.check_identifiability()
print(identifiability.transpose())
```

```
parameterId               k1        k2
parameterId               k1        k2
lowerBound                -2        -2
upperBound                 2         2
ensemble_mean      -0.559689 -0.810604
ensemble_std        0.287255  0.364753
ensemble_median    -0.559689 -0.810604
within lb: 1 std        True      True
within ub: 1 std        True      True
within lb: 2 std        True      True
within ub: 2 std        True      True
within lb: 3 std        True      True
within ub: 3 std        True      True
within lb: perc 5       True      True
within lb: perc 20      True      True
within ub: perc 80      True      True
within ub: perc 95      True      True
```

[13]:
```
# run a prediction
ensemble_prediction = my_ensemble.predict(predictor, prediction_id='ratio_over_time')

# go for some analysis
prediction_summary = ensemble_prediction.compute_summary(percentiles_list=(1, 5, 10,␣
↪25, 75, 90, 95, 99))
dict(prediction_summary)
```

[13]:
```
{'mean': <pypesto.predict.result.PredictionResult at 0x7fe734b68760>,
 'std': <pypesto.predict.result.PredictionResult at 0x7fe734b698b0>,
 'median': <pypesto.predict.result.PredictionResult at 0x7fe734b69d60>,
 'percentile 1': <pypesto.predict.result.PredictionResult at 0x7fe734b69a30>,
```

(continues on next page)

(continued from previous page)

```
 'percentile 5': <pypesto.predict.result.PredictionResult at 0x7fe734b69a60>,
 'percentile 10': <pypesto.predict.result.PredictionResult at 0x7fe734b69640>,
 'percentile 25': <pypesto.predict.result.PredictionResult at 0x7fe734b69610>,
 'percentile 75': <pypesto.predict.result.PredictionResult at 0x7fe734b693d0>,
 'percentile 90': <pypesto.predict.result.PredictionResult at 0x7fe734b69e20>,
 'percentile 95': <pypesto.predict.result.PredictionResult at 0x7fe734b692e0>,
 'percentile 99': <pypesto.predict.result.PredictionResult at 0x7fe734b69730>}
```

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.optimize as optimize
import pypesto.visualize as 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)
```

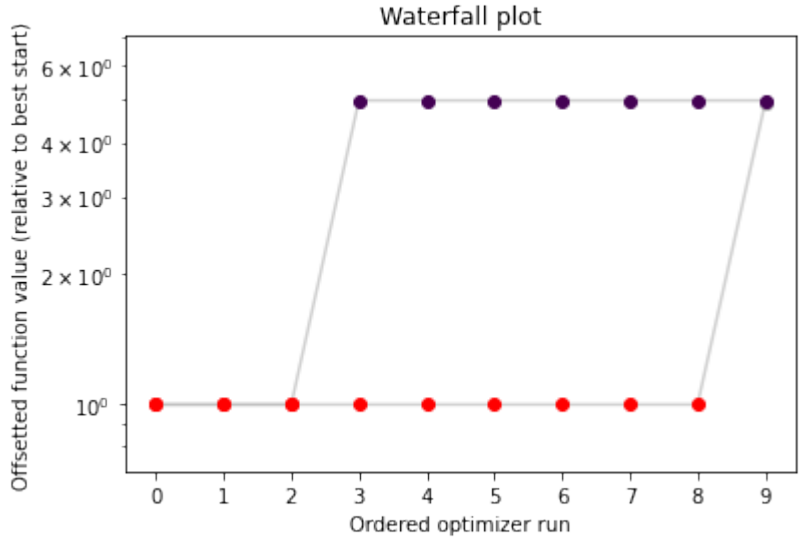
2.3.2 Optimize

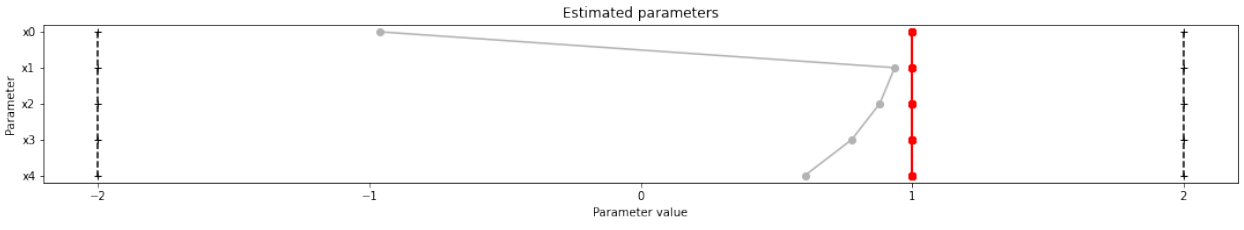
[3]:
```
optimizer = optimize.ScipyOptimizer()
n_starts = 10

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

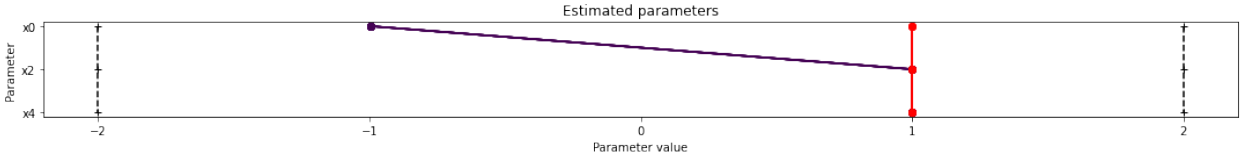
2.3.3 Visualize

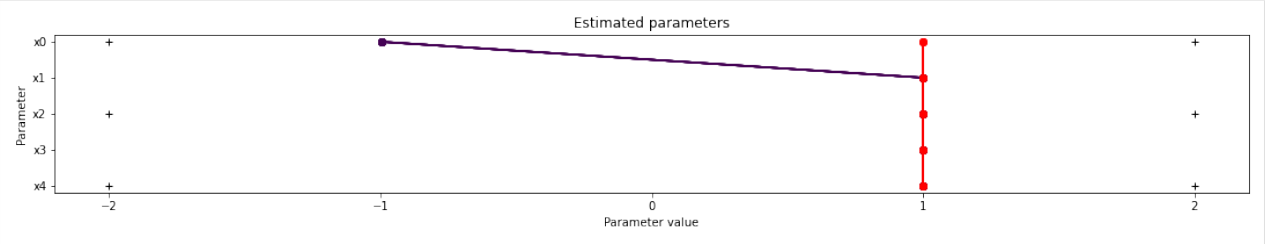
[4]:
```
fig, ax = plt.subplots()
visualize.waterfall(result1, ax)
visualize.waterfall(result2, ax)
visualize.parameters(result1)
visualize.parameters(result2)
visualize.parameters(result2, parameter_indices='all')
```

[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff3da236a20>
```









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

[5]:

	fval	x	\
0	2.563931e-14	[0.9999999859217336, 0.9999999812160436, 0.999...	
1	4.103854e-14	[1.0000000033181213, 1.00000001070042, 1.00000...	
2	2.430040e-13	[0.9999999979980921, 0.9999999872750013, 0.999...	
3	2.993261e-12	[1.0000000655628785, 1.0000002137366326, 1.000...	
4	3.028019e-11	[1.0000002263273202, 0.9999999457510741, 1.000...	
5	1.504857e-10	[1.0000008747306284, 1.000001813929941, 1.0000...	
6	3.713657e-10	[1.0000011952242212, 1.000001771893066, 1.0000...	
7	4.012393e-10	[0.9999986079063197, 0.9999988670990364, 0.999...	
8	5.247717e-10	[1.000000368254703, 1.0000009022274876, 0.9999...	
9	3.930839e+00	[-0.9620510415103535, 0.9357394330313418, 0.88...	

	grad
0	[-3.7771869883630873e-06, 3.2004378806360524e-...
1	[-1.6190347383411735e-06, 5.768553691118231e-0...
2	[3.4844693764909735e-06, 4.6873211372756083e-0...
3	[-3.291322500031286e-05, 6.600823794056182e-07...
4	[0.00020321414758544783, -0.000184783444508992...
5	[-2.4037728901340504e-05, -1.168240814877157e-...
6	[0.00024981346612303615, 0.0001962351003382311...
7	[-0.000663297051531534, 0.000537723456872972, ...
8	[-6.555069341760695e-05, 0.0009407121705420637...
9	[-1.109923131625834e-06, 5.109232684041842e-06...

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

[6]:

	fval	x	\
0	4.679771e-17	[0.9999999998641961, 1.0, 1.0000000002266116, ...	
1	4.825331e-16	[0.9999999995848748, 1.0, 0.9999999991941183, ...	
2	1.394704e-14	[1.0000000026325193, 1.0, 0.9999999987812758, ...	
3	3.989975e+00	[-0.9949747468838975, 1.0, 0.9999999999585671,...	
4	3.989975e+00	[-0.9949747461383964, 1.0, 0.9999999963588824,...	
5	3.989975e+00	[-0.9949747436177196, 1.0, 0.9999999894437084,...	
6	3.989975e+00	[-0.9949747458936441, 1.0, 0.99999997533737, 1...	
7	3.989975e+00	[-0.9949747793023977, 1.0, 1.000000023888003, ...	
8	3.989975e+00	[-0.9949748033666262, 1.0, 1.0000000080319777,...	
9	3.989975e+00	[-0.994974648260114, 1.0, 0.9999999725753793, ...	

	grad
0	[-1.0891474676223493e-07, nan, 2.2706484163692...
1	[-3.329303753527845e-07, nan, -8.0749345757971...
2	[2.1112804950914665e-06, nan, -1.2211616799204...
3	[-4.2116658605095836e-08, nan, -4.151572285811...
4	[5.468066182068299e-07, nan, -3.64839985427999...
5	[2.5380648831507813e-06, nan, -1.0577404068293...
6	[7.40153570877311e-07, nan, -2.471195460075688...

(continues on next page)

(continued from previous page)

```
7  [-2.5651750697797127e-05, nan, 2.3935779637870...
8  [-4.466176453288284e-05, nan, 8.0480417566767e...
9  [7.78676721049365e-05, nan, -2.747946901432181...
```

2.4 AMICI Python example "Boehm"

This is an example using the "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')

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()])
```

(continues on next page)

(continued from previous page)

```
    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()
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()
```

(continues on next page)

(continued from previous page)

```
# 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

model.requireSensitivitiesForAllParameters()


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


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

[15]:
```
import pypesto.optimize as optimize

# create optimizer object which contains all information for doing the optimization
optimizer = optimize.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 = optimize.minimize(problem=problem,
                           optimizer=optimizer,
                           n_starts=10) # 200
```

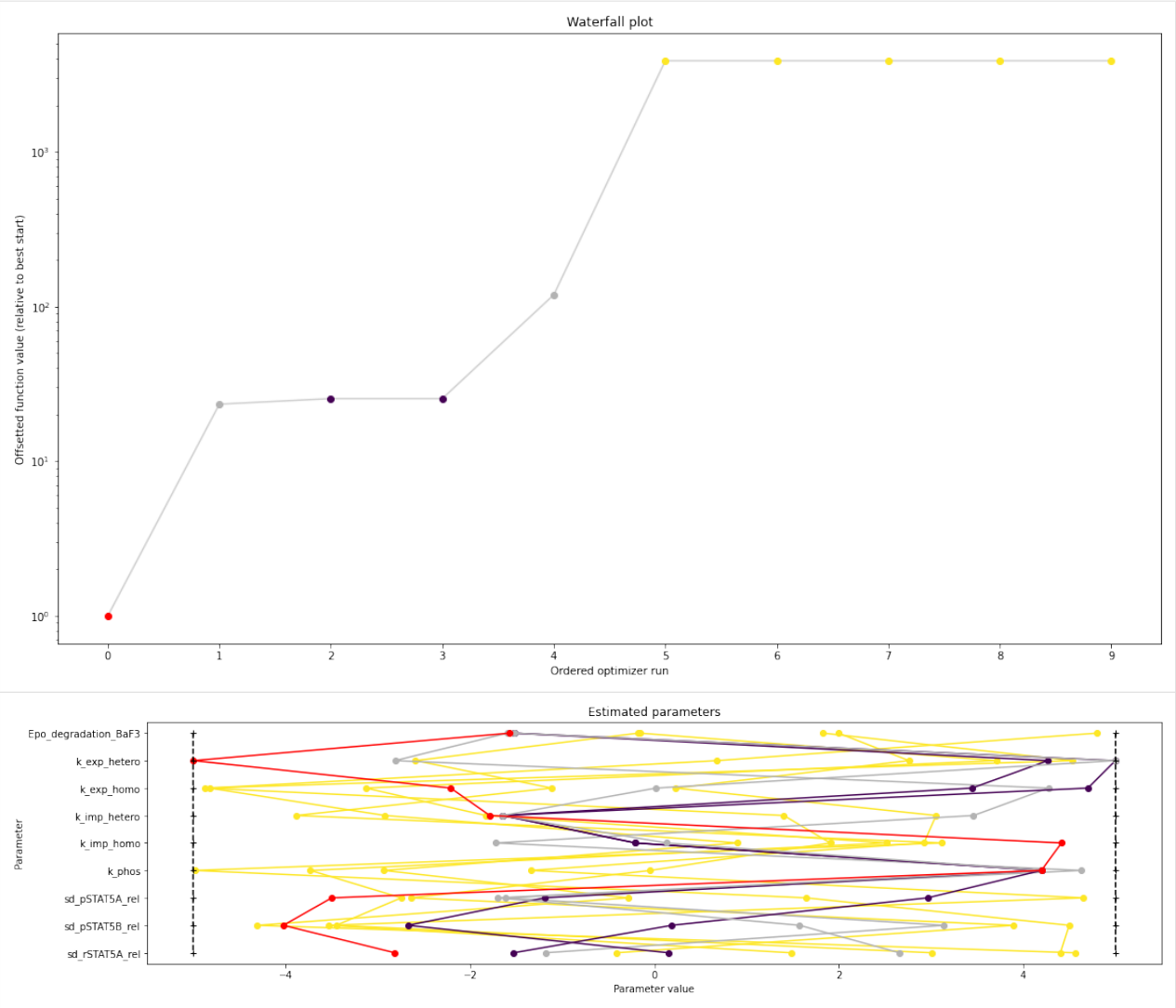
```
[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 197.098 and h = 5.31558e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 197.097676:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 153.887 and h = 3.19493e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 153.886960:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 175.27 and h = 1.75497e-05, the error test failed repeatedly or with␣
↪|h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 175.270281:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 89.6211 and h = 2.65581e-05, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 89.621132:
AMICI failed to integrate the forward problem
```

2.4.5 Visualization

Create waterfall and parameter plot

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

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

[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7feec1f59280>
```



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 pypesto.petab
import pypesto.optimize as optimize
import pypesto.visualize as visualize
import amici
import petab

import os
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

(continues on next page)

(continued from previous page)

```
!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.petab.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([[0., 0., 0.
↪, 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]]), 'rdatas': [<amici.numpy.
↪ReturnDataView object at 0x7fb394714cd0>]}
```

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

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:  0.012310244824532846
```

In short

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

[10]:
```
importer = pypesto.petab.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 = optimize.ScipyOptimizer()

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

# do the optimization
result = optimize.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 = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
```

(continues on next page)

(continued from previous page)

```
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
```

(continues on next page)

(continued from previous page)

```
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
AMICI failed to integrate the forward problem

[Warning] AMICI:CVODES:CVode:ERR_FAILURE: AMICI ERROR: in module CVODES in function␣
↪CVode : At t = 129.296 and h = 7.99525e-06, the error test failed repeatedly or␣
↪with |h| = hmin.
[Warning] AMICI:simulation: AMICI forward simulation failed at t = 129.295950:
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.

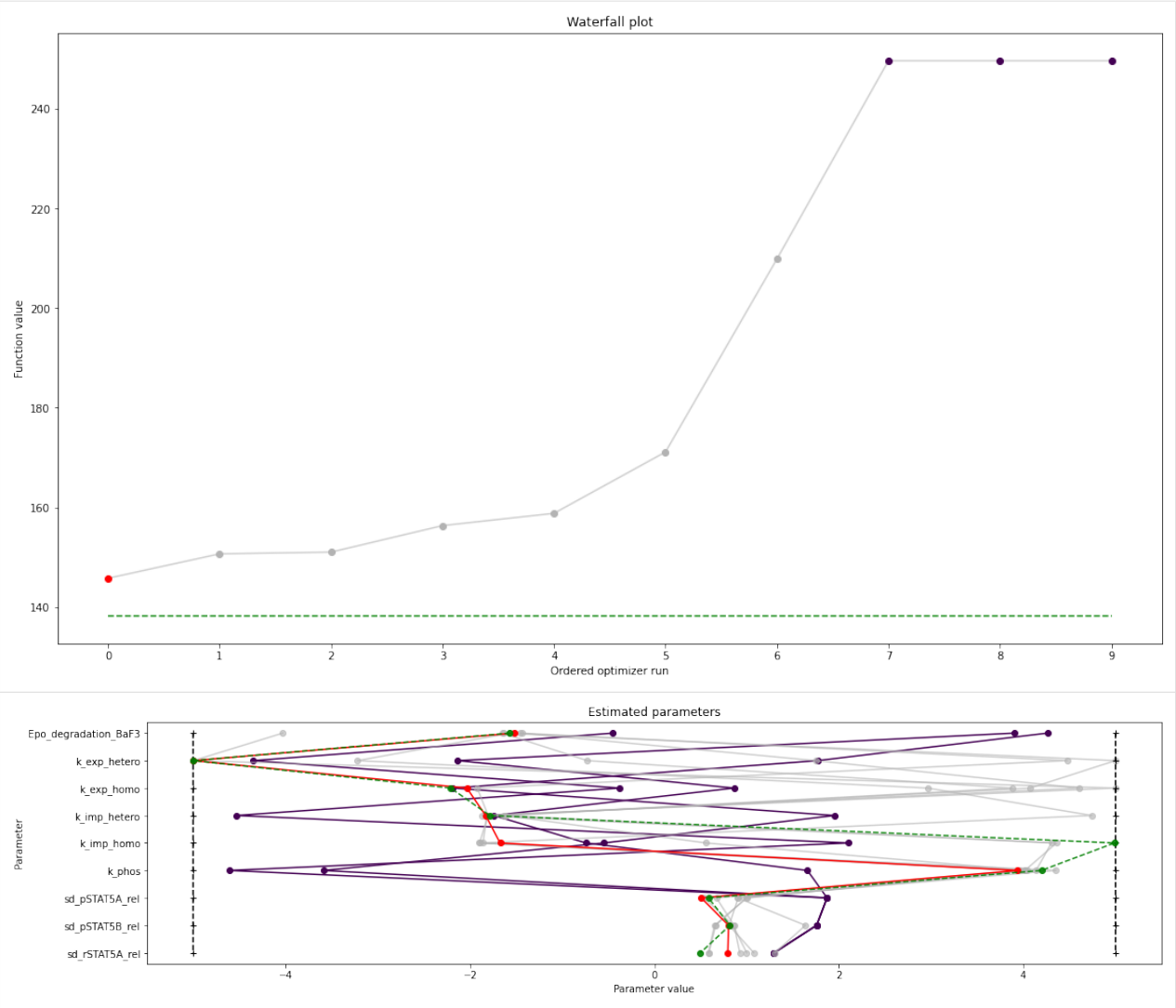
[12]:
```
result.optimize_result.get_for_key('fval')
```

[12]:
```
[145.75941164508708,
 150.66829665808604,
 151.0111203508512,
 156.3408523704166,
 158.80993946232553,
 171.1342910354579,
 209.9307084952844,
 249.7459886904473,
 249.74599725959808,
 249.7459974434843]
```

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

[13]:
```
ref = visualize.create_references(
    x=petab_problem.x_nominal_scaled, fval=obj(petab_problem.x_nominal_scaled))

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

[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb345161310>
```



2.6 Storage

This notebook illustrates how simulations and results can be saved to file.

[1]:
```
import pypesto
import pypesto.optimize as optimize
import pypesto.visualize as visualize
from pypesto.store import (save_to_hdf5, read_from_hdf5)

import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
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 = -3 * np.ones((dim_full, 1))
ub = 3 * np.ones((dim_full, 1))

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

# create optimizers
optimizer = optimize.ScipyOptimizer(method='l-bfgs-b')

# set number of starts
n_starts = 20
```

2.6.2 Objective function traces

During optimization, it is possible to regularly write the objective function trace to file. This is useful e.g. when runs fail, or for various diagnostics. Currently, pyPESTO can save histories to 3 backends: in-memory, as CSV files, or to HDF5 files.

Memory History

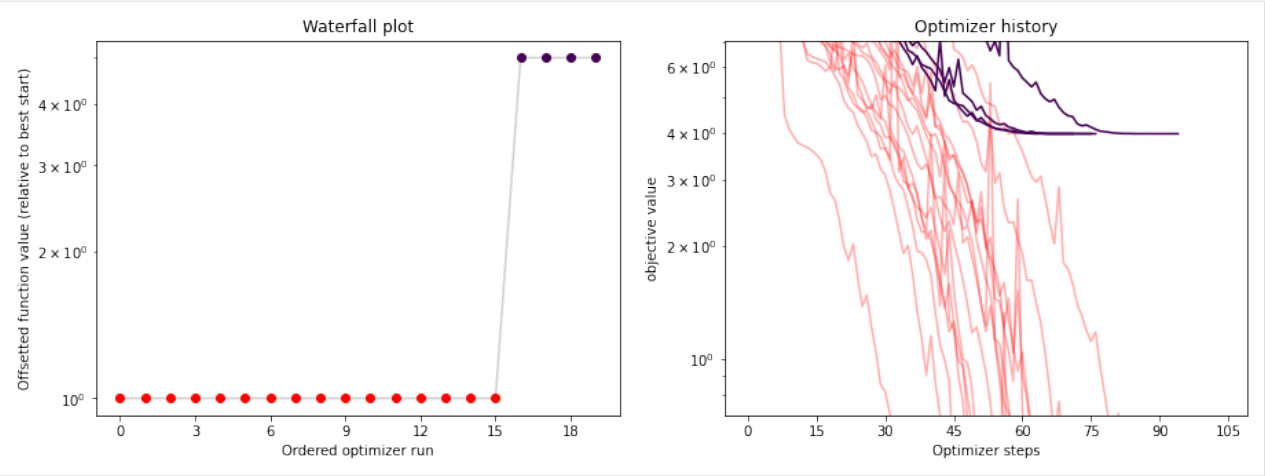
To record the history in-memory, just set `trace_record=True` in the `pypesto.HistoryOptions`. Then, the optimization result contains those histories:

[3]:
```
# record the history
history_options = pypesto.HistoryOptions(trace_record=True)

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

Now, in addition to queries on the result, we can also access the

[4]:
```
print("History type: ", type(result.optimize_result.list[0].history))
# print("Function value trace of best run: ", result.optimize_result.list[0].history.
↪get_fval_trace())

fig, ax = plt.subplots(1, 2)
visualize.waterfall(result, ax=ax[0])
visualize.optimizer_history(result, ax=ax[1])
fig.set_size_inches((15, 5))
```

```
History type:  <class 'pypesto.objective.history.MemoryHistory'>
```



CSV History

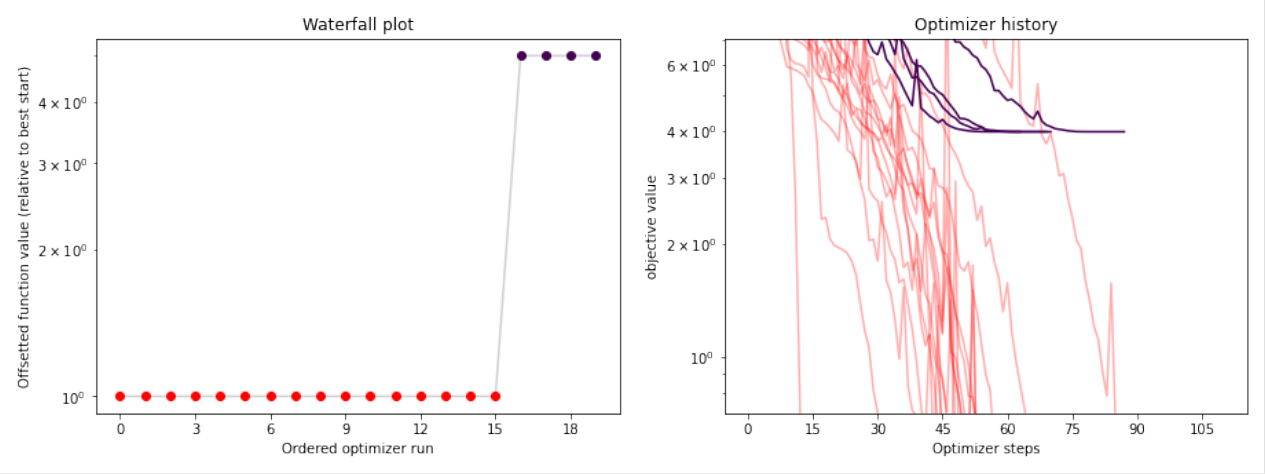
The in-memory storage is however not stored anywhere. To do that, it is possible to store either to CSV or HDF5. This is specified via the `storage_file` option. If it ends in `.csv`, a `pypesto.objective.history.CsvHistory` will be employed; if it ends in `.hdf5` a `pypesto.objective.history.Hdf5History`. Occurrences of the substring `{id}` in the filename are replaced by the multistart id, allowing to maintain a separate file per run (this is necessary for CSV as otherwise runs are overwritten).

[5]:
```
# record the history and store to CSV
history_options = pypesto.HistoryOptions(trace_record=True, storage_file='history_{id}
↪.csv')

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

Note that for this simple cost function, saving to CSV takes a considerable amount of time. This overhead decreases for more costly simulators, e.g. using ODE simulations via AMICI.

[6]:
```
print("History type: ", type(result.optimize_result.list[0].history))
# print("Function value trace of best run: ", result.optimize_result.list[0].history.
↪get_fval_trace())

fig, ax = plt.subplots(1, 2)
visualize.waterfall(result, ax=ax[0])
visualize.optimizer_history(result, ax=ax[1])
fig.set_size_inches((15, 5))
```

```
History type:  <class 'pypesto.objective.history.CsvHistory'>
```



HDF5 History

TODO: This is not fully implemented yet (it's on the way . . .).

2.6.3 Result storage

Result objects can be stored to HDF5 files. When appliable, this is preferable to just pickling results, which is not guaranteed to be reproducible in the future.

[7]:
```
# Run optimizaitons
result = optimize.minimize(
    problem=problem, optimizer=optimizer,
    n_starts=n_starts)
```

[8]:
```
result.optimize_result.list[0:2]
```

[8]:
```
[{'id': '17',
  'x': array([0.99999994, 0.99999994, 1.        , 1.00000003, 1.00000011,
         1.00000009, 1.00000002, 0.99999991, 0.99999978, 0.99999952]),
  'fval': 8.707800564711112e-12,
  'grad': array([-2.31616041e-05, -3.81308795e-05,  1.32978065e-05, -1.23392144e-05,
          6.52303854e-05,  3.58850228e-05,  1.86401788e-05, -7.46042767e-06,
          8.02520832e-06, -8.71388750e-06]),
  'hess': None,
  'res': None,
  'sres': None,
  'n_fval': 87,
  'n_grad': 87,
  'n_hess': 0,
  'n_res': 0,
  'n_sres': 0,
  'x0': array([ 1.45114268,  2.06074379,  1.64058197,  0.6213187 ,  2.28867279,
          0.20877178,  1.83054994, -0.35049857, -2.66672642, -2.75180939]),
  'fval0': 16215.296810239959,
  'history': <pypesto.objective.history.History at 0x7f8dbd6ae070>,
  'exitflag': 0,
  'time': 0.020003557205200195,
  'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
```

(continues on next page)

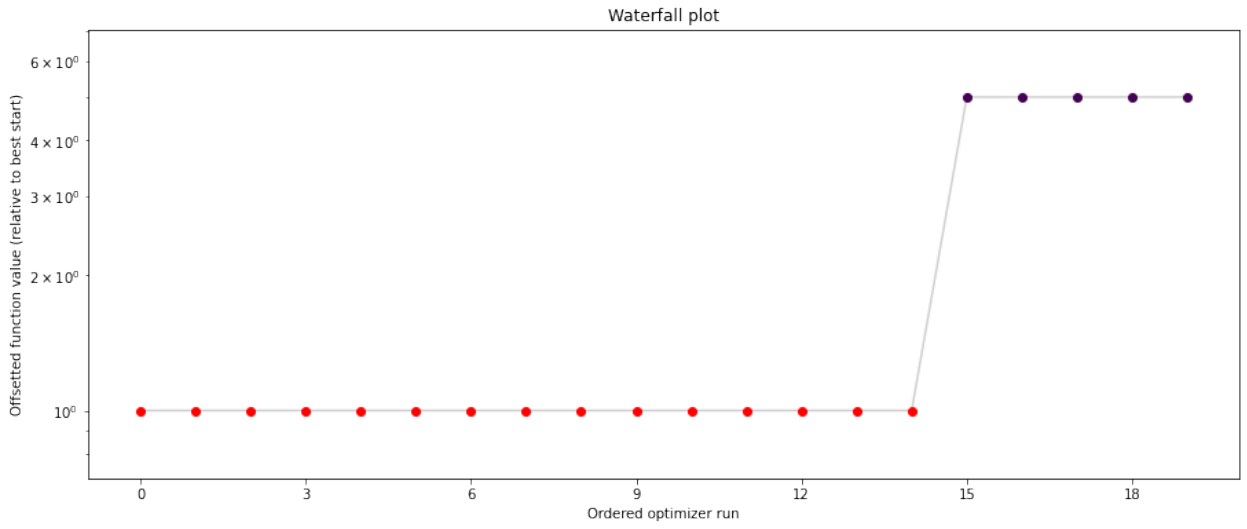
(continued from previous page)

```
{'id': '7',
 'x': array([0.99999998, 0.99999991, 0.99999996, 1.00000002, 1.00000011,
        1.00000024, 1.00000032, 1.00000046, 1.00000083, 1.00000177]),
 'fval': 1.2244681497661217e-11,
 'grad': array([ 1.82728495e-05, -6.74518178e-05, -1.27149830e-05, -2.05128948e-06,
         3.27446361e-06,  6.39483721e-05,  4.57675698e-05, -4.81356983e-06,
        -5.53900259e-05,  2.06167771e-05]),
 'hess': None,
 'res': None,
 'sres': None,
 'n_fval': 91,
 'n_grad': 91,
 'n_hess': 0,
 'n_res': 0,
 'n_sres': 0,
 'x0': array([ 0.80798177, -0.91430344, -2.6742686 , -1.76685642,  0.16784518,
         1.70273894,  0.03732323,  2.71928657,  1.29546904, -2.9200907 ]),
 'fval0': 18006.95154502575,
 'history': <pypesto.objective.history.History at 0x7f8dbd6ae6a0>,
 'exitflag': 0,
 'time': 0.024848461151123047,
 'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'}]
```

As usual, having obtained our result, we can directly perform some plots:

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

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8dbd812d00>
```



Save optimization result as HDF5 file

The optimization result can be saved via a `pypesto.store.OptimizationResultHDF5Writer`.

[10]:
```
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)
```

Read optimization result from HDF5 file

When reading in the stored result again, we recover the original optimization result:

[11]:
```
# Read result and problem
hdf5_reader = read_from_hdf5.OptimizationResultHDF5Reader(fn)
result = hdf5_reader.read()
```

[12]:
```
result.optimize_result.list[0:2]
```

[12]:
```
[{'id': '17',
  'x': array([0.99999994, 0.99999994, 1.        , 1.00000003, 1.00000011,
         1.00000009, 1.00000002, 0.99999991, 0.99999978, 0.99999952]),
  'fval': 8.707800564711112e-12,
  'grad': array([-2.31616041e-05, -3.81308795e-05,  1.32978065e-05, -1.23392144e-05,
          6.52303854e-05,  3.58850228e-05,  1.86401788e-05, -7.46042767e-06,
          8.02520832e-06, -8.71388750e-06]),
  'hess': None,
  'res': None,
  'sres': None,
  'n_fval': 87,
  'n_grad': 87,
  'n_hess': 0,
  'n_res': 0,
  'n_sres': 0,
  'x0': array([ 1.45114268,  2.06074379,  1.64058197,  0.6213187 ,  2.28867279,
          0.20877178,  1.83054994, -0.35049857, -2.66672642, -2.75180939]),
  'fval0': 16215.296810239959,
  'history': None,
  'exitflag': 0,
  'time': 0.020003557205200195,
  'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'},
 {'id': '7',
  'x': array([0.99999998, 0.99999991, 0.99999996, 1.00000002, 1.00000011,
         1.00000024, 1.00000032, 1.00000046, 1.00000083, 1.00000177]),
  'fval': 1.2244681497661217e-11,
  'grad': array([ 1.82728495e-05, -6.74518178e-05, -1.27149830e-05, -2.05128948e-06,
          3.27446361e-06,  6.39483721e-05,  4.57675698e-05, -4.81356983e-06,
         -5.53900259e-05,  2.06167771e-05]),
  'hess': None,
  'res': None,
  'sres': None,
```

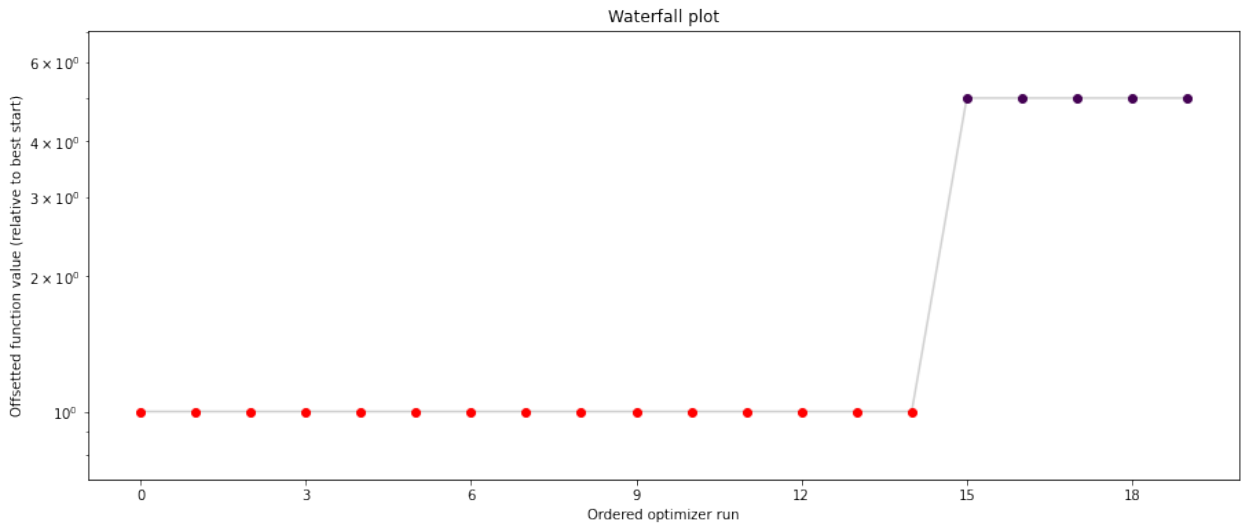
(continues on next page)

(continued from previous page)

```
  'n_fval': 91,
  'n_grad': 91,
  'n_hess': 0,
  'n_res': 0,
  'n_sres': 0,
  'x0': array([ 0.80798177, -0.91430344, -2.6742686 , -1.76685642,  0.16784518,
          1.70273894,  0.03732323,  2.71928657,  1.29546904, -2.9200907 ]),
  'fval0': 18006.95154502575,
  'history': None,
  'exitflag': 0,
  'time': 0.024848461151123047,
  'message': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'}]
```

[13]:
```
# plot waterfalls
pypesto.visualize.waterfall(result, size=(15,6))
```

[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8dbd534a60>
```



2.7 A sampler study

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

2.7.1 The pipeline

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

[1]:
```
import pypesto
import pypesto.petab
import pypesto.optimize as optimize
import pypesto.sample as sample
import pypesto.visualize as visualize
import petab
```

(continues on next page)

(continued from previous page)

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

```
2021-03-15 08:50:22.614 - amici.petab_import - INFO - Importing model ...
2021-03-15 08:50:22.617 - amici.petab_import - INFO - Model name is 'conversion_
↪reaction_0'.
Writing model code to '/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0'.
2021-03-15 08:50:22.619 - amici.petab_import - INFO - Species: 2
2021-03-15 08:50:22.619 - amici.petab_import - INFO - Global parameters: 2
2021-03-15 08:50:22.620 - amici.petab_import - INFO - Reactions: 2
2021-03-15 08:50:22.629 - amici.petab_import - INFO - Observables: 1
2021-03-15 08:50:22.632 - amici.petab_import - INFO - Sigmas: 1
2021-03-15 08:50:22.635 - amici.petab_import - DEBUG - Adding output parameters to␣
↪model: OrderedDict()
2021-03-15 08:50:22.636 - amici.petab_import - DEBUG - Adding initial assignments for␣
↪[]
2021-03-15 08:50:22.637 - amici.petab_import - DEBUG - Condition table: (1, 0)
2021-03-15 08:50:22.637 - amici.petab_import - DEBUG - Fixed parameters are []
2021-03-15 08:50:22.638 - amici.petab_import - INFO - Overall fixed parameters: 0
2021-03-15 08:50:22.639 - amici.petab_import - INFO - Variable parameters: 2
2021-03-15 08:50:22.655 - amici.sbml_import - INFO - Finished gathering local SBML␣
↪symbols       ++ (3.76E-03s)
2021-03-15 08:50:22.660 - amici.sbml_import - INFO - Finished processing SBML␣
↪parameters         ++ (1.52E-04s)
2021-03-15 08:50:22.665 - amici.sbml_import - INFO - Finished processing SBML␣
↪compartments       ++ (2.55E-04s)
2021-03-15 08:50:22.674 - amici.sbml_import - INFO - Finished processing SBML species␣
↪initials  +++ (2.79E-04s)
2021-03-15 08:50:22.679 - amici.sbml_import - INFO - Finished processing SBML rate␣
↪rules        +++ (3.09E-05s)
2021-03-15 08:50:22.680 - amici.sbml_import - INFO - Finished processing SBML species␣
↪           ++ (9.89E-03s)
2021-03-15 08:50:22.688 - amici.sbml_import - INFO - Finished processing SBML␣
↪reactions          ++ (1.64E-03s)
2021-03-15 08:50:22.693 - amici.sbml_import - INFO - Finished processing SBML rules  ␣
↪           ++ (2.03E-04s)
2021-03-15 08:50:22.698 - amici.sbml_import - INFO - Finished processing SBML initial␣
↪assignments++ (7.04E-05s)
2021-03-15 08:50:22.704 - amici.sbml_import - INFO - Finished processing SBML species␣
↪references ++ (2.42E-04s)
2021-03-15 08:50:22.704 - amici.sbml_import - INFO - Finished importing SBML        ␣
↪            + (5.85E-02s)
2021-03-15 08:50:22.758 - amici.sbml_import - INFO - Finished processing SBML␣
↪observables         + (4.68E-02s)
2021-03-15 08:50:22.772 - amici.ode_export - INFO - Finished running smart_multiply  ␣
↪           ++ (1.29E-03s)
2021-03-15 08:50:22.785 - amici.ode_export - INFO - Finished simplifying w          ␣
↪          +++ (1.97E-03s)
2021-03-15 08:50:22.785 - amici.ode_export - INFO - Finished computing w            ␣
↪           ++ (7.49E-03s)
2021-03-15 08:50:22.788 - amici.ode_export - INFO - Finished importing SbmlImporter  ␣
↪            + (2.05E-02s)
```

(continues on next page)

(continued from previous page)

```
2021-03-15 08:50:22.810 - amici.ode_export - INFO - Finished simplifying Jy
↪          ++++ (6.82E-03s)
2021-03-15 08:50:22.810 - amici.ode_export - INFO - Finished computing Jy
↪           +++ (1.05E-02s)
2021-03-15 08:50:22.832 - amici.ode_export - INFO - Finished writing Jy.cpp
↪            ++ (3.50E-02s)
2021-03-15 08:50:22.854 - amici.ode_export - INFO - Finished running smart_jacobian
↪          ++++ (1.02E-02s)
2021-03-15 08:50:22.863 - amici.ode_export - INFO - Finished simplifying dJydsigmay
↪          ++++ (3.55E-03s)
2021-03-15 08:50:22.863 - amici.ode_export - INFO - Finished computing dJydsigmay
↪           +++ (2.29E-02s)
2021-03-15 08:50:22.867 - amici.ode_export - INFO - Finished writing dJydsigmay.cpp
↪            ++ (3.04E-02s)
2021-03-15 08:50:22.884 - amici.ode_export - INFO - Finished running smart_jacobian
↪          ++++ (5.70E-03s)
2021-03-15 08:50:22.895 - amici.ode_export - INFO - Finished simplifying dJydy
↪          ++++ (5.63E-03s)
2021-03-15 08:50:22.897 - amici.ode_export - INFO - Finished computing dJydy
↪           +++ (2.16E-02s)
2021-03-15 08:50:22.904 - amici.ode_export - INFO - Finished writing dJydy.cpp
↪            ++ (3.23E-02s)
2021-03-15 08:50:22.921 - amici.ode_export - INFO - Finished simplifying root
↪          ++++ (7.09E-05s)
2021-03-15 08:50:22.922 - amici.ode_export - INFO - Finished computing root
↪           +++ (6.78E-03s)
2021-03-15 08:50:22.923 - amici.ode_export - INFO - Finished writing root.cpp
↪            ++ (1.28E-02s)
2021-03-15 08:50:22.944 - amici.ode_export - INFO - Finished running smart_jacobian
↪          ++++ (3.90E-03s)
2021-03-15 08:50:22.953 - amici.ode_export - INFO - Finished simplifying dwdp
↪          ++++ (1.53E-03s)
2021-03-15 08:50:22.956 - amici.ode_export - INFO - Finished computing dwdp
↪           +++ (2.09E-02s)
2021-03-15 08:50:22.964 - amici.ode_export - INFO - Finished writing dwdp.cpp
↪            ++ (3.48E-02s)
2021-03-15 08:50:22.991 - amici.ode_export - INFO - Finished running smart_jacobian
↪          ++++ (4.88E-03s)
2021-03-15 08:50:22.997 - amici.ode_export - INFO - Finished simplifying dwdx
↪          ++++ (1.64E-03s)
2021-03-15 08:50:23.000 - amici.ode_export - INFO - Finished computing dwdx
↪           +++ (2.08E-02s)
2021-03-15 08:50:23.006 - amici.ode_export - INFO - Finished writing dwdx.cpp
↪            ++ (3.27E-02s)
2021-03-15 08:50:23.022 - amici.ode_export - INFO - Finished running smart_jacobian
↪          ++++ (1.28E-04s)
2021-03-15 08:50:23.028 - amici.ode_export - INFO - Finished simplifying dwdw
↪          ++++ (8.05E-05s)
2021-03-15 08:50:23.029 - amici.ode_export - INFO - Finished computing dwdw
↪           +++ (1.16E-02s)
2021-03-15 08:50:23.032 - amici.ode_export - INFO - Finished writing dwdw.cpp
↪            ++ (2.06E-02s)
2021-03-15 08:50:23.056 - amici.ode_export - INFO - Finished simplifying xdot
↪          ++++ (2.48E-03s)
2021-03-15 08:50:23.057 - amici.ode_export - INFO - Finished computing xdot
↪           +++ (1.02E-02s)
2021-03-15 08:50:23.066 - amici.ode_export - INFO - Finished writing dxdotdw.cpp
↪            ++ (2.14E-02s)
```

(continues on next page)

(continued from previous page)

```
2021-03-15 08:50:23.089 - amici.ode_export - INFO - Finished running smart_jacobian
↪         ++++ (2.54E-03s)
2021-03-15 08:50:23.100 - amici.ode_export - INFO - Finished simplifying dxdotdx_
↪explicit      ++++ (1.42E-04s)
2021-03-15 08:50:23.102 - amici.ode_export - INFO - Finished computing dxdotdx_
↪explicit         +++ (2.35E-02s)
2021-03-15 08:50:23.104 - amici.ode_export - INFO - Finished writing dxdotdx_explicit.
↪cpp        ++ (3.17E-02s)
2021-03-15 08:50:23.133 - amici.ode_export - INFO - Finished running smart_jacobian
↪         ++++ (2.19E-04s)
2021-03-15 08:50:23.142 - amici.ode_export - INFO - Finished simplifying dxdotdp_
↪explicit      ++++ (1.24E-04s)
2021-03-15 08:50:23.143 - amici.ode_export - INFO - Finished computing dxdotdp_
↪explicit         +++ (1.97E-02s)
2021-03-15 08:50:23.146 - amici.ode_export - INFO - Finished writing dxdotdp_explicit.
↪cpp        ++ (3.14E-02s)
2021-03-15 08:50:23.179 - amici.ode_export - INFO - Finished simplifying y
↪       ++++++ (4.50E-05s)
2021-03-15 08:50:23.179 - amici.ode_export - INFO - Finished computing y
↪        +++++ (4.03E-03s)
2021-03-15 08:50:23.188 - amici.ode_export - INFO - Finished running smart_jacobian
↪        +++++ (8.90E-04s)
2021-03-15 08:50:23.194 - amici.ode_export - INFO - Finished simplifying dydx
↪        +++++ (6.74E-05s)
2021-03-15 08:50:23.195 - amici.ode_export - INFO - Finished computing dydx
↪         ++++ (2.33E-02s)
2021-03-15 08:50:23.211 - amici.ode_export - INFO - Finished running smart_jacobian
↪        +++++ (1.15E-04s)
2021-03-15 08:50:23.218 - amici.ode_export - INFO - Finished simplifying dydw
↪        +++++ (9.86E-05s)
2021-03-15 08:50:23.219 - amici.ode_export - INFO - Finished computing dydw
↪         ++++ (1.33E-02s)
2021-03-15 08:50:23.225 - amici.ode_export - INFO - Finished simplifying dydx
↪         ++++ (8.66E-05s)
2021-03-15 08:50:23.226 - amici.ode_export - INFO - Finished computing dydx
↪          +++ (5.92E-02s)
2021-03-15 08:50:23.230 - amici.ode_export - INFO - Finished writing dydx.cpp
↪           ++ (6.91E-02s)
2021-03-15 08:50:23.249 - amici.ode_export - INFO - Finished running smart_jacobian
↪        +++++ (1.44E-04s)
2021-03-15 08:50:23.256 - amici.ode_export - INFO - Finished simplifying dydp
↪        +++++ (6.76E-05s)
2021-03-15 08:50:23.257 - amici.ode_export - INFO - Finished computing dydp
↪         ++++ (1.17E-02s)
2021-03-15 08:50:23.262 - amici.ode_export - INFO - Finished simplifying dydp
↪         ++++ (6.56E-05s)
2021-03-15 08:50:23.262 - amici.ode_export - INFO - Finished computing dydp
↪          +++ (2.19E-02s)
2021-03-15 08:50:23.263 - amici.ode_export - INFO - Finished writing dydp.cpp
↪           ++ (2.69E-02s)
2021-03-15 08:50:23.278 - amici.ode_export - INFO - Finished simplifying sigmay
↪        +++++ (4.36E-05s)
2021-03-15 08:50:23.278 - amici.ode_export - INFO - Finished computing sigmay
↪         ++++ (3.84E-03s)
2021-03-15 08:50:23.283 - amici.ode_export - INFO - Finished running smart_jacobian
↪         ++++ (2.24E-04s)
2021-03-15 08:50:23.288 - amici.ode_export - INFO - Finished simplifying dsigmaydp
↪         ++++ (6.41E-05s)
```

(continues on next page)

(continued from previous page)

```
2021-03-15 08:50:23.289 - amici.ode_export - INFO - Finished computing dsigmaydp
↪          +++ (1.77E-02s)
2021-03-15 08:50:23.290 - amici.ode_export - INFO - Finished writing dsigmaydp.cpp
↪           ++ (2.23E-02s)
2021-03-15 08:50:23.299 - amici.ode_export - INFO - Finished writing sigmay.cpp
↪           ++ (1.19E-03s)
2021-03-15 08:50:23.308 - amici.ode_export - INFO - Finished computing stau
↪          +++ (1.14E-04s)
2021-03-15 08:50:23.309 - amici.ode_export - INFO - Finished writing stau.cpp
↪           ++ (4.80E-03s)
2021-03-15 08:50:23.317 - amici.ode_export - INFO - Finished computing deltasx
↪          +++ (1.12E-04s)
2021-03-15 08:50:23.318 - amici.ode_export - INFO - Finished writing deltasx.cpp
↪           ++ (4.36E-03s)
2021-03-15 08:50:23.324 - amici.ode_export - INFO - Finished writing w.cpp
↪           ++ (1.32E-03s)
2021-03-15 08:50:23.336 - amici.ode_export - INFO - Finished simplifying x0
↪         ++++ (4.87E-05s)
2021-03-15 08:50:23.337 - amici.ode_export - INFO - Finished computing x0
↪          +++ (4.14E-03s)
2021-03-15 08:50:23.339 - amici.ode_export - INFO - Finished writing x0.cpp
↪           ++ (9.07E-03s)
2021-03-15 08:50:23.351 - amici.ode_export - INFO - Finished simplifying x0_
↪fixedParameters    ++++ (3.09E-05s)
2021-03-15 08:50:23.352 - amici.ode_export - INFO - Finished computing x0_
↪fixedParameters       +++ (4.07E-03s)
2021-03-15 08:50:23.353 - amici.ode_export - INFO - Finished writing x0_
↪fixedParameters.cpp      ++ (9.17E-03s)
2021-03-15 08:50:23.364 - amici.ode_export - INFO - Finished running smart_jacobian
↪         ++++ (2.02E-04s)
2021-03-15 08:50:23.369 - amici.ode_export - INFO - Finished simplifying sx0
↪         ++++ (8.59E-05s)
2021-03-15 08:50:23.370 - amici.ode_export - INFO - Finished computing sx0
↪          +++ (9.57E-03s)
2021-03-15 08:50:23.371 - amici.ode_export - INFO - Finished writing sx0.cpp
↪           ++ (1.38E-02s)
2021-03-15 08:50:23.383 - amici.ode_export - INFO - Finished running smart_jacobian
↪         ++++ (4.07E-05s)
2021-03-15 08:50:23.390 - amici.ode_export - INFO - Finished running smart_jacobian
↪         ++++ (6.25E-05s)
2021-03-15 08:50:23.396 - amici.ode_export - INFO - Finished simplifying sx0_
↪fixedParameters   ++++ (4.51E-05s)
2021-03-15 08:50:23.397 - amici.ode_export - INFO - Finished computing sx0_
↪fixedParameters      +++ (1.78E-02s)
2021-03-15 08:50:23.398 - amici.ode_export - INFO - Finished writing sx0_
↪fixedParameters.cpp     ++ (2.25E-02s)
2021-03-15 08:50:23.407 - amici.ode_export - INFO - Finished writing xdot.cpp
↪           ++ (2.42E-03s)
2021-03-15 08:50:23.412 - amici.ode_export - INFO - Finished writing y.cpp
↪           ++ (5.83E-04s)
2021-03-15 08:50:23.425 - amici.ode_export - INFO - Finished simplifying x_rdata
↪         ++++ (6.01E-05s)
2021-03-15 08:50:23.425 - amici.ode_export - INFO - Finished computing x_rdata
↪          +++ (4.55E-03s)
2021-03-15 08:50:23.427 - amici.ode_export - INFO - Finished writing x_rdata.cpp
↪           ++ (9.85E-03s)
2021-03-15 08:50:23.442 - amici.ode_export - INFO - Finished simplifying total_cl
↪         ++++ (3.69E-05s)
```

(continues on next page)

(continued from previous page)

```
2021-03-15 08:50:23.442 - amici.ode_export - INFO - Finished computing total_cl
↪          +++ (4.29E-03s)
2021-03-15 08:50:23.443 - amici.ode_export - INFO - Finished writing total_cl.cpp
↪           ++ (9.21E-03s)
2021-03-15 08:50:23.459 - amici.ode_export - INFO - Finished simplifying x_solver
↪         ++++ (7.44E-05s)
2021-03-15 08:50:23.460 - amici.ode_export - INFO - Finished computing x_solver
↪          +++ (5.19E-03s)
2021-03-15 08:50:23.462 - amici.ode_export - INFO - Finished writing x_solver.cpp
↪           ++ (1.14E-02s)
2021-03-15 08:50:23.472 - amici.ode_export - INFO - Finished generating cpp code
↪            + (6.78E-01s)
2021-03-15 08:50:55.810 - amici.ode_export - INFO - Finished compiling cpp code
↪            + (3.23E+01s)
2021-03-15 08:50:55.813 - amici.petab_import - INFO - Finished Importing PEtab model
↪              (3.32E+01s)
running build_ext
Changed extra_compile_args for unix to ['-fopenmp', '-std=c++14']
building 'conversion_reaction_0._conversion_reaction_0' extension
Testing SWIG executable swig4.0... FAILED.
Testing SWIG executable swig3.0... SUCCEEDED.
swigging swig/conversion_reaction_0.i to swig/conversion_reaction_0_wrap.cpp
swig3.0 -python -c++ -modern -outdir conversion_reaction_0 -I/usr/local/lib/python3.8/
↪dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/swig -I/usr/local/lib/
↪python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/include -o swig/
↪conversion_reaction_0_wrap.cpp swig/conversion_reaction_0.i
creating build
creating build/temp.linux-x86_64-3.8
creating build/temp.linux-x86_64-3.8/swig
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c swig/conversion_reaction_0_wrap.cpp -o build/temp.linux-x86_64-3.8/swig/
↪conversion_reaction_0_wrap.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_Jy.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_0_
↪Jy.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_deltasx.cpp -o build/temp.linux-x86_64-3.8/conversion_
```

(continues on next page)

(continued from previous page)

```
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dxdotdp_explicit_rowvals.cpp -o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_dxdotdp_explicit_rowvals.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dxdotdw.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_dxdotdw.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dwdp_rowvals.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_dwdp_rowvals.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_w.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_0_
↪w.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dydx.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_
↪0_dydx.o -fopenmp -std=c++14
```

(continues on next page)

(continued from previous page)

```
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_stau.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_
↪0_stau.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_x0.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_0_
↪x0.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_total_cl.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_total_cl.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dwdw_colptrs.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_dwdw_colptrs.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dJydsigmay.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_dJydsigmay.o -fopenmp -std=c++14
```

(continues on next page)

(continued from previous page)

```
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dxdotdx_explicit.cpp -o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_dxdotdx_explicit.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dydp.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_
↪0_dydp.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_sigmay.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_sigmay.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_0.o -
↪fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dJydy_colptrs.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_dJydy_colptrs.o -fopenmp -std=c++14
```

(continues on next page)

(continued from previous page)

```
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dxdotdx_explicit_colptrs.cpp -o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_dxdotdx_explicit_colptrs.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_xdot.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_
↪0_xdot.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dwdx.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_
↪0_dwdx.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dwdp.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_
↪0_dwdp.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dwdw.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_
↪0_dwdw.o -fopenmp -std=c++14
```

(continues on next page)

(continued from previous page)

```
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_y.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_0_
↪y.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_x_solver.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_x_solver.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dJydy_rowvals.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_dJydy_rowvals.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_sx0.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_
↪0_sx0.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dsigmaydp.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_dsigmaydp.o -fopenmp -std=c++14
```

(continues on next page)

(continued from previous page)

```
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c wrapfunctions.cpp -o build/temp.linux-x86_64-3.8/wrapfunctions.o -fopenmp -
↪std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dxdotdx_explicit_rowvals.cpp -o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_dxdotdx_explicit_rowvals.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dxdotdw_colptrs.cpp -o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_dxdotdw_colptrs.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_x_rdata.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_x_rdata.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dxdotdp_explicit.cpp -o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_dxdotdp_explicit.o -fopenmp -std=c++14
```

(continues on next page)

(continued from previous page)

```
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_sx0_fixedParameters.cpp -o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_sx0_fixedParameters.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dxdotdp_explicit_colptrs.cpp -o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_dxdotdp_explicit_colptrs.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dwdx_colptrs.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_dwdx_colptrs.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dwdp_colptrs.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_dwdp_colptrs.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_x0_fixedParameters.cpp -o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_x0_fixedParameters.o -fopenmp -std=c++14
```

(continues on next page)

(continued from previous page)

```
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dxdotdw_rowvals.cpp -o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_dxdotdw_rowvals.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dJydy.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_dJydy.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dwdx_rowvals.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_dwdx_rowvals.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_root.cpp -o build/temp.linux-x86_64-3.8/conversion_reaction_
↪0_root.o -fopenmp -std=c++14
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -
↪O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-security -g -fwrapv -
↪O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time -D_
↪FORTIFY_SOURCE=2 -fPIC -I/home/elba/Downloads/pyPESTO/doc/example/amici_models/
↪conversion_reaction_0 -I/usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-
↪linux-x86_64.egg/amici/include -I/usr/local/lib/python3.8/dist-packages/amici-0.11.
↪13-py3.8-linux-x86_64.egg/amici/ThirdParty/gsl -I/usr/local/lib/python3.8/dist-
↪packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/ThirdParty/sundials/include -I/
↪usr/local/lib/python3.8/dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/usr/include/python3.8 -
↪c conversion_reaction_0_dwdw_rowvals.cpp -o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_dwdw_rowvals.o -fopenmp -std=c++14
```

(continues on next page)

(continued from previous page)

```
x86_64-linux-gnu-g++ -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-Bsymbolic-
↪functions -Wl,-z,relro -g -fwrapv -O2 -Wl,-Bsymbolic-functions -Wl,-z,relro -g -
↪fwrapv -O2 -g -fstack-protector-strong -Wformat -Werror=format-security -Wdate-time␣
↪-D_FORTIFY_SOURCE=2 build/temp.linux-x86_64-3.8/swig/conversion_reaction_0_wrap.o␣
↪build/temp.linux-x86_64-3.8/conversion_reaction_0_Jy.o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_deltasx.o build/temp.linux-x86_64-3.8/conversion_reaction_0_
↪dxdotdp_explicit_rowvals.o build/temp.linux-x86_64-3.8/conversion_reaction_0_
↪dxdotdw.o build/temp.linux-x86_64-3.8/conversion_reaction_0_dwdp_rowvals.o build/
↪temp.linux-x86_64-3.8/conversion_reaction_0_w.o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_dydx.o build/temp.linux-x86_64-3.8/conversion_reaction_0_stau.
↪o build/temp.linux-x86_64-3.8/conversion_reaction_0_x0.o build/temp.linux-x86_64-3.
↪8/conversion_reaction_0_total_cl.o build/temp.linux-x86_64-3.8/conversion_reaction_
↪0_dwdw_colptrs.o build/temp.linux-x86_64-3.8/conversion_reaction_0_dJydsigmay.o␣
↪build/temp.linux-x86_64-3.8/conversion_reaction_0_dxdotdx_explicit.o build/temp.
↪linux-x86_64-3.8/conversion_reaction_0_dydp.o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_sigmay.o build/temp.linux-x86_64-3.8/conversion_reaction_0.o␣
↪build/temp.linux-x86_64-3.8/conversion_reaction_0_dJydy_colptrs.o build/temp.linux-
↪x86_64-3.8/conversion_reaction_0_dxdotdx_explicit_colptrs.o build/temp.linux-x86_64-
↪3.8/conversion_reaction_0_xdot.o build/temp.linux-x86_64-3.8/conversion_reaction_0_
↪dwdx.o build/temp.linux-x86_64-3.8/conversion_reaction_0_dwdp.o build/temp.linux-
↪x86_64-3.8/conversion_reaction_0_dwdw.o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_y.o build/temp.linux-x86_64-3.8/conversion_reaction_0_x_solver.o build/
↪temp.linux-x86_64-3.8/conversion_reaction_0_dJydy_rowvals.o build/temp.linux-x86_64-
↪3.8/conversion_reaction_0_sx0.o build/temp.linux-x86_64-3.8/conversion_reaction_0_
↪dsigmaydp.o build/temp.linux-x86_64-3.8/wrapfunctions.o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_dxdotdx_explicit_rowvals.o build/temp.linux-x86_64-3.8/
↪conversion_reaction_0_dxdotdw_colptrs.o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_x_rdata.o build/temp.linux-x86_64-3.8/conversion_reaction_0_dxdotdp_
↪explicit.o build/temp.linux-x86_64-3.8/conversion_reaction_0_sx0_fixedParameters.o␣
↪build/temp.linux-x86_64-3.8/conversion_reaction_0_dxdotdp_explicit_colptrs.o build/
↪temp.linux-x86_64-3.8/conversion_reaction_0_dwdx_colptrs.o build/temp.linux-x86_64-
↪3.8/conversion_reaction_0_dwdp_colptrs.o build/temp.linux-x86_64-3.8/conversion_
↪reaction_0_x0_fixedParameters.o build/temp.linux-x86_64-3.8/conversion_reaction_0_
↪dxdotdw_rowvals.o build/temp.linux-x86_64-3.8/conversion_reaction_0_dJydy.o build/
↪temp.linux-x86_64-3.8/conversion_reaction_0_dwdx_rowvals.o build/temp.linux-x86_64-
↪3.8/conversion_reaction_0_root.o build/temp.linux-x86_64-3.8/conversion_reaction_0_
↪dwdw_rowvals.o -L/usr/lib/x86_64-linux-gnu/hdf5/serial -L/usr/local/lib/python3.8/
↪dist-packages/amici-0.11.13-py3.8-linux-x86_64.egg/amici/libs -lamici -lsundials -
↪lsuitesparse -lcblas -lhdf5_hl_cpp -lhdf5_hl -lhdf5_cpp -lhdf5 -o /home/elba/
↪Downloads/pyPESTO/doc/example/amici_models/conversion_reaction_0/conversion_
↪reaction_0/_conversion_reaction_0.cpython-38-x86_64-linux-gnu.so -fopenmp
```

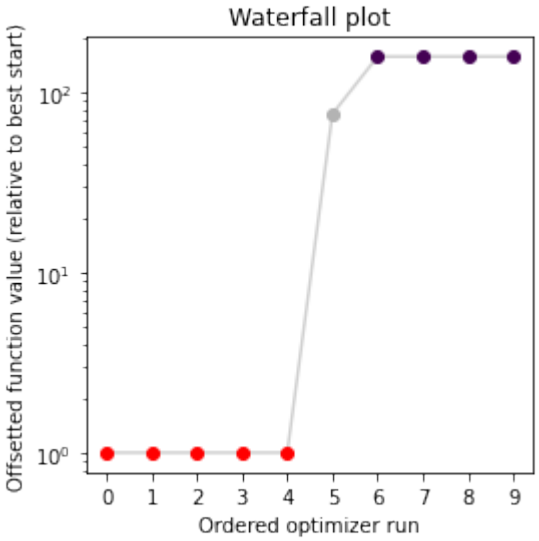
Commonly, as a first step, optimization is performed, in order to find good parameter point estimates.

[2]:
```
%%time
result = optimize.minimize(problem, n_starts=10)
```

```
Parameters obtained from history and optimizer do not match: [ -0.91617976 -10.
↪39367452], [ -0.9161923  -10.39367204]
```

```
CPU times: user 2.21 s, sys: 406 ms, total: 2.62 s
Wall time: 2.06 s
```

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



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

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

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

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

```
100%|| 10000/10000 [00:55<00:00, 179.06it/s]
```

```
CPU times: user 1min, sys: 9.35 s, total: 1min 9s
Wall time: 55.9 s
```

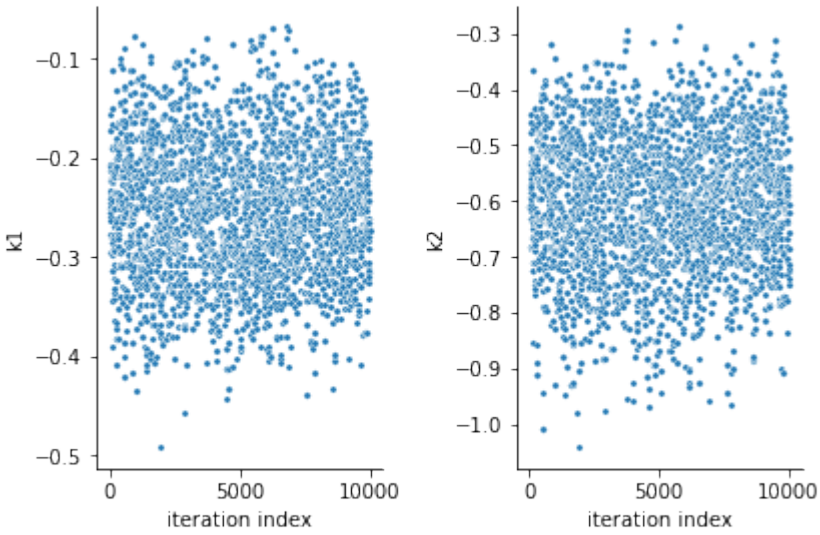
When the sampling is finished, we can analyse our results. A first thing to do is to analyze the sampling burn-in:

[6]:
```
sample.geweke_test(result)
```

[6]:
```
0
```

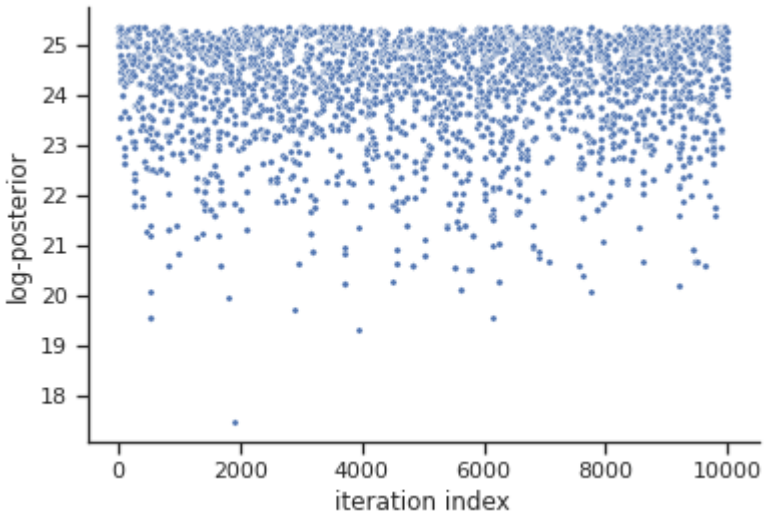
pyPESTO provides functions to analyse both the sampling process as well as the obtained sampling result. Visualizing the traces e.g. allows to detect burn-in phases, or fine-tune hyperparameters. First, the parameter trajectories can be visualized:

[7]:
```
sample.geweke_test(result)
ax = visualize.sampling_parameter_traces(result, use_problem_bounds=False)
```



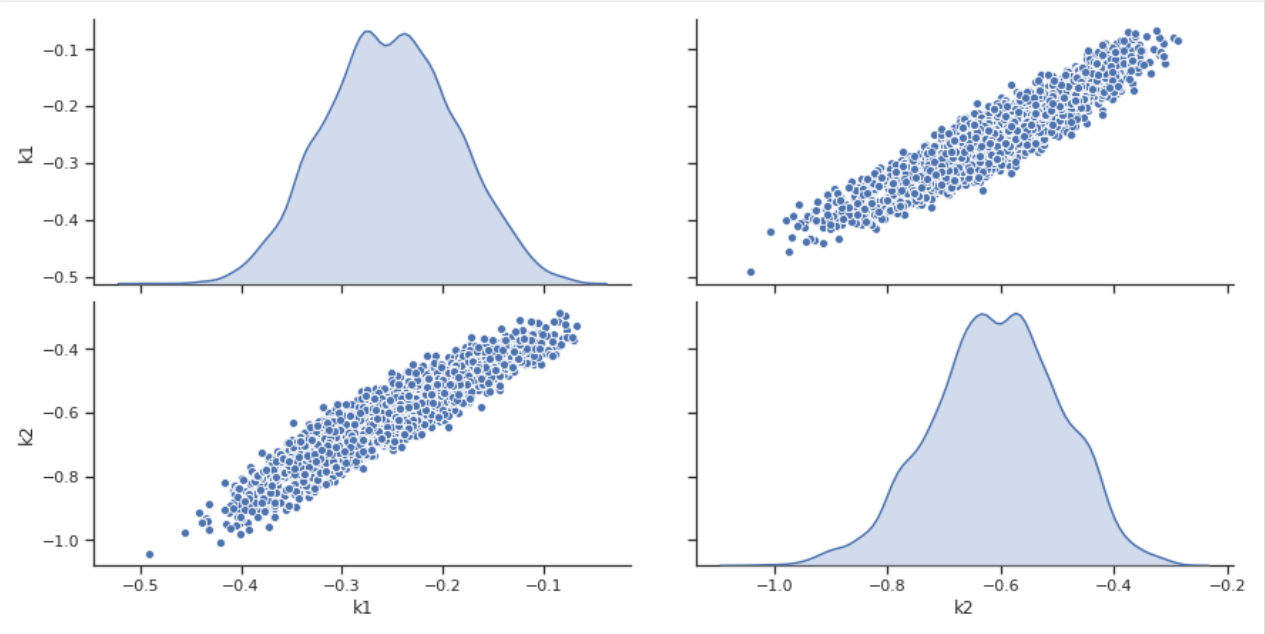
Next, also the log posterior trace can be visualized:

[8]:
```
ax = visualize.sampling_fval_traces(result)
```



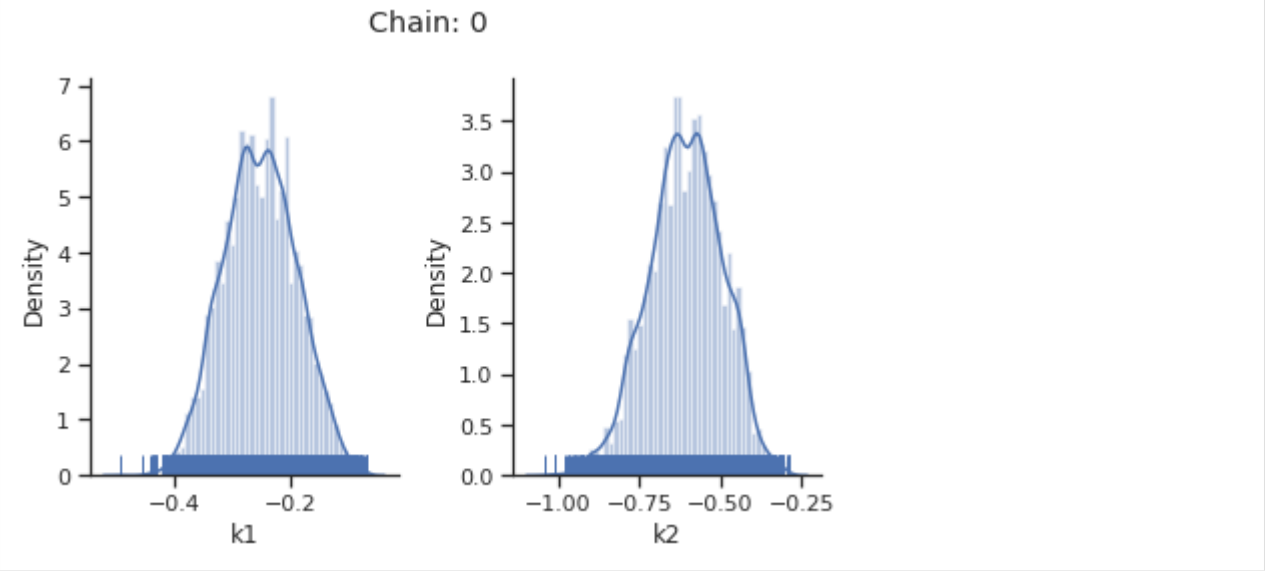
To visualize the result, there are various options. The scatter plot shows histograms of 1-dim parameter marginals and scatter plots of 2-dimensional parameter combinations:

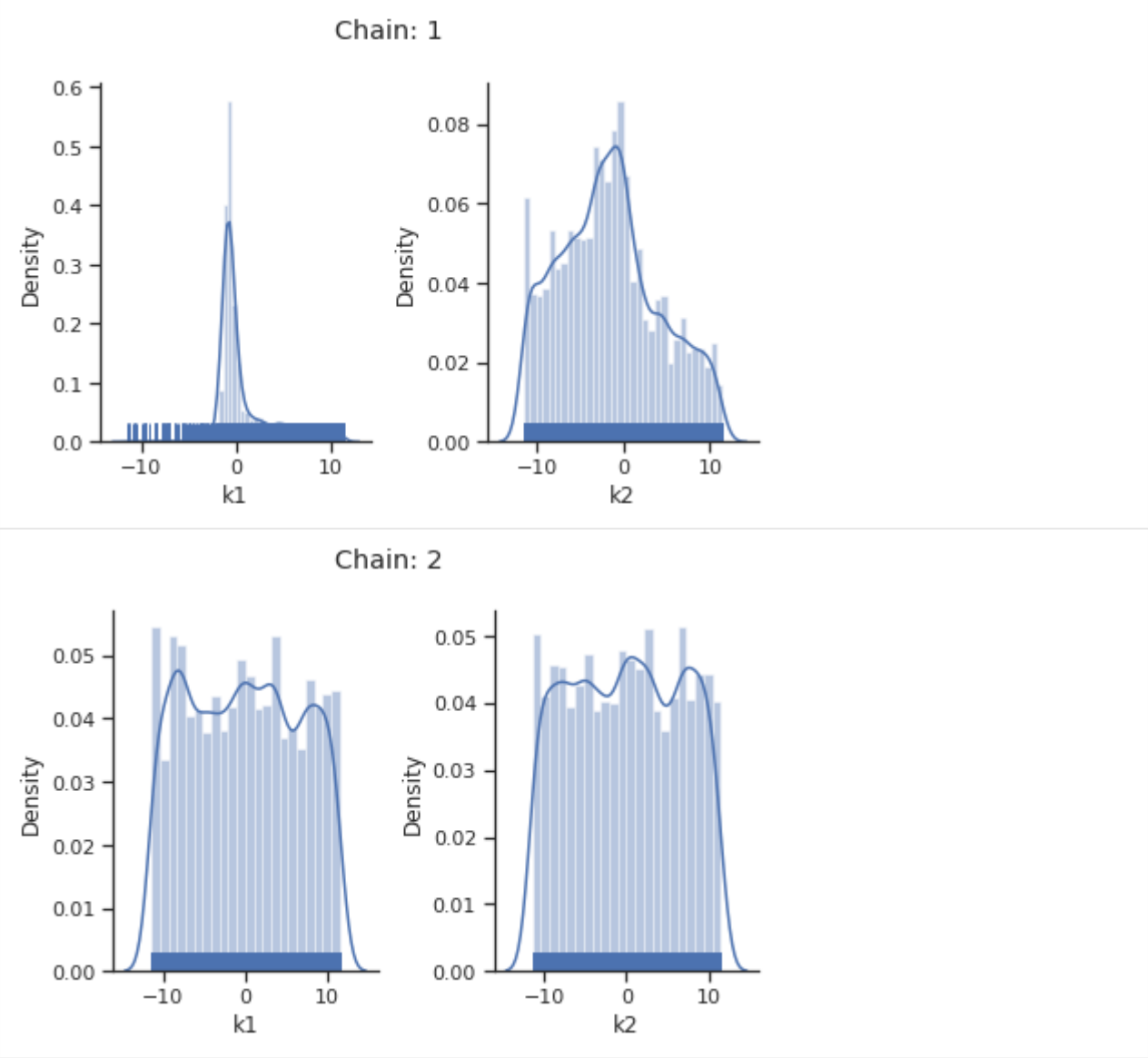
[9]:
```
ax = visualize.sampling_scatter(result, size=[13,6])
```



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

[10]:
```
for i_chain in range(len(result.sample_result.betas)):
    visualize.sampling_1d_marginals(
        result, i_chain=i_chain, suptitle=f"Chain: {i_chain}")
```





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

2.7.2 1-dim test problem

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

[1]:
```
import numpy as np
from scipy.stats import multivariate_normal
import seaborn as sns
import pypesto
import pypesto.sample as sample
import pypesto.visualize as visualize

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

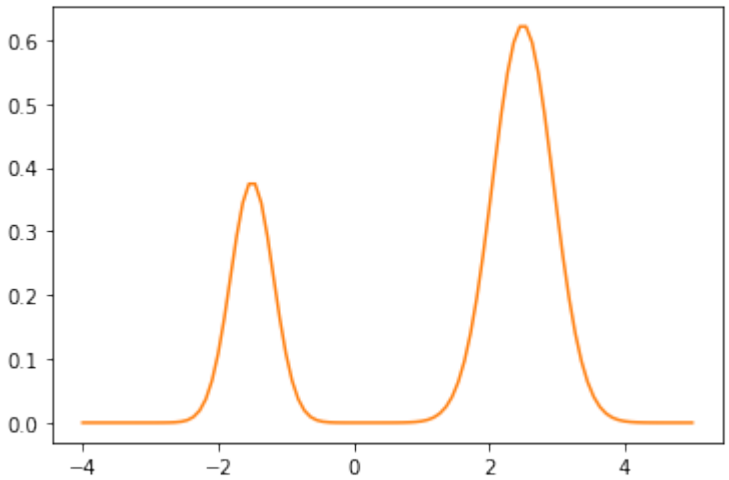
(continues on next page)

(continued from previous page)

```
        0.7*multivariate_normal.pdf(x, mean=2.5, cov=0.2)

def nllh(x):
    return - np.log(density(x))

objective = pypesto.Objective(fun=nllh)
problem = pypesto.Problem(
    objective=objective, lb=-4, ub=5, x_names=['x'])
```

The likelihood has two separate modes:

[2]:
```
xs = np.linspace(-4, 5, 100)
ys = [density(x) for x in xs]

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



Metropolis sampler

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

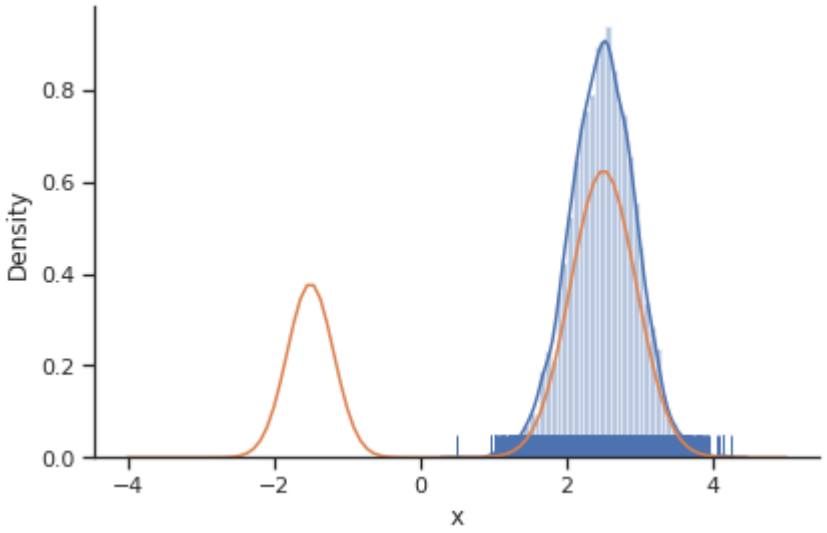
[13]:
```
%%time
sampler = sample.MetropolisSampler({'std': 0.5})
result = sample.sample(problem, 1e4, sampler, x0=np.array([0.5]))
```

```
100%|| 10000/10000 [00:03<00:00, 2774.43it/s]
```

```
CPU times: user 3.65 s, sys: 194 ms, total: 3.84 s
Wall time: 3.62 s
```

[14]:
```
sample.geweke_test(result)
ax = visualize.sampling_1d_marginals(result)
ax[0][0].plot(xs, ys)
```

[14]:
```
[<matplotlib.lines.Line2D at 0x7f8466613f40>]
```



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

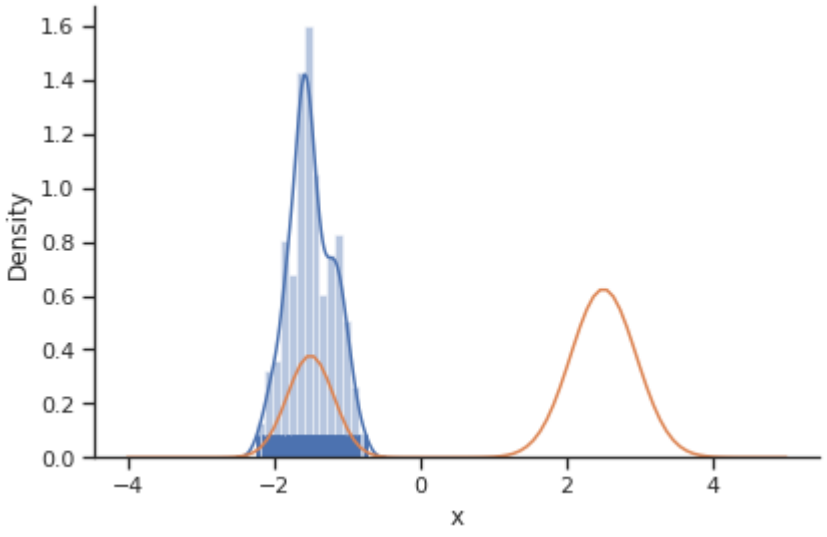
[15]:
```
%%time
sampler = sample.MetropolisSampler({'std': 1})
result = sample.sample(problem, 1e4, sampler, x0=np.array([0.5]))
```

```
100%|| 10000/10000 [00:03<00:00, 2849.08it/s]
```

```
CPU times: user 3.55 s, sys: 190 ms, total: 3.74 s
Wall time: 3.52 s
```

[16]:
```
sample.geweke_test(result)
ax = visualize.sampling_1d_marginals(result)
ax[0][0].plot(xs, ys)
```

[16]:
```
[<matplotlib.lines.Line2D at 0x7f844fffcdf0>]
```



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

Parallel tempering sampler

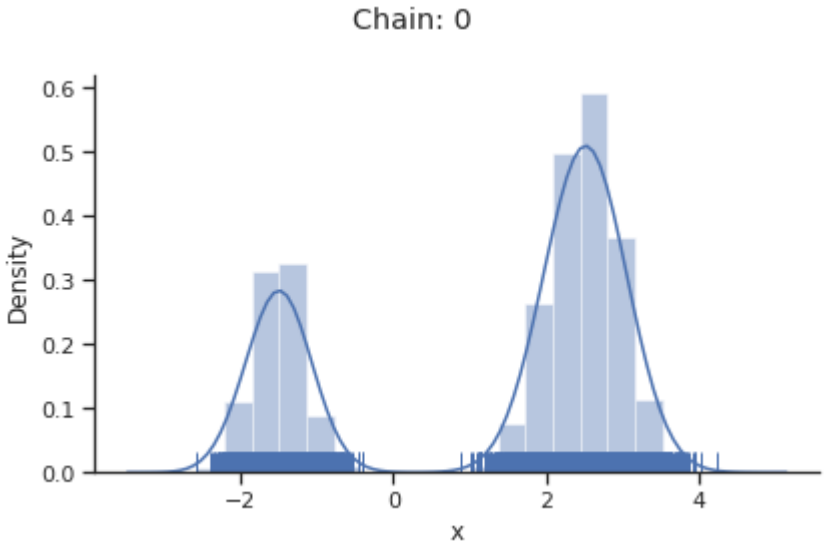
In pyPESTO, the most basic parallel tempering algorithm is the `pypesto.sample.ParallelTemperingSampler`. It takes an `internal_sampler` parameter, to specify what sampler to use for performing sampling the different chains. Further, we can directly specify what inverse temperatures `betas` to use. When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

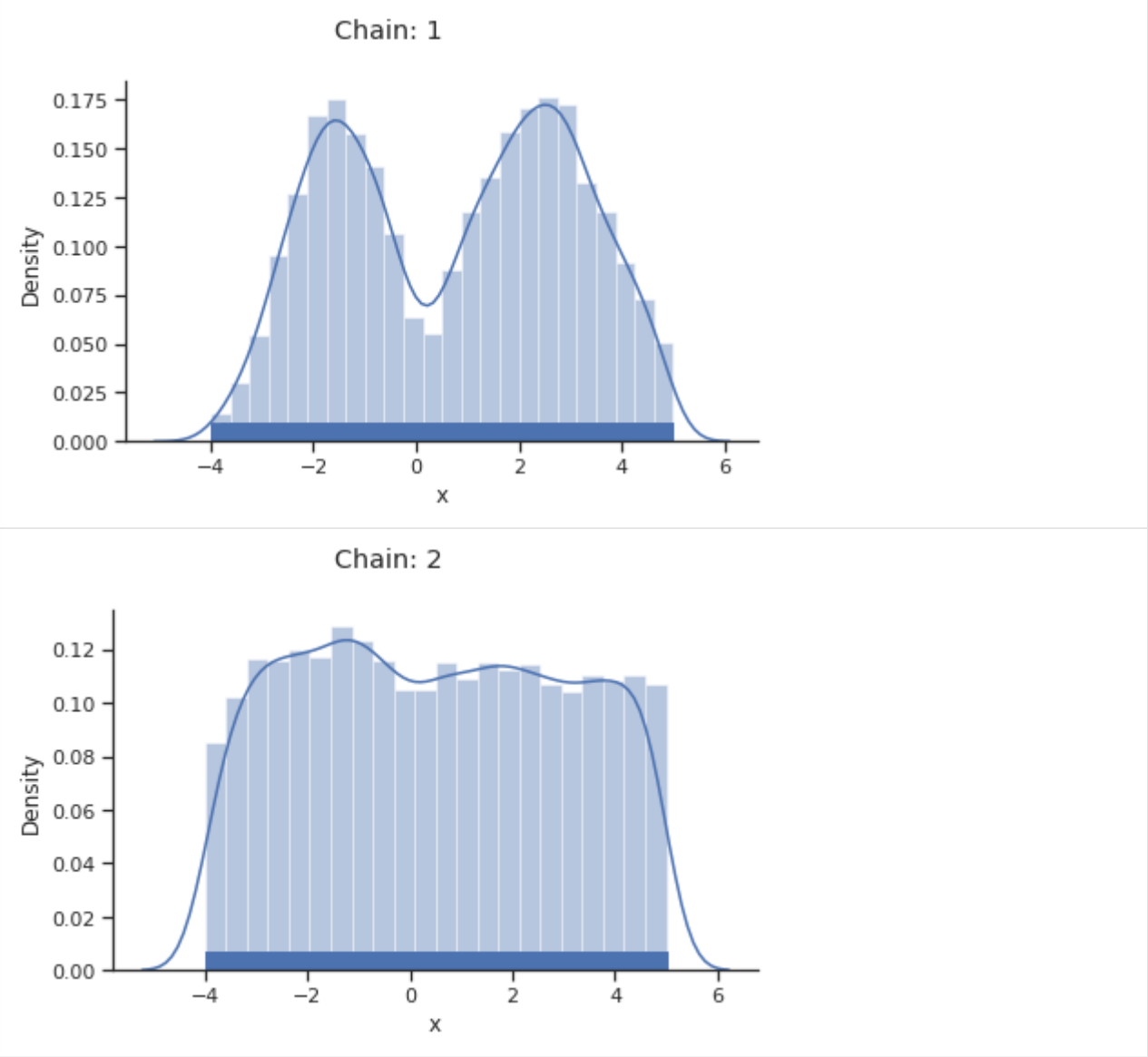
[17]:
```
%%time
sampler = sample.ParallelTemperingSampler(
    internal_sampler=sample.MetropolisSampler(),
    betas=[1, 1e-1, 1e-2])
result = sample.sample(problem, 1e4, sampler, x0=np.array([0.5]))
```

```
100%|| 10000/10000 [00:11<00:00, 863.05it/s]
```

```
CPU times: user 11.8 s, sys: 621 ms, total: 12.4 s
Wall time: 11.6 s
```

[18]:
```
sample.geweke_test(result)
for i_chain in range(len(result.sample_result.betas)):
    visualize.sampling_1d_marginals(
        result, i_chain=i_chain, suptitle=f"Chain: {i_chain}")
```





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

Adaptive Metropolis sampler

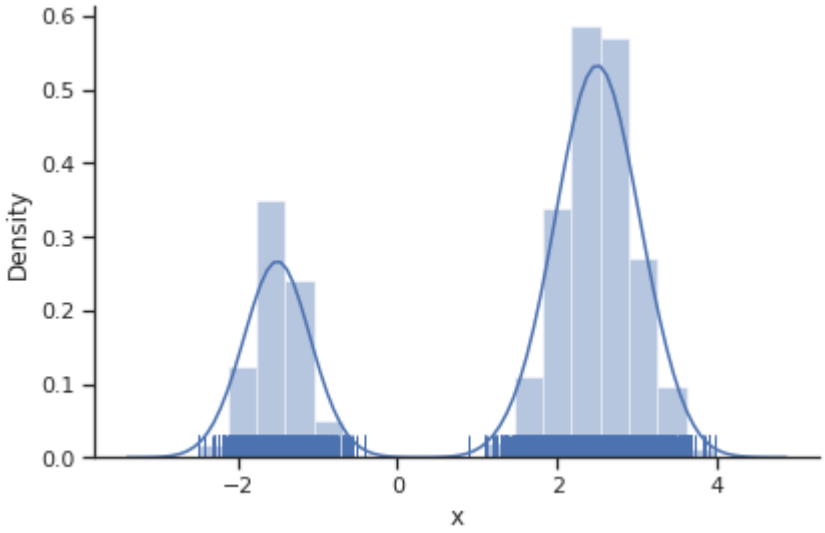
The problem of having to specify the proposal step variation manually can be overcome by using the `pypesto.sample.AdaptiveMetropolisSampler`, which iteratively adjusts the proposal steps to the function landscape.

[19]:
```
%%time
sampler = sample.AdaptiveMetropolisSampler()
result = sample.sample(problem, 1e4, sampler, x0=np.array([0.5]))
```

```
100%|| 10000/10000 [00:04<00:00, 2337.47it/s]
```

```
CPU times: user 4.35 s, sys: 55.4 ms, total: 4.41 s
Wall time: 4.3 s
```

[20]:
```
sample.geweke_test(result)
ax = visualize.sampling_1d_marginals(result)
```



Adaptive parallel tempering sampler

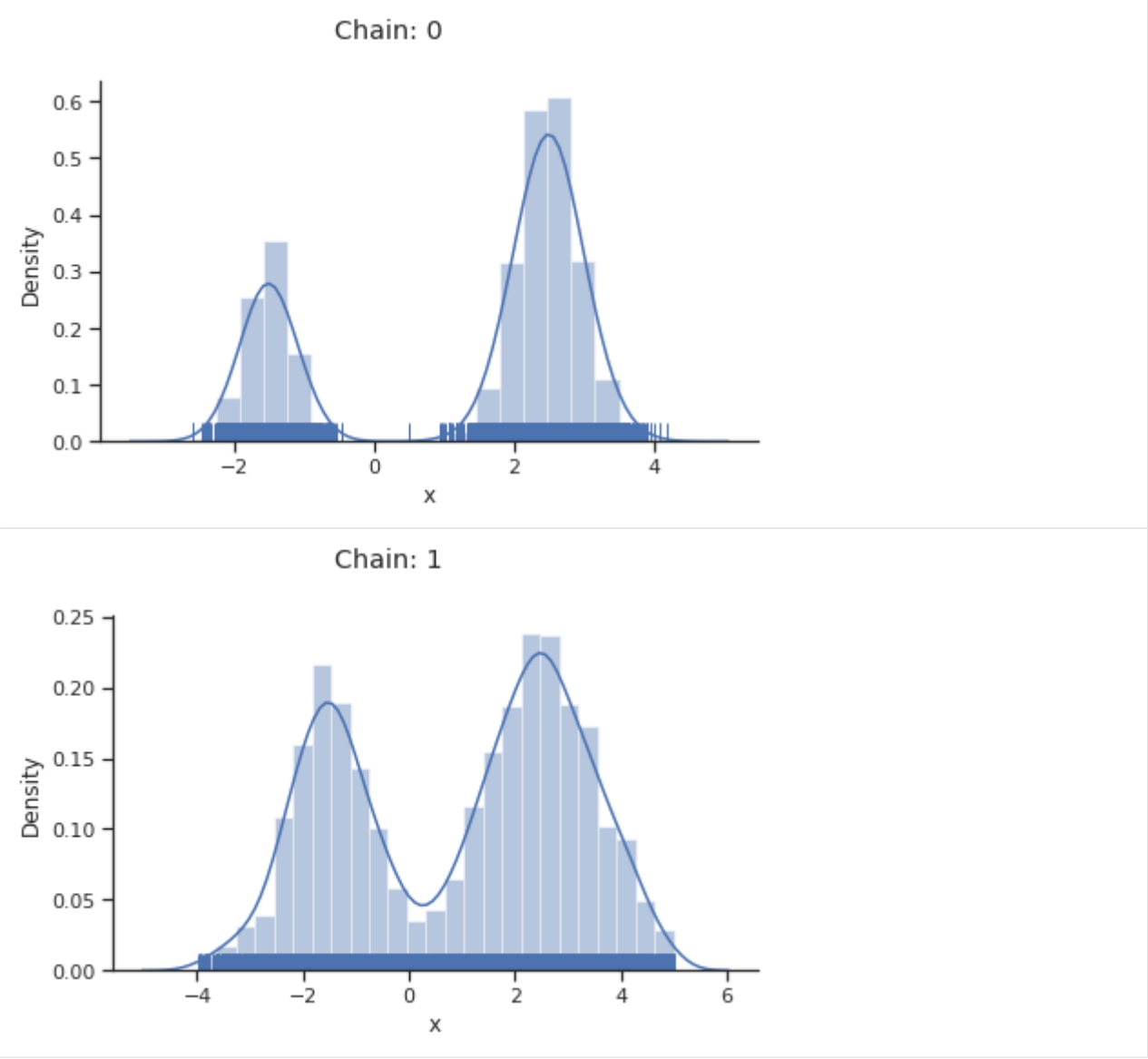
The `pypesto.sample.AdaptiveParallelTemperingSampler` iteratively adjusts the temperatures to obtain good swapping rates between chains.

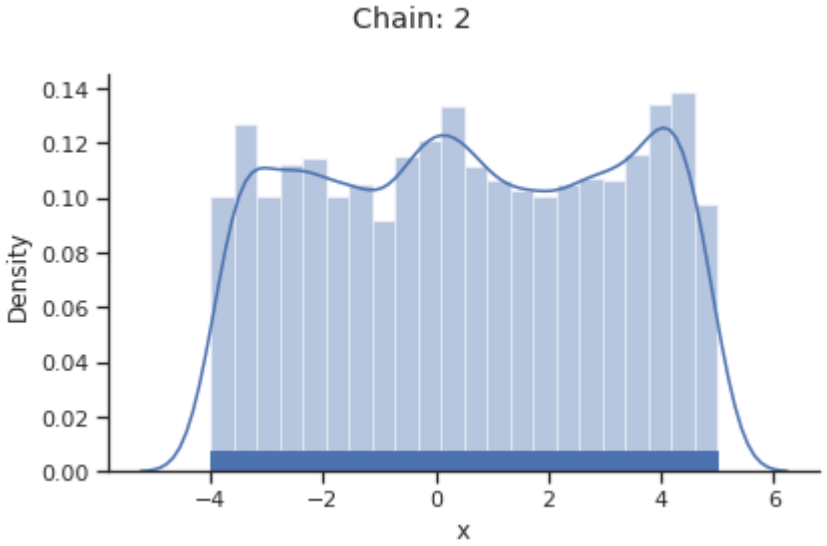
[21]:
```
%%time
sampler = sample.AdaptiveParallelTemperingSampler(
    internal_sampler=sample.AdaptiveMetropolisSampler(), n_chains=3)
result = sample.sample(problem, 1e4, sampler, x0=np.array([0.5]))
```

```
100%|| 10000/10000 [00:13<00:00, 752.49it/s]
```

```
CPU times: user 13.3 s, sys: 91 ms, total: 13.4 s
Wall time: 13.3 s
```

[22]:
```
sample.geweke_test(result)
for i_chain in range(len(result.sample_result.betas)):
    visualize.sampling_1d_marginals(
        result, i_chain=i_chain, suptitle=f"Chain: {i_chain}")
```


Chain: 0
Density
x
Chain: 1
Density
x



[23]:
```
result.sample_result.betas
```

[23]:
```
array([1.0000000e+00, 2.1503801e-01, 2.0000000e-05])
```

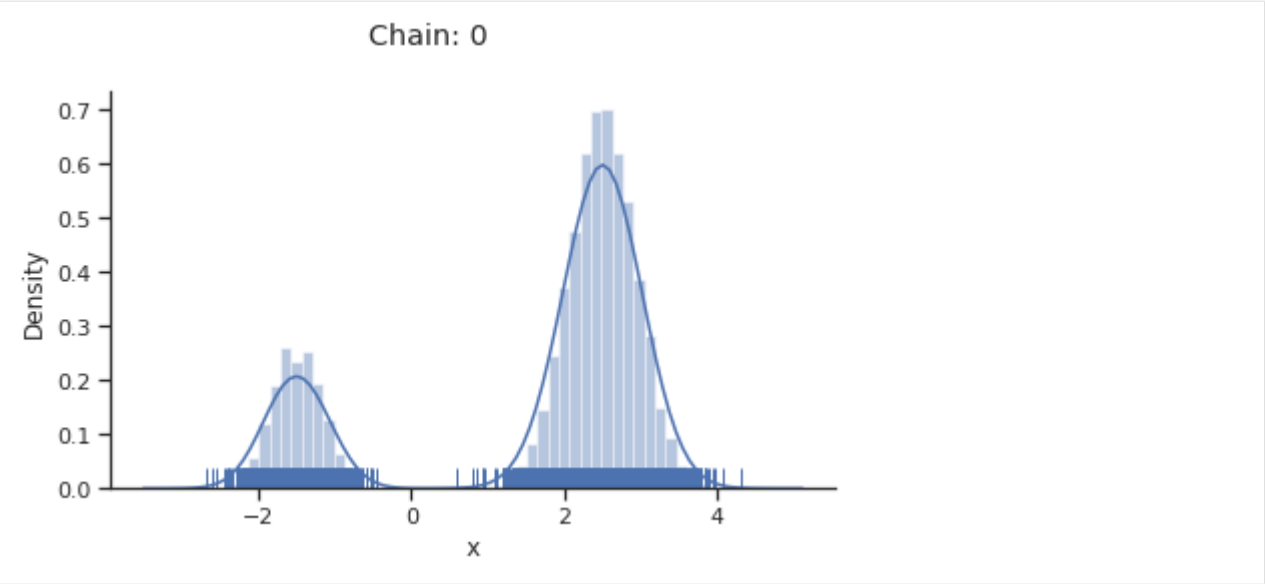
Pymc3 sampler

[24]:
```
%%time
sampler = sample.Pymc3Sampler()
result = sample.sample(problem, 1e4, sampler, x0=np.array([0.5]))
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Initializing NUTS failed. Falling back to elementwise auto-assignment.
Sequential sampling (1 chains in 1 job)
Slice: [x]
Sampling chain 0, 0 divergences: 100%|| 10500/10500 [00:25<00:00, 407.36it/s]
Only one chain was sampled, this makes it impossible to run some convergence checks
```

```
CPU times: user 29.2 s, sys: 860 ms, total: 30.1 s
Wall time: 30.8 s
```

[25]:
```
sample.geweke_test(result)
for i_chain in range(len(result.sample_result.betas)):
    visualize.sampling_1d_marginals(
        result, i_chain=i_chain, suptitle=f"Chain: {i_chain}")
```



If not specified, pymc3 chooses an adequate sampler automatically.

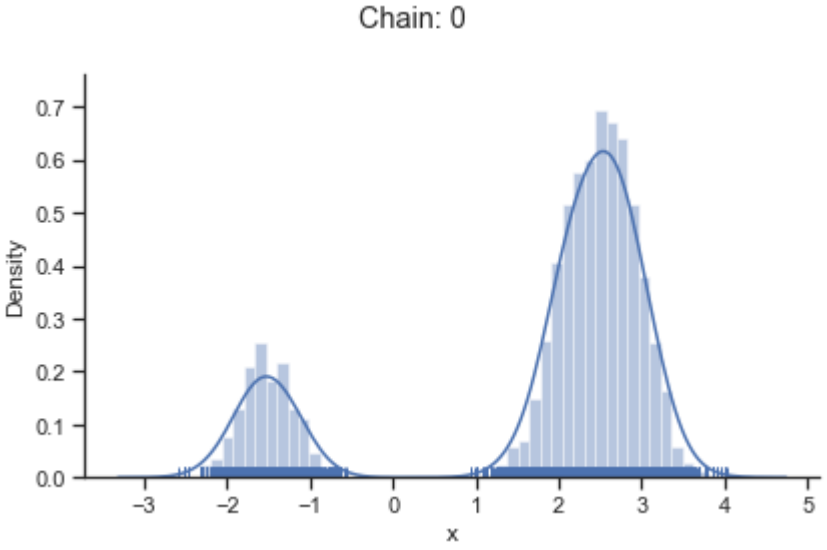
Emcee sampler

[4]:
```
%%time
sampler = sample.EmceeSampler(nwalkers=10, run_args={'progress': True})
result = sample.sample(problem, 1e4, sampler, x0=np.array([0.5]))
```

```
100%|| 10000/10000 [00:35<00:00, 278.38it/s]
```

```
CPU times: user 35.8 s, sys: 344 ms, total: 36.2 s
Wall time: 35.9 s
```

[5]:
```
sample.geweke_test(result)
for i_chain in range(len(result.sample_result.betas)):
    visualize.sampling_1d_marginals(
        result, i_chain=i_chain, suptitle=f"Chain: {i_chain}")
```



2.7.3 2-dim test problem: Rosenbrock banana

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

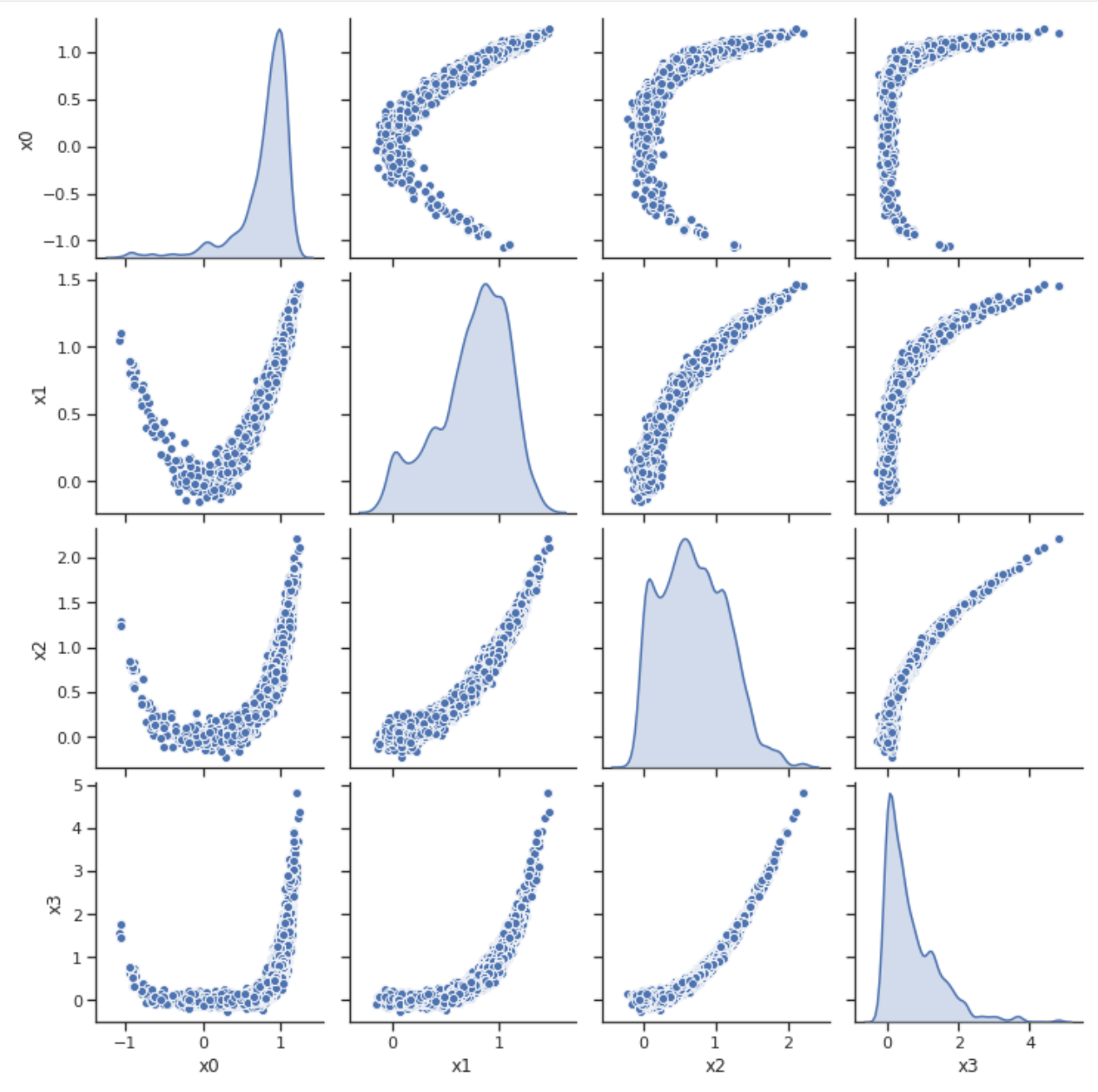
[26]:
```
import scipy.optimize as so
import pypesto

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

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

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

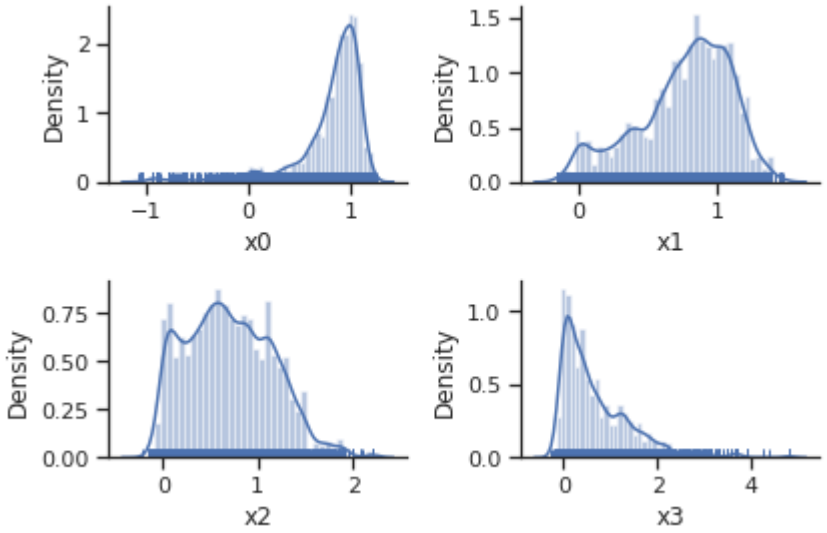
[27]:
```
%%time
sampler = sample.AdaptiveParallelTemperingSampler(
    internal_sampler=sample.AdaptiveMetropolisSampler(), n_chains=10)
result =sample.sample(problem, 1e4, sampler, x0=np.zeros(dim_full))
```

```
100%|| 10000/10000 [00:35<00:00, 279.44it/s]
```

```
CPU times: user 35.6 s, sys: 47.5 ms, total: 35.7 s
Wall time: 35.9 s
```

[28]:
```
sample.geweke_test(result)
ax = visualize.sampling_scatter(result)
ax = visualize.sampling_1d_marginals(result)
```


x0
x1
x2
x3
x0
x1
x2
x3



2.8 MCMC sampling diagnostics

In this notebook, we illustrate how to assess the quality of your MCMC samples, e.g. convergence and auto-correlation, in pyPESTO.

2.8.1 The pipeline

First, we load the model and data to generate the MCMC samples from. In this example we show a toy example of a conversion reaction, loaded as a PEtab problem.

[1]:
```
import pypesto
import pypesto.petab
import pypesto.optimize as optimize
import pypesto.sample as sample
import pypesto.visualize as visualize

import petab
import numpy as np
import logging
import matplotlib.pyplot as plt

# log diagnostics
logger = logging.getLogger("pypesto.sample.diagnostics")
logger.setLevel(logging.INFO)
logger.addHandler(logging.StreamHandler())

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

Create the sampler object, in this case we will use adaptive parallel tempering with 3 temperatures.

[2]:
```
sampler = sample.AdaptiveParallelTemperingSampler(
    internal_sampler=sample.AdaptiveMetropolisSampler(),
    n_chains=3)
```

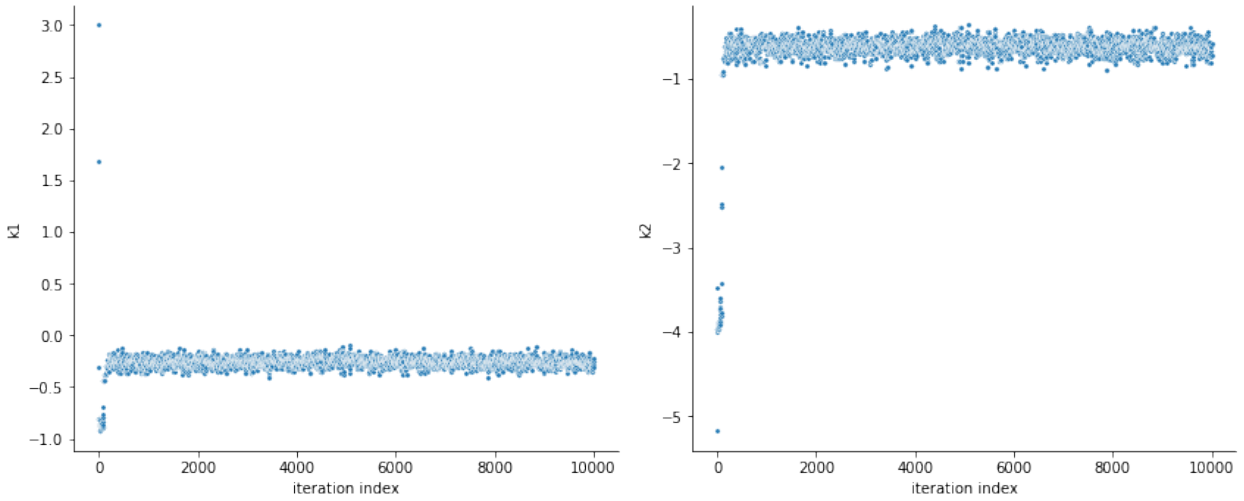
First, we will initiate the MCMC chain at a "random" point in parameter space, e.g. $\theta_{start} = [3, -4]$

[3]:
```
result = sample.sample(problem, n_samples=10000, sampler=sampler, x0=np.array([3,-4]))
elapsed_time = result.sample_result.time
print(f'Elapsed time: {round(elapsed_time,2)}')
```

```
100%|| 10000/10000 [00:44<00:00, 225.23it/s]
```

```
Elapsed time: 54.85
```

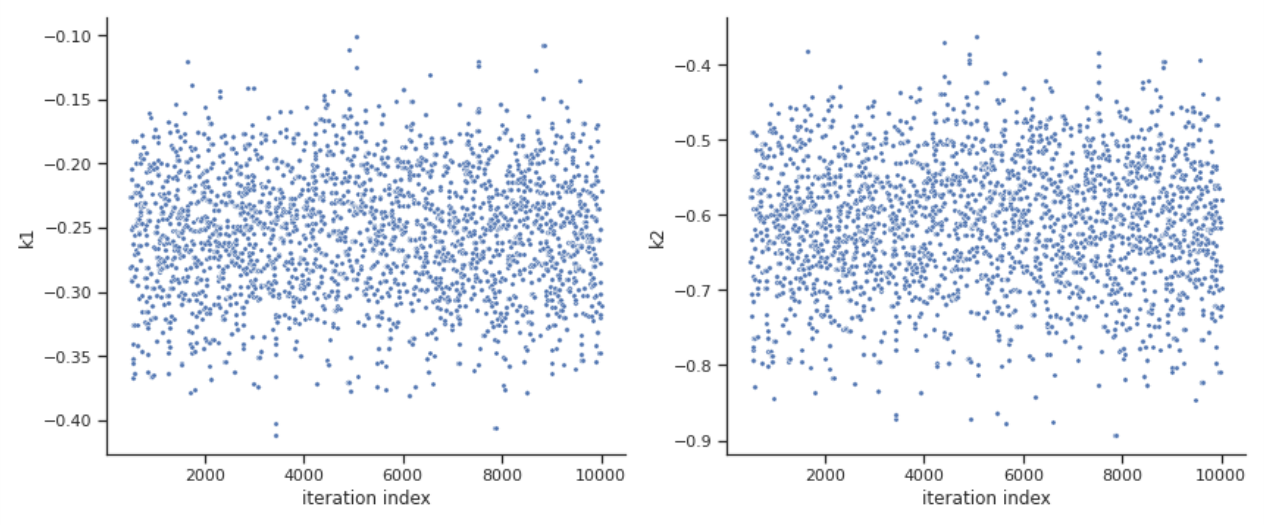
[4]:
```
ax = visualize.sampling_parameter_traces(result, use_problem_bounds=False, size=(12,
↪5))
```



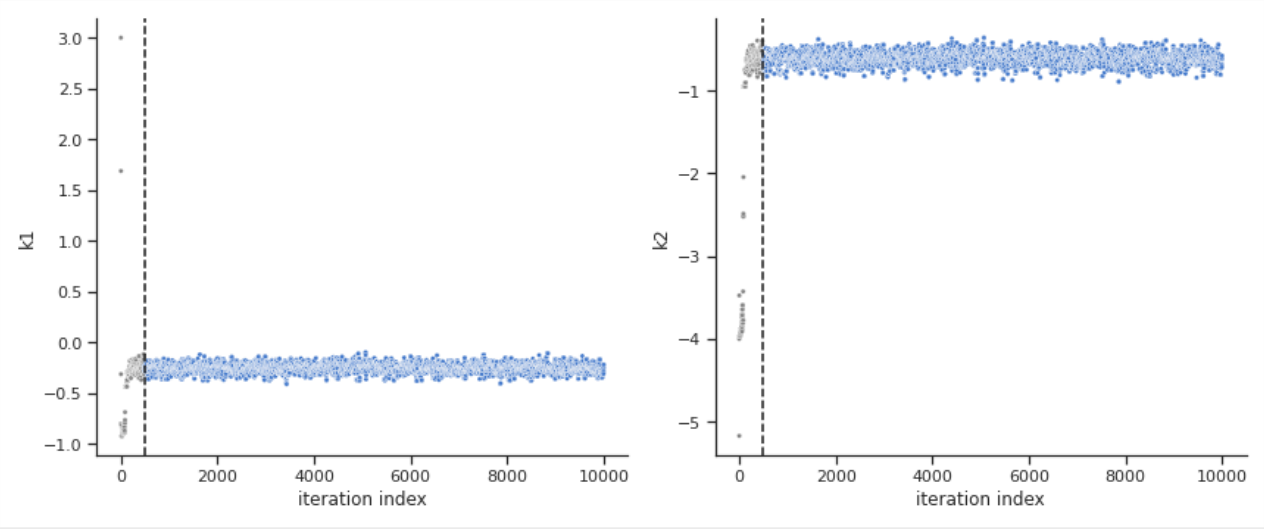
By visualizing the chains, we can see a warm up phase occurring until convergence of the chain is reached. This is commonly known as "burn in" phase and should be discarded. An automatic way to evaluate and find the index of the chain in which the warm up is finished can be done by using the Geweke test.

[5]:
```
sample.geweke_test(result=result)
ax = visualize.sampling_parameter_traces(result, use_problem_bounds=False, size=(12,
↪5))
```

```
Geweke burn-in index: 500
```



[6]:
```
ax = visualize.sampling_parameter_traces(result, use_problem_bounds=False, full_
↪trace=True, size=(12,5))
```



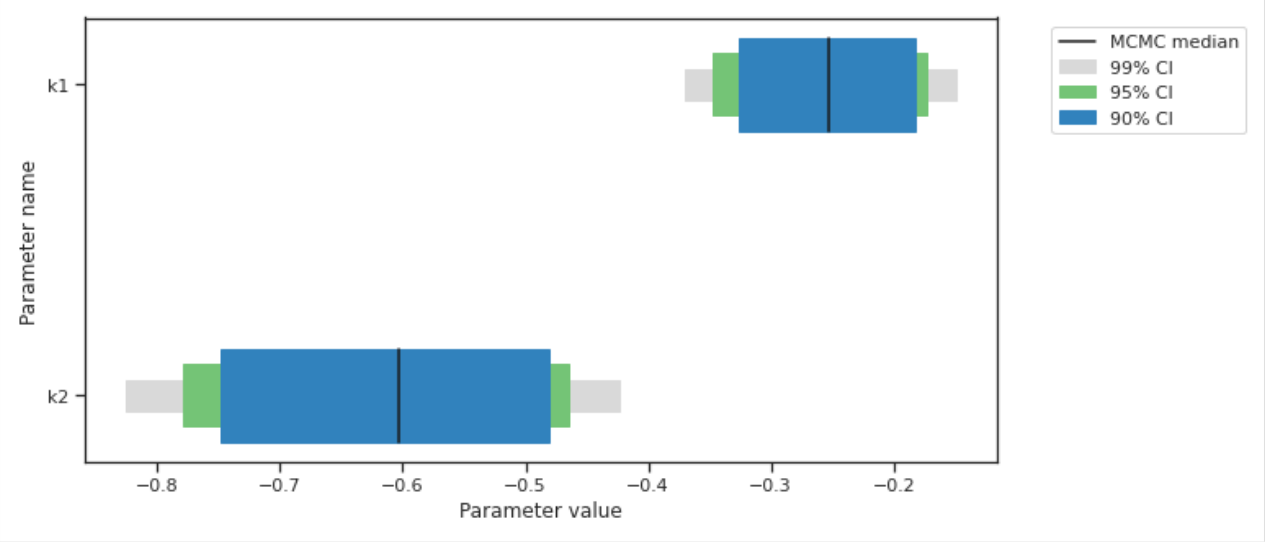
Calculate the effective sample size per computation time. We save the results in a variable as we will compare them later.

[7]:
```
sample.effective_sample_size(result=result)
ess = result.sample_result.effective_sample_size
print(f'Effective sample size per computation time: {round(ess/elapsed_time,2)}')
```

```
Estimated chain autocorrelation: 8.482536903793251
Estimated effective sample size: 1001.947062942551
```

```
Effective sample size per computation time: 18.27
```

[8]:
```
alpha = [99, 95, 90]
ax = visualize.sampling_parameter_cis(result, alpha=alpha, size=(10,5))
```



Predictions can be performed by creating a parameter ensemble from the sample, then applying a predictor to the ensemble. The predictor requires a simulation tool. Here, AMICI is used. First, the predictor is setup.

[9]:
```
from pypesto.predict.constants import AMICI_STATUS, AMICI_T, AMICI_X, AMICI_Y
from pypesto.predict import AmiciPredictor


# This post_processor will transform the output of the simulation tool
# such that the output is compatible with the next steps.
def post_processor(amici_outputs, output_type, output_ids):
    outputs = [
        amici_output[output_type] if amici_output[AMICI_STATUS] == 0
        else np.full((len(amici_output[AMICI_T]), len(output_ids)), np.nan)
        for amici_output in amici_outputs
    ]
    return outputs


# Setup post-processors for both states and observables.
from functools import partial
amici_objective = result.problem.objective
state_ids = amici_objective.amici_model.getStateIds()
observable_ids = amici_objective.amici_model.getObservableIds()
post_processor_x = partial(
    post_processor,
    output_type=AMICI_X,
    output_ids=state_ids,
)
post_processor_y = partial(
    post_processor,
    output_type=AMICI_Y,
    output_ids=observable_ids,
)

# Create pyPESTO predictors for states and observables
predictor_x = AmiciPredictor(
    amici_objective,
    post_processor=post_processor_x,
```

(continues on next page)

(continued from previous page)

```
    output_ids=state_ids,
)
predictor_y = AmiciPredictor(
    amici_objective,
    post_processor=post_processor_y,
    output_ids=observable_ids,
)
```

Next, the ensemble is created.

[10]:
```
from pypesto.ensemble import Ensemble, EnsembleType

# corresponds to only the estimated parameters
x_names = result.problem.get_reduced_vector(result.problem.x_names)

# Create the ensemble with the MCMC chain from parallel tempering with the real␣
↪temperature.
ensemble = Ensemble.from_sample(
    result,
    chain_slice=slice(None, None, 2),  # Optional argument: only use every second␣
↪vector in the chain.
    x_names=x_names,
    ensemble_type=EnsembleType.sample,
    lower_bound=result.problem.lb,
    upper_bound=result.problem.ub
)
```

The predictor is then applied to the ensemble to generate predictions.

[11]:
```
from pypesto.engine import MultiProcessEngine

# Currently, parallelization of predictions is supported with the
# `pypesto.engine.MultiProcessEngine` and `pypesto.engine.MultiThreadEngine`
# engines. If no engine is specified, the `pypesto.engine.SingleCoreEngine`
# engine is used.
engine = MultiProcessEngine()

ensemble_prediction = ensemble.predict(predictor_x, prediction_id=AMICI_X,␣
↪engine=engine)
```

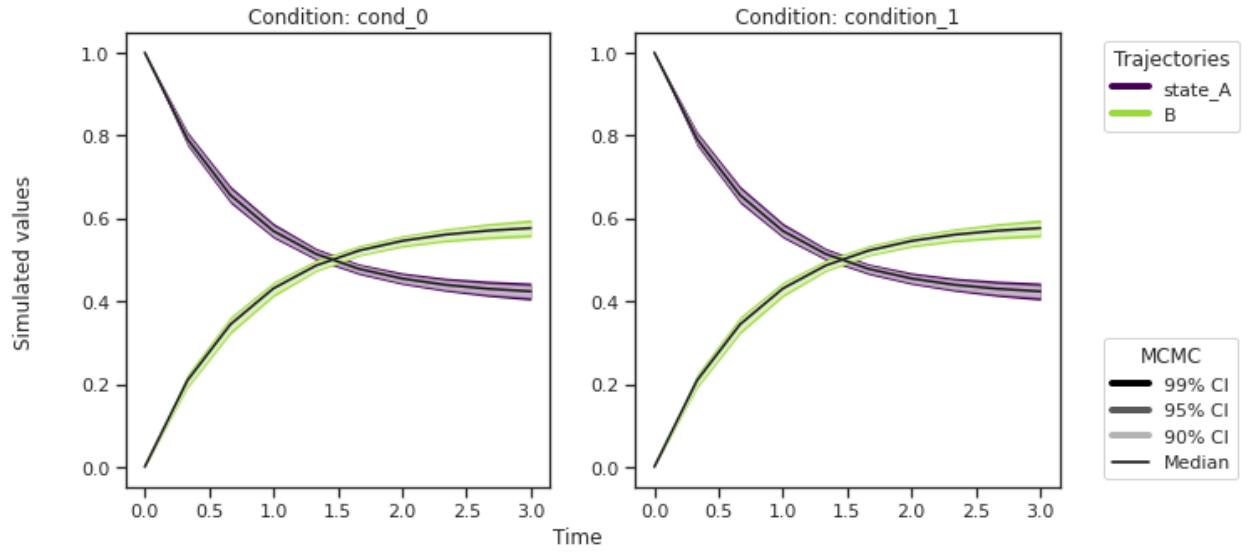
```
Engine set up to use up to 8 processes in total. The number was automatically␣
↪determined and might not be appropriate on some systems.
100%|| 8/8 [00:00<00:00, 641.55it/s]
```

[12]:
```
from pypesto.predict.constants import CONDITION, OUTPUT

credibility_interval_levels = [90, 95, 99]

ax = visualize.sampling_prediction_trajectories(
    ensemble_prediction,
    levels=credibility_interval_levels,
    size=(10,5),
    labels={'A': 'state_A', 'condition_0': 'cond_0'},
    axis_label_padding=60,
    groupby=CONDITION,
    condition_ids=['condition_0', 'condition_1'],  # `None` for all conditions
    output_ids=['A', 'B'],  # `None` for all outputs
```

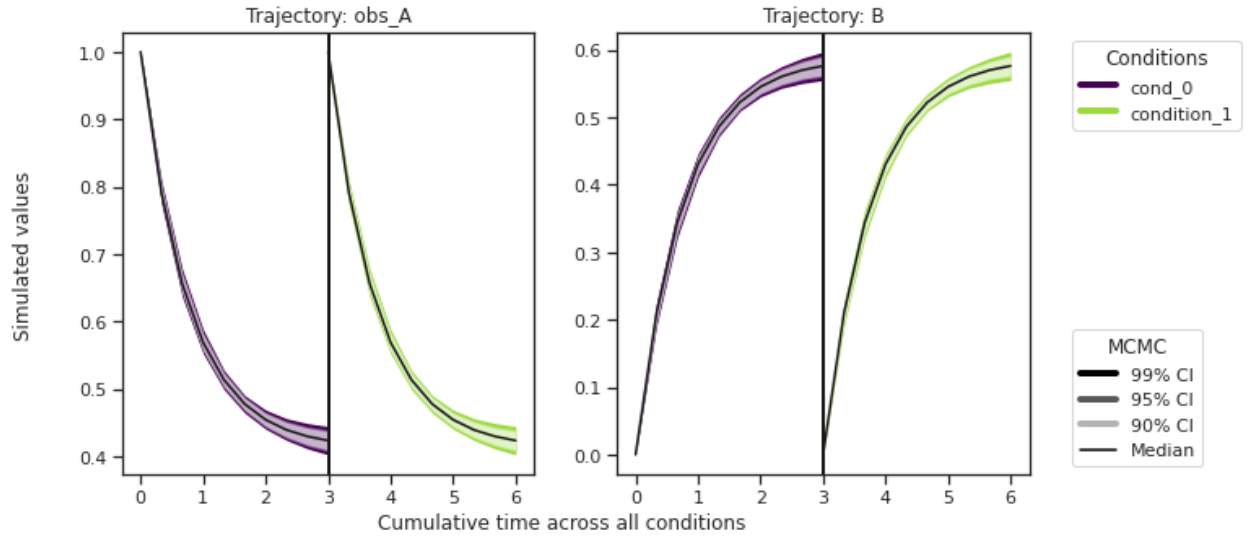
(continues on next page)

(continued from previous page)

```
)
```



[13]:
```
ax = visualize.sampling_prediction_trajectories(
    ensemble_prediction,
    levels=credibility_interval_levels,
    size=(10,5),
    labels={'A': 'obs_A', 'condition_0': 'cond_0'},
    axis_label_padding=60,
    groupby=OUTPUT,
)
```



Predictions are stored in `ensemble_prediction.prediction_summary`.

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

[14]:
```
res = optimize.minimize(problem, n_starts=10)
```

```
100%|| 10/10 [00:02<00:00,  4.72it/s]
```

By passing the result object to the function, the previously found global optimum is used as starting point for the MCMC sampling.

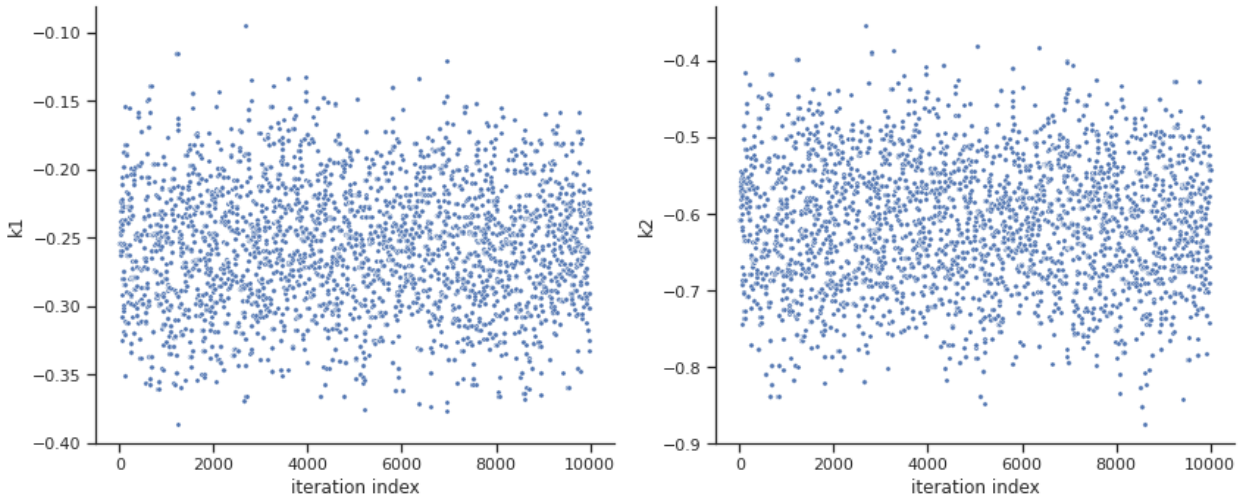
[15]:
```
res = sample.sample(problem, n_samples=10000, sampler=sampler, result=res)
elapsed_time = res.sample_result.time
print('Elapsed time: '+str(round(elapsed_time,2)))
```

```
100%|| 10000/10000 [00:42<00:00, 237.21it/s]
```

```
Elapsed time: 52.21
```

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

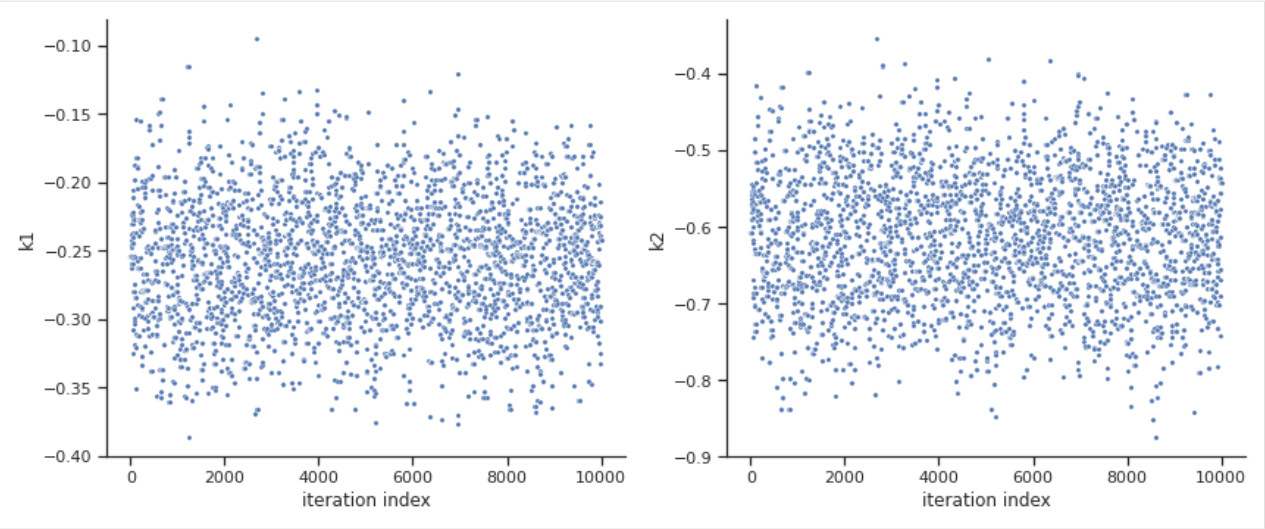
[16]:
```
ax = visualize.sampling_parameter_traces(res, use_problem_bounds=False, size=(12,5))
```



By visual inspection one can see that the chain is already converged from the start. This is already showing the benefit of initiating the chain at the optimal parameter vector. However, this may not be always the case.

[17]:
```
sample.geweke_test(result=res)
ax = visualize.sampling_parameter_traces(res, use_problem_bounds=False, size=(12,5))
```

```
Geweke burn-in index: 0
```

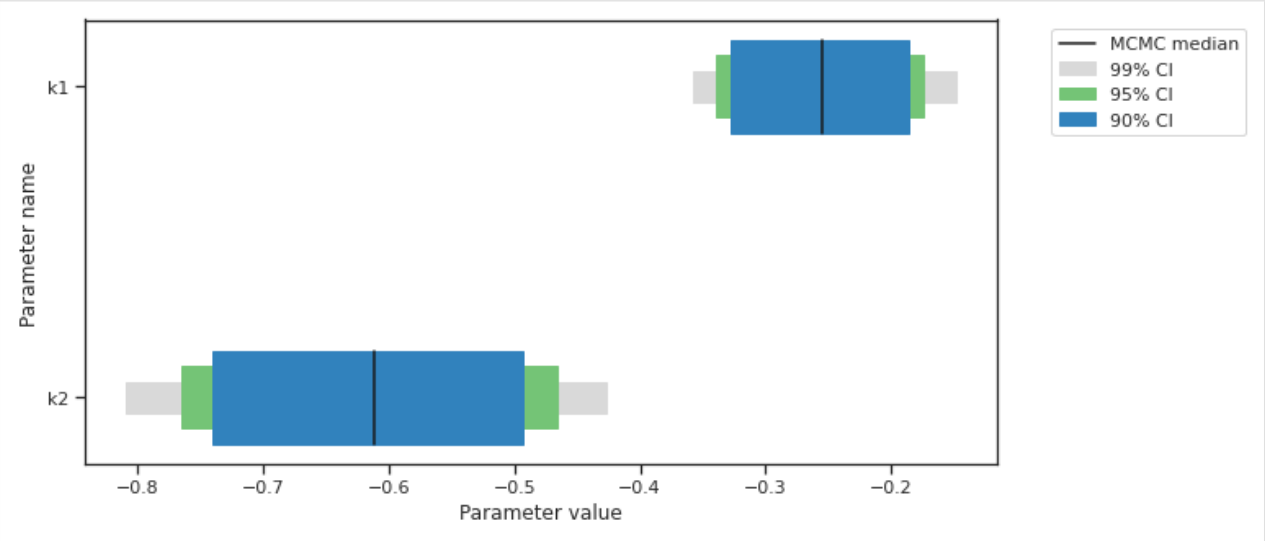


[18]:
```
sample.effective_sample_size(result=res)
ess = res.sample_result.effective_sample_size
print(f'Effective sample size per computation time: {round(ess/elapsed_time,2)}')
```

```
Estimated chain autocorrelation: 7.213987603646274
Estimated effective sample size: 1217.5572307365612
```

```
Effective sample size per computation time: 23.32
```

[19]:
```
percentiles = [99, 95, 90]
ax = visualize.sampling_parameter_cis(res, alpha=percentiles, size=(10,5))
```



[20]:
```
# Create the ensemble with the MCMC chain from parallel tempering with the real␣
↪temperature.
ensemble = Ensemble.from_sample(
    res,
    x_names=x_names,
    ensemble_type=EnsembleType.sample,
    lower_bound=res.problem.lb,
    upper_bound=res.problem.ub
```

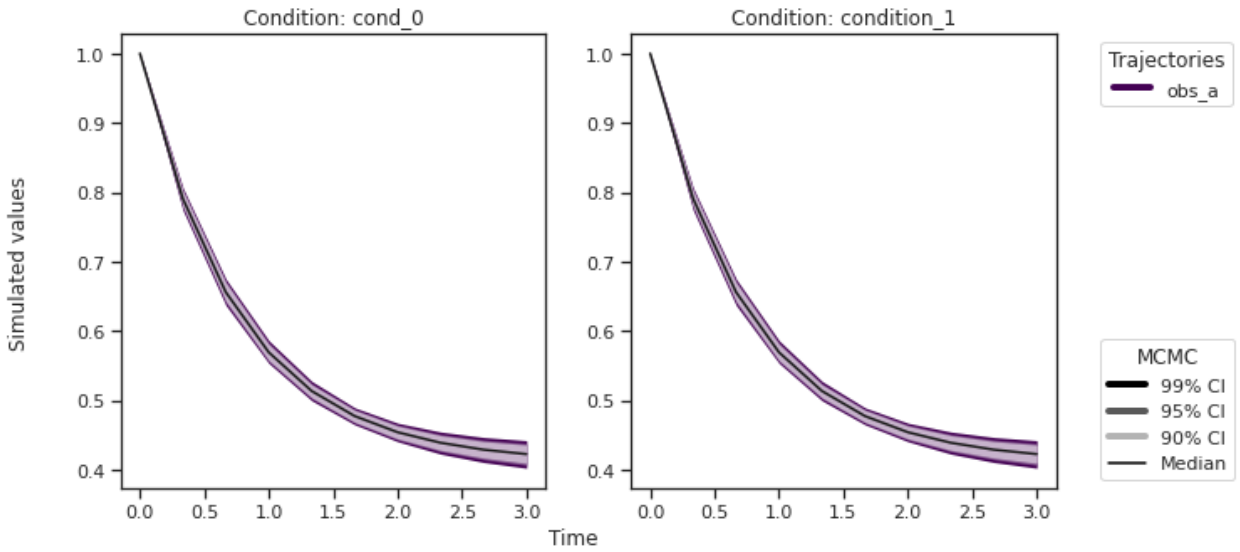
(continues on next page)

(continued from previous page)

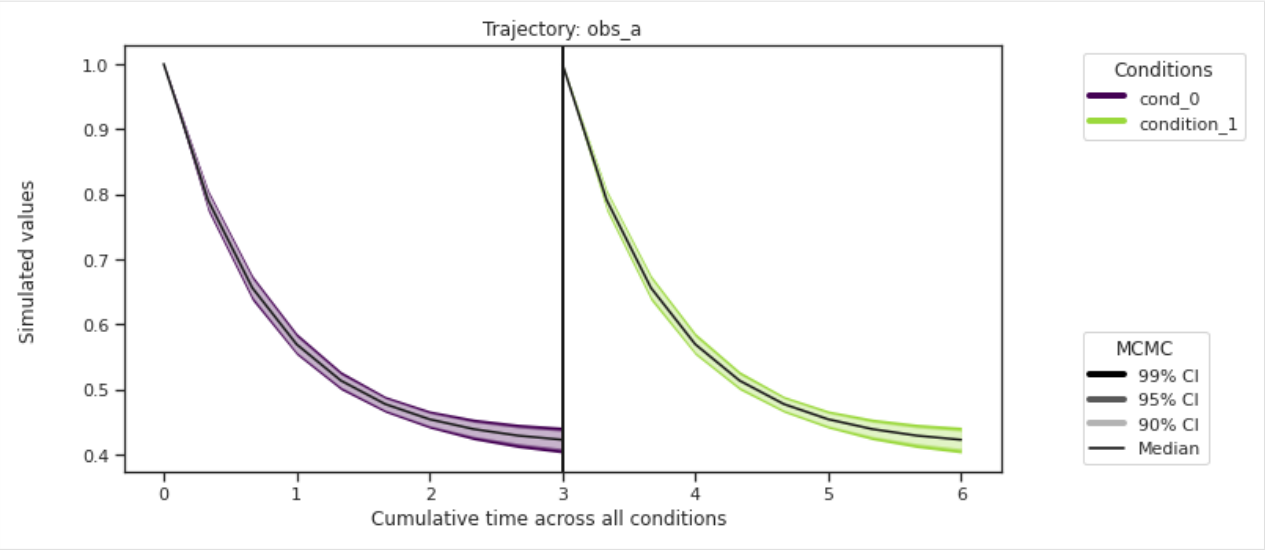
```
)

ensemble_prediction = ensemble.predict(predictor_y, prediction_id=AMICI_Y,␣
↪engine=engine)
```

```
100%|| 8/8 [00:00<00:00, 2219.21it/s]
```

[21]:
```
ax = visualize.sampling_prediction_trajectories(
    ensemble_prediction,
    levels=credibility_interval_levels,
    size=(10,5),
    labels={'A': 'obs_A', 'condition_0': 'cond_0'},
    axis_label_padding=60,
    groupby=CONDITION,
)
```



[22]:
```
ax = visualize.sampling_prediction_trajectories(
    ensemble_prediction,
    levels=credibility_interval_levels,
    size=(10,5),
    labels={'A': 'obs_A', 'condition_0': 'cond_0'},
    axis_label_padding=60,
    groupby=OUTPUT,
)
```

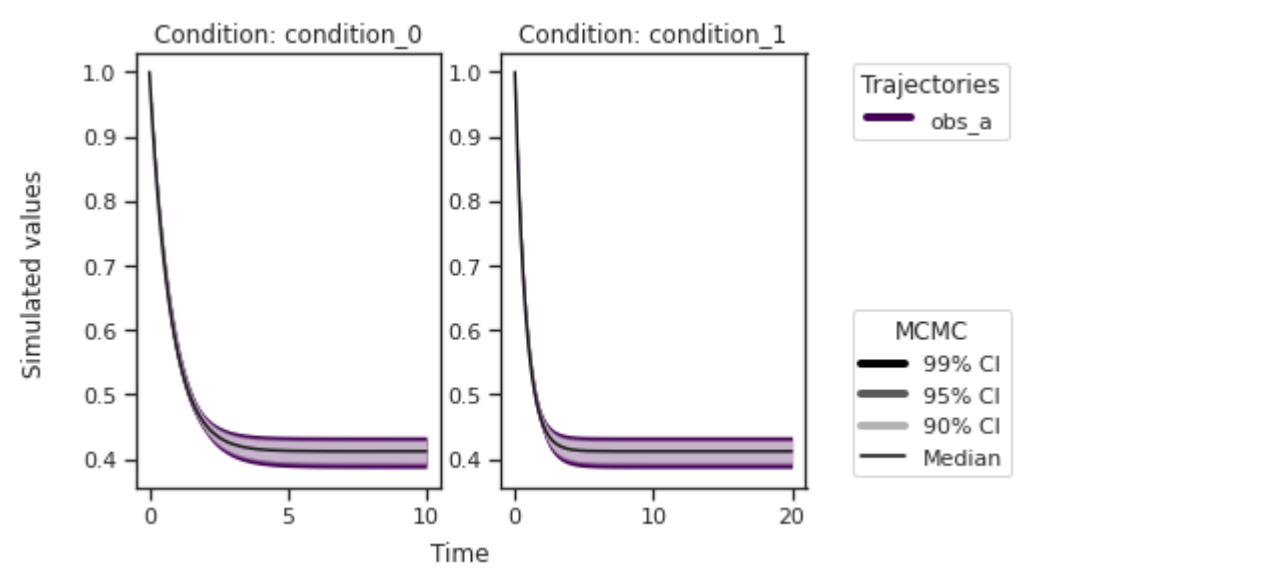


Custom timepoints can also be specified, either for each condition - `amici_objective.set_custom_timepoints(..., timepoints=...)`

or for all conditions - `amici_objective.set_custom_timepoints(..., timepoints_global=...)`.

[23]:
```
# Create a custom objective with new output timepoints.
timepoints = [np.linspace(0, 10, 100), np.linspace(0, 20, 200)]
amici_objective_custom = amici_objective.set_custom_timepoints(timepoints=timepoints)

# Create an observable predictor with the custom objective.
predictor_y_custom = AmiciPredictor(
    amici_objective_custom,
    post_processor=post_processor_y,
    output_ids=observable_ids,
)

# Predict then plot.
ensemble_prediction = ensemble.predict(predictor_y_custom, prediction_id=AMICI_Y,␣
↪engine=engine)
ax = visualize.sampling_prediction_trajectories(
    ensemble_prediction,
    levels=credibility_interval_levels,
    groupby=CONDITION,
)
```

```
100%|| 8/8 [00:00<00:00, 1193.09it/s]
```



2.9 Optimization with Synthetic Data

In this notebook, optimization is performed with an SBML model and PEtab parameter estimation problem, which includes some measurements.

Next, optimization is performed with synthetic data as measurements, which is generated using PEtab and AMICI. The ability to recover the parameter vector that was used to generate the synthetic data is demonstrated.

2.9.1 Requirements

Additional requirements for this notebook can be installed with `pip install amici petab`.

1. Load required packages. PEtab provides a base class that is designed to be easily extended to support simulation with different tools. Here, the AMICI implementation of this base class is used.

[1]:
```
import amici.petab_simulate
import matplotlib.pyplot as plt
import petab
import pypesto.optimize
import pypesto.petab
import pypesto.visualize

# Helper function to get the maximum likelihood estimate as a dictionary from a␣
↪pyPESTO optimization result.
def get_x_mle(optimize_result, pypesto_problem, petab_problem, scaled=True):
    if not scaled:
        scaling = petab.parameters.get_optimization_parameter_scaling(petab_problem.
↪parameter_df)
    return {
        x_id: (petab.parameters.unscale(x_value, scaling[x_id]) if not scaled else x_
↪value)
        for x_id, x_value in zip(pypesto_problem.x_names, optimize_result.list[0]['x
↪'])
```

(continues on next page)

(continued from previous page)

```
        #if x_id in scaling
    }
```

2.10 Standard Optimization

The PEtab problem is used to generate a pyPESTO problem, which is used to estimate model parameters.

2. Load a PEtab problem. The synthetic data returned by the PEtab-derived synthetic data generator (later, an instance of `amici.petab_simulate.PetabSimulator`) will be equivalent to switching the measurements in the PEtab problem's measurements table with simulated values.

[2]:
```
petab_yaml_filename = 'conversion_reaction/conversion_reaction.yaml'
petab_problem_original = petab.Problem.from_yaml(petab_yaml_filename)
```

3. Create a pyPESTO problem from the PEtab problem. Here, the original PEtab problem is used for parameter estimation (no synthetic data is generated).

[3]:
```
pypesto_importer_original = pypesto.petab.PetabImporter(petab_problem_original)
pypesto_problem_original = pypesto_importer_original.create_problem()
```

4. Estimate parameters. Multi-start local optimization with 100 starts is used, with the default pyPESTO optimizer.

[4]:
```
pypesto_result_original = pypesto.optimize.minimize(pypesto_problem_original, n_
↪starts=100)
```

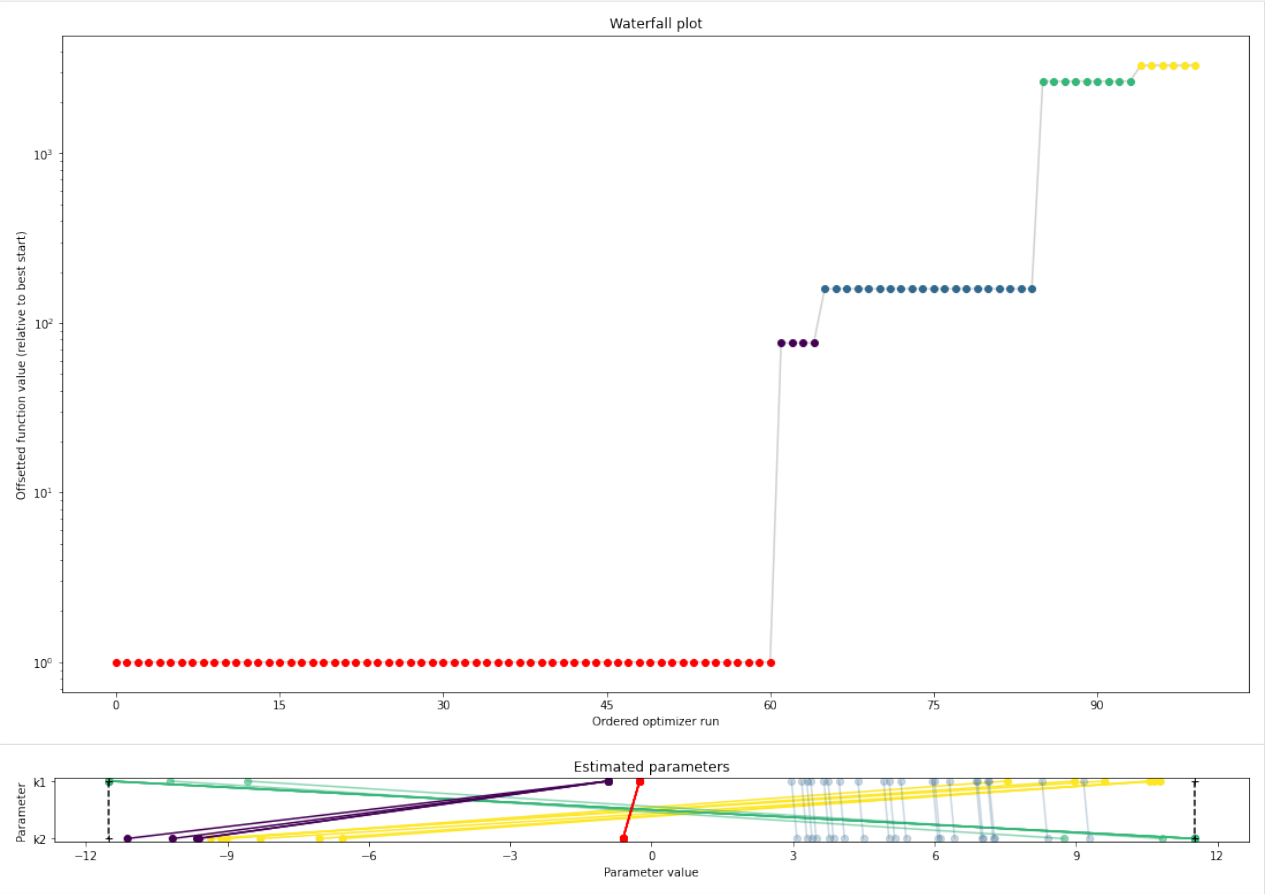
```
Parameters obtained from history and optimizer do not match: [-0.25418068 -0.
↪60837086], [-0.25416788 -0.60834112]
```

5. Visualize parameter estimation. Here, estimated values for `k1` and `k2` are shown, then a waterfall plot to indicate optimization quality, then a plot of the estimated parameters from the different starts to indicate identifiability.

Here, parameter estimation appears to have been successful. In the case of problematic parameter estimation, synthetic data can be used to determine whether parameter estimation can be used to identify known parameter values.

[5]:
```
x_mle_unscaled_original = get_x_mle(pypesto_result_original.optimize_result,
                                    pypesto_problem_original,
                                    petab_problem_original,
                                    scaled=False)
print('Parameters are estimated to be (linear scale):')
print('\n'.join([f'{x_id}: {x_value}' for x_id, x_value in x_mle_unscaled_original.
↪items()]))

pypesto.visualize.waterfall(pypesto_result_original);
pypesto.visualize.parameters(pypesto_result_original);
```

```
Parameters are estimated to be (linear scale):
k1: 0.7755615818811391
k2: 0.5442529577589637
```



2.11 Synthetic Optimization

Similar to the standard optimization, except the PEtab measurements table is replaced with synthetic data that is generated from specified parameters, with noise, and then used for optimization.

Here, parameters are specified with a dictionary that is used to update the original PEtab parameters table. An alternative is use a second PEtab YAML file that is identical to the original, except for the parameters table, which would now contain the parameter values to be used for synthetic data generation.

2.11.1 Noise

Noise is added to the simulated data according to the: - noise distribution in the PEtab observables table; - noise formula in the PEtab observables table, which is used to calculate the scale of the noise distribution; and - noise parameters in the PEtab measurements table, which are substituted into the noise formula for measurement-specific noise distribution scales.

6. As before, load a PEtab problem. This time, the parameters table is edited to contain parameters values that will be used for synthetic data generation (`synthetic_parameters`). Then, the simulator is used to generate synthetic data, which replaces the measurements table of the PEtab problem for parameter estimation in the next step.

Here, synthetic data also has noise added (`noise=True`), which is defined by the PEtab problem as described above. A noise scaling factor can also be specified (here, a small value - `noise_scaling_factor=0.01` - is used, to reduce noise such that the synthetic parameters are more likely to be recovered with parameter estimation).

The simulator working directory is then deleted along with its contents.

[6]:
```
petab_problem_synthetic = petab.Problem.from_yaml(petab_yaml_filename)

synthetic_parameters = {'k1': 1.5, 'k2': 2.5}
petab_problem_synthetic.parameter_df[petab.C.NOMINAL_VALUE].update(synthetic_
↪parameters)

simulator = amici.petab_simulate.PetabSimulator(petab_problem_synthetic)
# Optional: the AMICI simulator is provided a model, to avoid recompilation
petab_problem_synthetic.measurement_df = simulator.simulate(
    noise=True,
    noise_scaling_factor=0.01,
    amici_model=pypesto_problem_original.objective.amici_model,
)
simulator.remove_working_dir()
```

7. Create a pyPESTO problem from the edited PEtab problem, and estimate parameters.

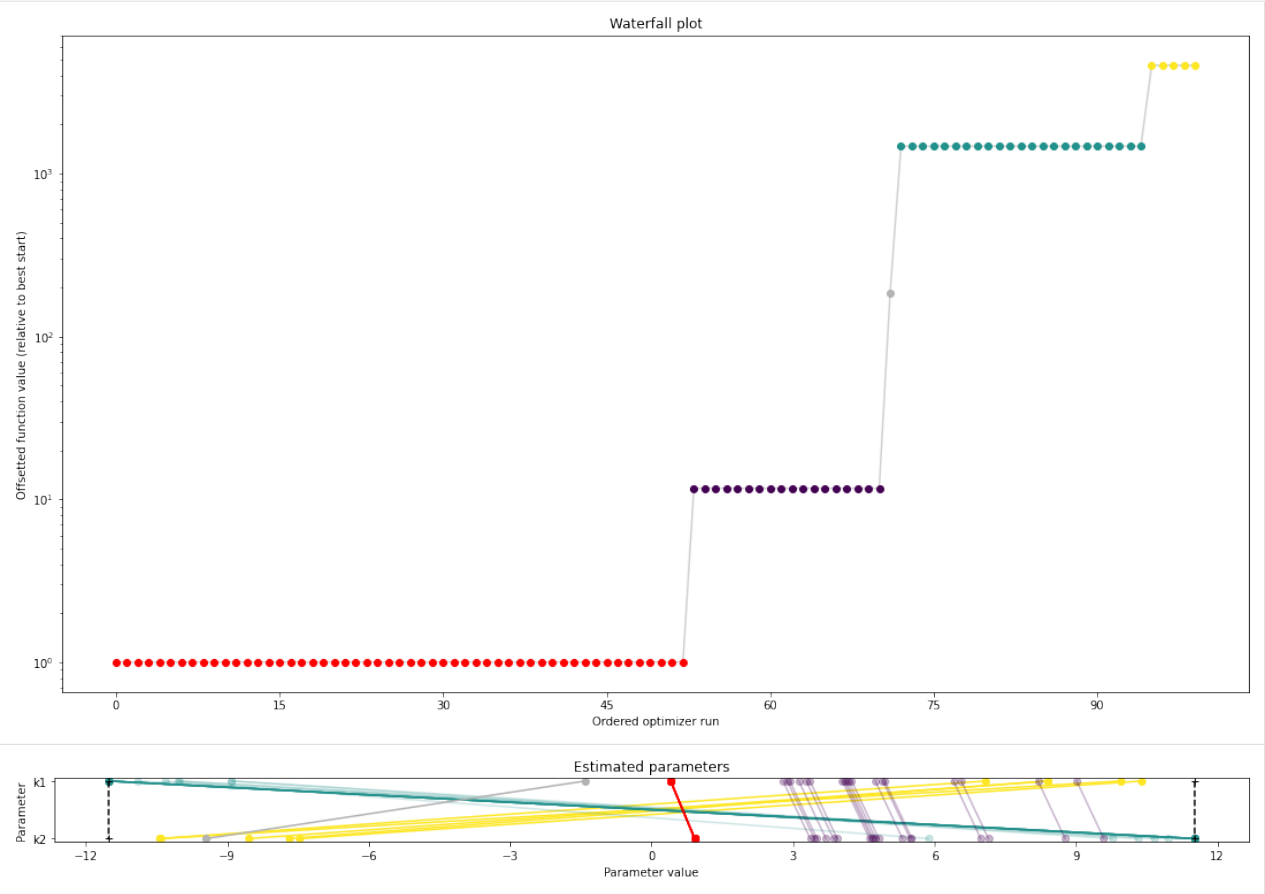
[7]:
```
pypesto_importer_synthetic = pypesto.petab.PetabImporter(petab_problem_synthetic)
pypesto_problem_synthetic = pypesto_importer_synthetic.create_problem()
pypesto_result_synthetic = pypesto.optimize.minimize(pypesto_problem_synthetic, n_
↪starts=100)
```

```
Function values from history and optimizer do not match: -24.31965832797165, 1439.
↪3853896684805
Parameters obtained from history and optimizer do not match: [0.10235171 0.50828654],␣
↪[-10.9022877   11.51292546]
```

8. Visualize parameter estimation. Here, the estimates for `k1` and `k2` are similar to the synthetic parameters, suggesting that parameter estimation works well with this PEtab problem and can be used to identify parameter values successfully (**caveat**: noise is reduced here; parameter estimation can be expected to perform worse with more realistically noisy data).

[8]:
```
x_mle_unscaled_synthetic = get_x_mle(pypesto_result_synthetic.optimize_result,
                                     pypesto_problem_synthetic,
                                     petab_problem_synthetic,
                                     scaled=False)
print('Parameters are estimated to be (linear scale):')
print('\n'.join([f'{x_id}: {x_value}' for x_id, x_value in x_mle_unscaled_synthetic.
↪items()]))

pypesto.visualize.waterfall(pypesto_result_synthetic);
pypesto.visualize.parameters(pypesto_result_synthetic);
```

```
Parameters are estimated to be (linear scale):
k1: 1.4969201262521281
k2: 2.494299196042553
```



2.12 Definition of Priors:

In this notebook we demonstrate how to include prior knowledge into a parameter inference problem, in particular how to define (log-)priors for parameters. If you want to maximize your posterior distribution, you need to define

- A (negative log-)likelihood
- A (log-)prior

The posterior is then built as an `AggregatedObjective`. If you import a problem via `PEtab` and the priors are contained in the parameter table, the definition of priors is done automatically.

CAUTION: The user needs to specify the **negative** *log-likelihood*, while the *log-prior* is internally mulitplied by -1.

[1]:
```
import numpy as np
import scipy as sp

import pypesto
```

2.12.1 Example: Rosenbrock Banana

We will use the Rosenbrock Banana

$$f(x,\theta) = \sum_{i=1}^{N} \underbrace{100 \cdot (x_i - x_{i-1}^2)^2}_{\text{"negative log-likelihood"}} + \underbrace{(x_{i-1} - 1)^2}_{\text{"Gaussian log-prior"}} \tag{2.1}$$

as an example. Here we interpret the first term as the *negative log-likelihood* and the second term as Gaussian *log-prior* with mean 1 and standard deviation $1/\sqrt{2}$.

Note that the second term is only equivalent to the negative log-distribution of a Gaussian up to a constant.

Define the negative log-likelihood

[2]:
```
n_x = 5


def rosenbrock_part_1(x):
    """
    Calculate obj. fct + gradient of the "likelihood" part.
    """
    obj = sum(100.0*(x[1:] - x[:-1]**2.0)**2.0)

    grad = np.zeros_like(x)
    grad[:-1] += -400 * (x[1:] - x[:-1]**2.0) * x[:-1]
    grad[1:] += 200 * (x[1:] - x[:-1]**2.0)

    return (obj, grad)

neg_log_likelihood = pypesto.Objective(fun=rosenbrock_part_1, grad=True)
```

Define the log-prior

A prior on an individual paramater is defined in a `prior_dict`, which contains the following key-value pairs:

- `index`: Index of the parameter
- `density_fun`: (Log-)posterior. (Scalar function!)
- `density_dx`: d/dx (Log-)posterior (optional)
- `density_ddx`: $d^{2/dx}2$ (Log-)posterior (optional)

A `prior_dict` can be either obtained by `get_parameter_prior_dict` for several common priors, or defined by the user.

[3]:
```
from pypesto.objective.priors import get_parameter_prior_dict

# create a list of prior dicts...
prior_list = []
mean = 1
std_dev = 1 / np.sqrt(2)

for i in range(n_x-1):
    prior_list.append(get_parameter_prior_dict(i, 'normal', [mean, std_dev]))
```

(continues on next page)

(continued from previous page)

```
# create the prior
neg_log_prior = pypesto.objective.NegLogParameterPriors(prior_list)
```

Define the negative log-posterior and the problem

The negative log-posterior is defined as an `AggregatedObjective`. Since optimization/visualization is not the main focus of this notebook, the reader is referred to other examples for a more in-depth presentation of these.

[4]:
```
neg_log_posterior = pypesto.objective.AggregatedObjective([neg_log_likelihood, neg_
↪log_prior])

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

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

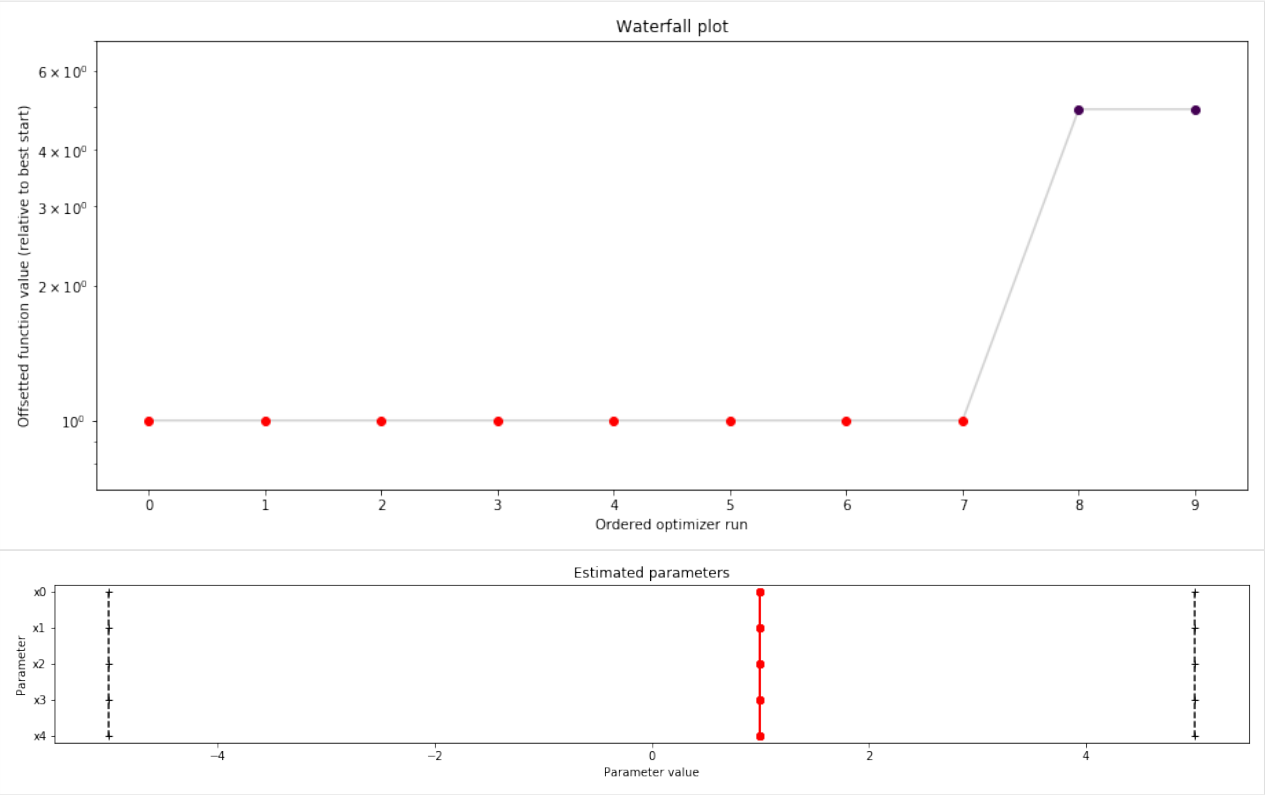
Optimize

[5]:
```
import pypesto.optimize as optimize

result = optimize.minimize(problem=problem, n_starts=10)
```

Some basic visualizations

[6]:
```
import pypesto.visualize as visualize

visualize.waterfall(result, size=(15,6))

# parallel coordinates plot for best 5 fits
visualize.parameters(result, start_indices=5)
```

[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x12faa1908>
```



[ ]:

2.13 Save and load results as HDF5 files

[24]:
```
import pypesto
import numpy as np
import scipy as sp
import pypesto.optimize as optimize
import matplotlib.pyplot as plt
import pypesto.store as store
import pypesto.profile as profile
import pypesto.sample as sample
import tempfile

%matplotlib inline
```

In this notebook, we will demonstrate how to save and (re-)load optimization results, profile results and sampling results to an `.hdf5` file. The use case of this notebook is to generate visualizations from reloaded result objects.

2.13.1 Define the objective and problem

[25]:
```
objective = pypesto.Objective(fun=sp.optimize.rosen,
                              grad=sp.optimize.rosen_der,
                              hess=sp.optimize.rosen_hess)

dim_full = 20
lb = -5 * np.ones((dim_full, 1))
ub = 5 * np.ones((dim_full, 1))

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

2.13.2 Fill result object with profile, sample, optimization

[26]:
```
# create optimizers
optimizer = optimize.ScipyOptimizer()

# set number of starts
n_starts = 10

# Optimization
result = pypesto.optimize.minimize(
    problem=problem, optimizer=optimizer,
    n_starts=n_starts)
# Profiling
result = profile.parameter_profile(
    problem=problem, result=result,
    optimizer=optimizer)
# Sampling
sampler = sample.AdaptiveMetropolisSampler()
result = sample.sample(problem=problem,
                       sampler=sampler,
                       n_samples=100000,
                       result=result)
```

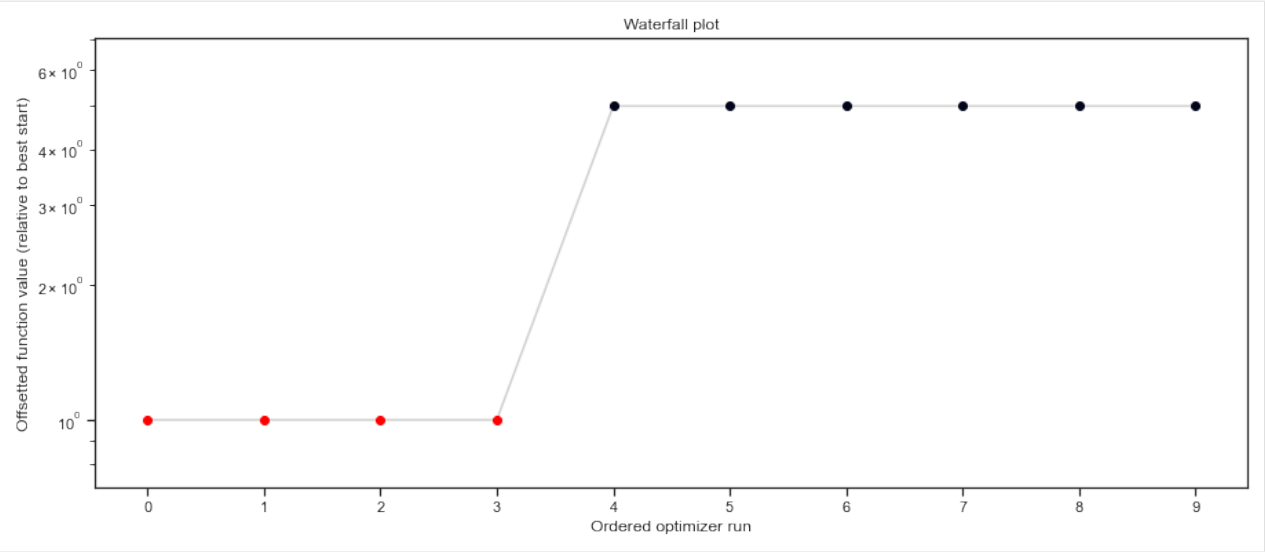
```
100%|| 100000/100000 [00:29<00:00, 3431.32it/s]
```

2.13.3 Plot results

We now want to plot the results (before saving).

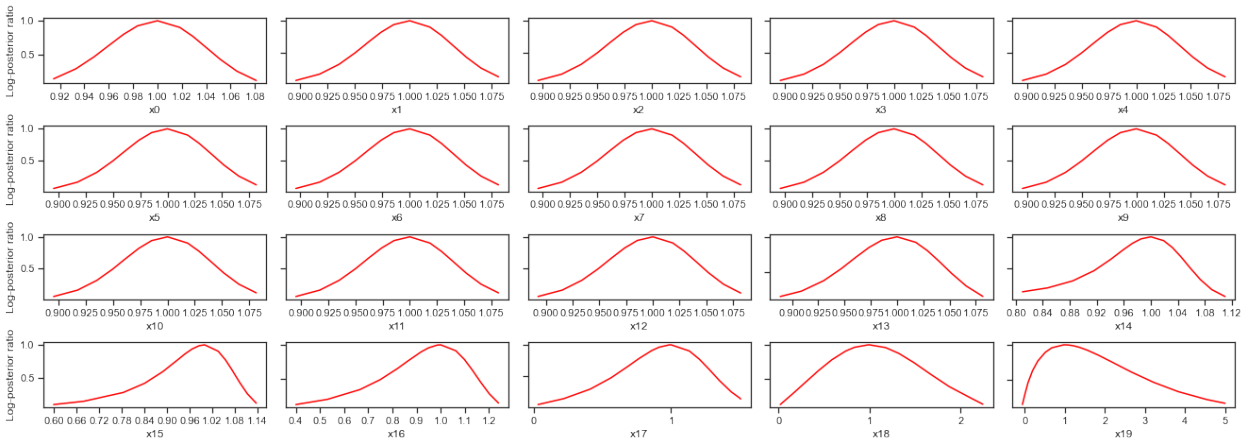
[27]:
```
import pypesto.visualize

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

[27]:
```
<AxesSubplot:title={'center':'Waterfall plot'}, xlabel='Ordered optimizer run',␣
↪ylabel='Offsetted function value (relative to best start)'>
```



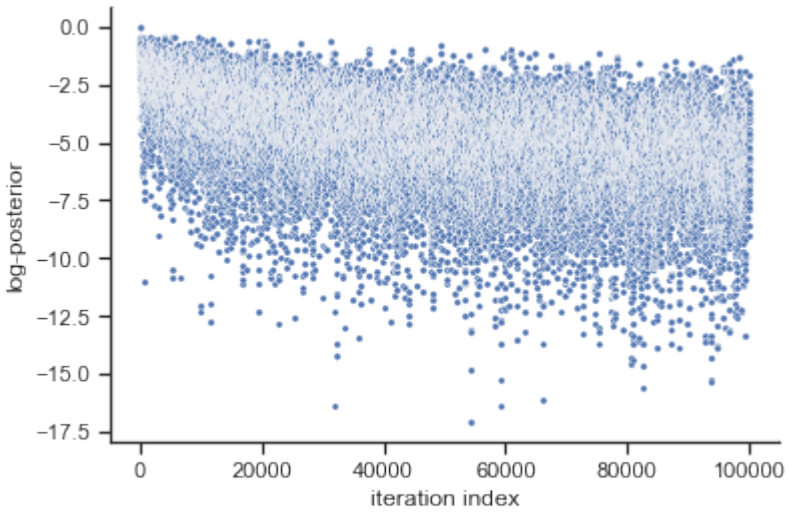
[28]:
```
# plot profiles
pypesto.visualize.profiles(result)
```

[28]:
```
[<AxesSubplot:xlabel='x0', ylabel='Log-posterior ratio'>,
 <AxesSubplot:xlabel='x1'>,
 <AxesSubplot:xlabel='x2'>,
 <AxesSubplot:xlabel='x3'>,
 <AxesSubplot:xlabel='x4'>,
 <AxesSubplot:xlabel='x5', ylabel='Log-posterior ratio'>,
 <AxesSubplot:xlabel='x6'>,
 <AxesSubplot:xlabel='x7'>,
 <AxesSubplot:xlabel='x8'>,
 <AxesSubplot:xlabel='x9'>,
 <AxesSubplot:xlabel='x10', ylabel='Log-posterior ratio'>,
 <AxesSubplot:xlabel='x11'>,
 <AxesSubplot:xlabel='x12'>,
 <AxesSubplot:xlabel='x13'>,
 <AxesSubplot:xlabel='x14'>,
 <AxesSubplot:xlabel='x15', ylabel='Log-posterior ratio'>,
 <AxesSubplot:xlabel='x16'>,
 <AxesSubplot:xlabel='x17'>,
 <AxesSubplot:xlabel='x18'>,
 <AxesSubplot:xlabel='x19'>]
```



[29]:
```
# plot samples
pypesto.visualize.sampling_fval_traces(result)
```

[29]:
```
<AxesSubplot:xlabel='iteration index', ylabel='log-posterior'>
```



2.13.4 Save result object in HDF5 File

[30]:
```
# create temporary file
fn = tempfile.mktemp(".hdf5")

# write result with write_result function.
# Choose which parts of the result object to save with
# corresponding booleans.
store.write_result(result=result,
                   filename=fn,
                   problem=True,
                   optimize=True,
                   profile=True,
                   sample=True)
```

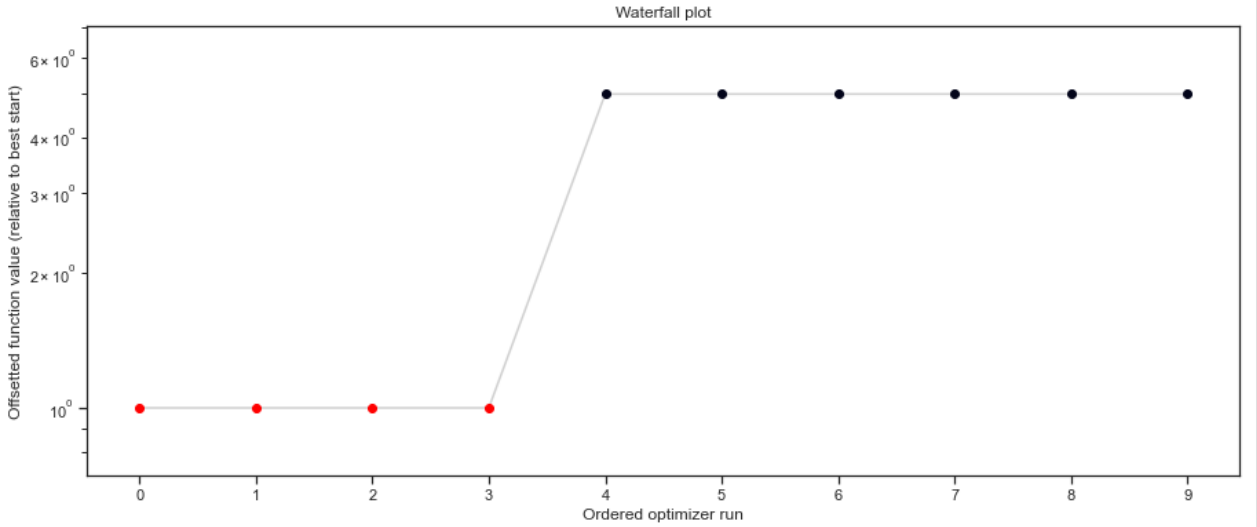
2.13.5 Reload results

[31]:
```
# Read result
result2 = store.read_result(fn)
```

2.13.6 Plot (reloaded) results

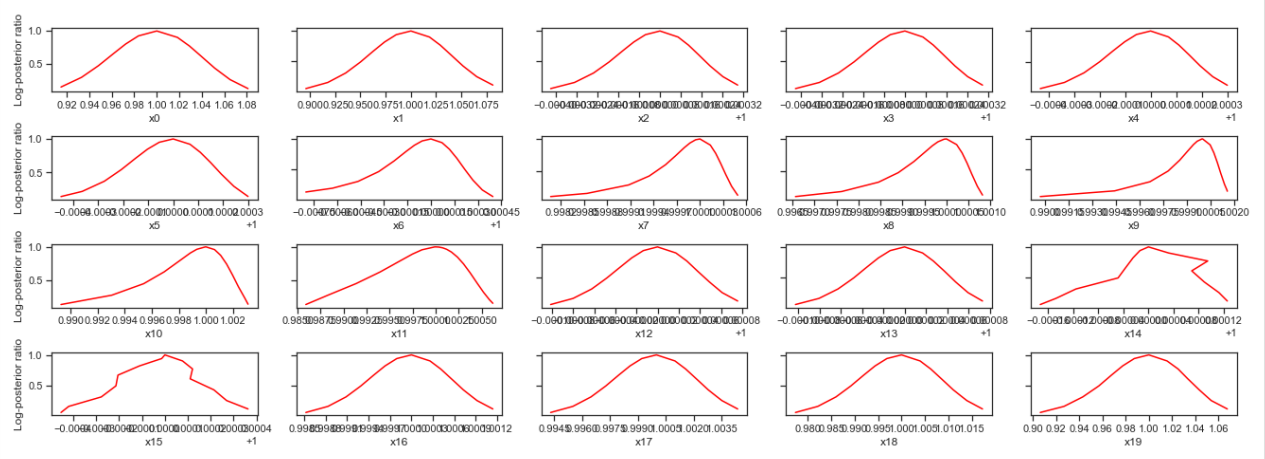
[32]:
```
# plot waterfalls
pypesto.visualize.waterfall(result2, size=(15,6))
```

[32]:
```
<AxesSubplot:title={'center':'Waterfall plot'}, xlabel='Ordered optimizer run',␣
↪ylabel='Offsetted function value (relative to best start)'>
```



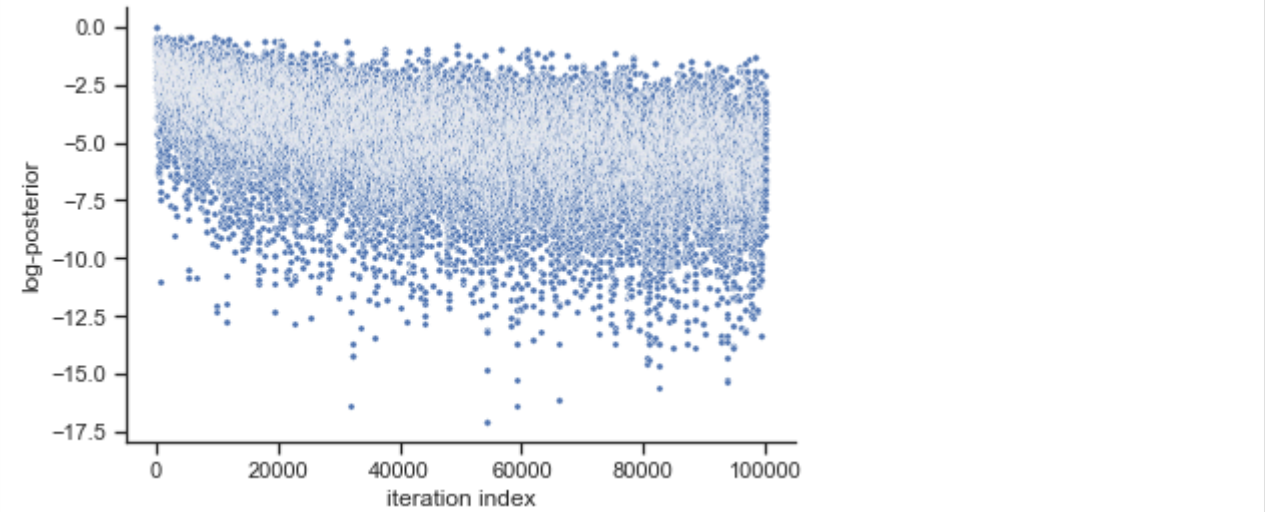
[33]:
```
# plot profiles
pypesto.visualize.profiles(result2)
```

[33]:
```
[<AxesSubplot:xlabel='x0', ylabel='Log-posterior ratio'>,
 <AxesSubplot:xlabel='x1'>,
 <AxesSubplot:xlabel='x2'>,
 <AxesSubplot:xlabel='x3'>,
 <AxesSubplot:xlabel='x4'>,
 <AxesSubplot:xlabel='x5', ylabel='Log-posterior ratio'>,
 <AxesSubplot:xlabel='x6'>,
 <AxesSubplot:xlabel='x7'>,
 <AxesSubplot:xlabel='x8'>,
 <AxesSubplot:xlabel='x9'>,
 <AxesSubplot:xlabel='x10', ylabel='Log-posterior ratio'>,
 <AxesSubplot:xlabel='x11'>,
 <AxesSubplot:xlabel='x12'>,
 <AxesSubplot:xlabel='x13'>,
 <AxesSubplot:xlabel='x14'>,
 <AxesSubplot:xlabel='x15', ylabel='Log-posterior ratio'>,
 <AxesSubplot:xlabel='x16'>,
 <AxesSubplot:xlabel='x17'>,
 <AxesSubplot:xlabel='x18'>,
 <AxesSubplot:xlabel='x19'>]
```



[34]:
```
# plot samples
pypesto.visualize.sampling_fval_traces(result2)
```

[34]:
```
<AxesSubplot:xlabel='iteration index', ylabel='log-posterior'>
```



For the saving of optimization history, we refer to *store.ipynb*.

2.14 Download the examples as notebooks

- Rosenbrock
- Conversion reaction
- Fixed parameters
- Boehm model
- Petab import
- Storage
- Sampler study
- Sampling diagnostics

- `Synthetic data`
- `Prior definition`
- `hdf5 storage`

Note: Some of the notebooks have extra dependencies.

CHAPTER

THREE

STORAGE

It is important to be able to store analysis results efficiently, easily accessible, and portable across systems. For this aim, pyPESTO allows to store results in efficient, portable HDF5 files. Further, optimization trajectories can be stored using various backends, including HDF5 and CSV.

In the following, describe the file formats. For detailed information on usage, consult the `doc/example/hdf5_storage.ipynb` and `doc/example/store.ipynb` notebook, and the API documentation for the `pypesto.objective.history` and `pypesto.storage` modules.

3.1 pyPESTO Problem

```
+ /problem/
  - Attributes:
    - filled by objective.get_config()
    - ...

  - lb [float n_par]
  - ub [float n_par]
  - lb_full [float n_par_full]
  - ub_full [float n_par_full]
  - dim [int]
  - dim_full [int]
  - x_fixed_values [float (n_par_full-n_par)]
  - x_fixed_indices [int (n_par_full-n_par)]
  - x_free_indices [int n_par]
  - x_names [str n_par_full]
```

3.2 Parameter estimation

3.2.1 Parameter estimation settings

Parameter estimation settings are saved in `/optimization/settings`.

3.2.2 Parameter estimation results

Parameter estimation results are saved in `/optimization/results/`.

Results per local optimization

Results of the `$n`'th multistart a saved in the format

```
+ /optimization/results/$n/
  - fval: [float]
      Objective function value of best iteration
  - x: [float n_par_full]
      Parameter set of best iteration
  - grad: [float n_par_full]
      Gradient of objective function at point x
  - hess: [float n_par_full x n_par_full]
      Hessian matrix of objective function at point x
  - n_fval: [int]
      Total number of objective function evaluations
  - n_grad: [int]
      Number of gradient evaluations
  - n_hess: [int]
      Number of Hessian evaluations
  - x0: [float n_par_full]
      Initial parameter set
  - fval0: [float]
      Objective function value at starting parameters
  - exitflag: [str] Some exit flag
  - time: [float] Execution time
  - message: [str] Some exit message
```

Trace per local optimization

When objective function call histories are saved to HDF5, they are under `/optimization/results/$n/trace/`.

```
+ /optimization/results/$n/trace/
  - fval: [float n_iter]
      Objective function value of best iteration
  - x: [float n_iter x n_par_full]
      Parameter set of best iteration
  - grad: [float n_iter x n_par_full]
      Gradient of objective function at point x
  - hess: [float n_iter x n_par_full x n_par_full]
      Hessian matrix of objective function at point x
  - time: [float n_iter] Executition time
  - chi2: [float n_iter x ...]
  - schi2: [float n_iter x ...]
```

3.3 Sampling

3.3.1 Sampling results

Sampling results are saved in `/sampling/results/`.

```
+ /sampling/results/
  - betas [float n_chains]
  - trace_neglogpost [float n_chains x (n_samples+1)]
  - trace_neglogprior [float n_chains x (n_samples+1)]
  - trace_x [float n_chains x (n_samples+1) x n_par]
  - Attributes:
    - time
```

3.4 Profiling

3.4.1 Profiling results

Profiling results are saved in `/profiling/$profiling_id/`, where `profiling_id` indicates the number of profilings done.

```
+/profiling/profiling_id/
  - $parameter_index/
    - exitflag_path [float n_iter]
    - fval_path [float n_iter]
    - gradnorm_path [float n_iter]
    - ratio_path [float n_iter]
    - time_path [float n_iter]
    - x_path [float n_par x n_iter]
    - Attributes:
      - time_total
      - IsNone
      - n_fval
      - n_grad
      - n_hess
```

CHAPTER

FOUR

API REFERENCE

4.1 pyPESTO

Parameter Estimation TOolbox for python.

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

Bases: pypesto.objective.base.ObjectiveBase

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

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

Constructor.

Parameters

- **amici_model** – The amici model.
- **amici_solver** – The solver to use for the numeric integration of the model.
- **edatas** – The experimental data. If a list is passed, its entries correspond to multiple experimental conditions.
- **max_sensi_order** – Maximum sensitivity order supported by the model. Defaults to 2 if the model was compiled with o2mode, otherwise 1.
- **x_ids** – Ids of optimization parameters. In the simplest case, this will be the AMICI model parameters (default).
- **x_names** – Names of optimization parameters.

- **parameter_mapping** – Mapping of optimization parameters to model parameters. Format as created by *amici.petab_objective.create_parameter_mapping*. The default is just to assume that optimization and simulation parameters coincide.
- **guess_steadystate** – Whether to guess steadystates based on previous steadystates and respective derivatives. This option may lead to unexpected results for models with conservation laws and should accordingly be deactivated for those models.
- **n_threads** – Number of threads that are used for parallelization over experimental conditions. If amici was not installed with openMP support this option will have no effect.
- **fim_for_hess** – Whether to use the FIM whenever the Hessian is requested. This only applies with forward sensitivities. With adjoint sensitivities, the true Hessian will be used, if available. FIM or Hessian will only be exposed if *max_sensi_order>1*.
- **amici_object_builder** – AMICI object builder. Allows recreating the objective for pickling, required in some parallelization schemes.
- **calculator** – Performs the actual calculation of the function values and derivatives.

apply_custom_timepoints()
Apply custom timepoints, if applicable.

See the *set_custom_timepoints* method for more information.

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

call_unprocessed(*x*, *sensi_orders*, *mode*, *edatas=None*, *parameter_mapping=None*)
Call objective function without pre- or post-processing and formatting.

Parameters

- **x** – The parameters for which to evaluate the objective function.
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns A dict containing the results.

Return type result

check_mode(*mode*)
Check if the objective is able to compute in the requested mode.

Parameters **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether mode is supported

Return type flag

check_sensi_orders(*sensi_orders*, *mode*) → bool
Check if the objective is able to compute the requested sensitivities.

Parameters

- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether combination of sensi_orders and mode is supported

Return type flag

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

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

reset_steadystate_guesses()
Resets all steadystate guess data

set_custom_timepoints(*timepoints: Optional[Sequence[Sequence[Union[float, int]]]] = None*, *timepoints_global: Optional[Sequence[Union[float, int]]] = None*) → pypesto.objective.amici.AmiciObjective
Create a copy of this objective that will be evaluated at custom timepoints.

The intended use is to aid in predictions at unmeasured timepoints.

Parameters

- **timepoints** – The outer sequence should contain a sequence of timepoints for each experimental condition.
- **timepoints_global** – A sequence of timepoints that will be used for all experimental conditions.

Returns

Return type The customized copy of this objective.

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

class pypesto.**AmiciPredictor**(*amici_objective: pypesto.objective.amici.AmiciObjective*, *amici_output_fields: Optional[Sequence[str]] = None*, *post_processor: Optional[Callable] = None*, *post_processor_sensi: Optional[Callable] = None*, *post_processor_time: Optional[Callable] = None*, *max_chunk_size: Optional[int] = None*, *output_ids: Optional[Sequence[str]] = None*, *condition_ids: Optional[Sequence[str]] = None*)
Bases: object

Do forward simulations (predictions) with parameter vectors, for an AMICI model. The user may supply post-processing methods. If post-processing methods are supplied, and a gradient of the prediction is requested, then the sensitivities of the AMICI model must also be post-processed. There are no checks here to ensure that the sensitivities are correctly post-processed, this is explicitly left to the user. There are also no safeguards if the postprocessor routines fail. This may happen if, e.g., a call to Amici fails, and no timepoints, states or observables are returned. As the AmiciPredictor is agnostic about the dimension of the postprocessor and also the dimension of the postprocessed output, these checks are also left to the user. An example for such a check is provided in the default output (see _default_output()).

__call__(*x: numpy.ndarray*, *sensi_orders: Tuple[int, ...] = (0)*, *mode: str = 'mode_fun'*, *output_file: str = ''*, *output_format: str = 'csv'*) → pypesto.predict.result.PredictionResult
Simulate a model for a certain prediction function. This method relies on the AmiciObjective, which is underlying, but allows the user to apply any post-processing of the results, the sensitivities, and the timepoints.

Parameters

- **x** – The parameters for which to evaluate the prediction function.
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.

- **mode** – Whether to compute function values or residuals.
- **output_file** – Path to an output file.
- **output_format** – Either 'csv', 'h5'. If an output file is specified, this routine will return a csv file, created from a DataFrame, or an h5 file, created from a dict.

Returns PredictionResult object containing timepoints, outputs, and output_sensitivities if requested

Return type results

__init__(*amici_objective: pypesto.objective.amici.AmiciObjective*, *amici_output_fields: Optional[Sequence[str]] = None*, *post_processor: Optional[Callable] = None*, *post_processor_sensi: Optional[Callable] = None*, *post_processor_time: Optional[Callable] = None*, *max_chunk_size: Optional[int] = None*, *output_ids: Optional[Sequence[str]] = None*, *condition_ids: Optional[Sequence[str]] = None*)

Constructor.

Parameters

- **amici_objective** – An objective object, which will be used to get model simulations
- **amici_output_fields** – keys that exist in the return data object from AMICI, which should be available for the post-processors
- **post_processor** – A callable function which applies postprocessing to the simulation results and possibly defines different outputs than those of the amici model. Default are the observables (*pypesto.predict.constants.AMICI_Y*) of the AMICI model. This method takes a list of dicts (with the returned fields *pypesto.predict.constants.AMICI_T*, *pypesto.predict.constants.AMICI_X*, and *pypesto.predict.constants.AMICI_Y* of the AMICI ReturnData objects) as input. Safeguards for, e.g., failure of AMICI are left to the user.
- **post_processor_sensi** – A callable function which applies postprocessing to the sensitivities of the simulation results. Defaults to the observable sensitivities of the AMICI model. This method takes a list of dicts (with the returned fields *pypesto.predict.constants.AMICI_T*, *pypesto.predict.constants.AMICI_X*, *pypesto.predict.constants.AMICI_Y*, *pypesto.predict.constants.AMICI_SX*, and *pypesto.predict.constants.AMICI_SY* of the AMICI ReturnData objects) as input. Safeguards for, e.g., failure of AMICI are left to the user.
- **post_processor_time** – A callable function which applies postprocessing to the timepoints of the simulations. Defaults to the timepoints of the amici model. This method takes a list of dicts (with the returned field *pypesto.predict.constants.AMICI_T* of the amici ReturnData objects) as input. Safeguards for, e.g., failure of AMICI are left to the user.
- **max_chunk_size** – In some cases, we don't want to compute all predictions at once when calling the prediction function, as this might not fit into the memory for large datasets and models. Here, the user can specify a maximum chunk size of conditions, which should be simulated at a time. Defaults to None, meaning that all conditions will be simulated.
- **output_ids** – IDs of outputs, as post-processing allows the creation of customizable outputs, which may not coincide with those from the AMICI model (defaults to AMICI observables).
- **condition_ids** – List of identifiers for the conditions of the edata objects of the amici objective, will be passed to the PredictionResult at call.

class pypesto.**CsvHistory**(*file: str, x_names: Optional[Sequence[str]] = None, options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None, load_from_file: bool = False*)

Bases: pypesto.objective.history.History

Stores a representation of the history in a CSV file.

Parameters

- **file** – CSV file name.
- **x_names** – Parameter names.
- **options** – History options.
- **load_from_file** – If True, history will be initialized from data in the specified file

__init__(*file: str, x_names: Optional[Sequence[str]] = None, options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None, load_from_file: bool = False*)

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

finalize()

Finalize history. Called after a run.

get_chi2_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]

Chi2 values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_fval_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]

Function values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_grad_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]

Gradients.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_hess_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]

Hessians.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_res_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]

Residuals.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_schi2_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]

Chi2 sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_sres_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Residual sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_time_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Cumulative execution times.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_x_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Parameters.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

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

Parameters

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

class pypesto.**Hdf5History**(*id: str*, *file: str*, *options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*)
Bases: pypesto.objective.history.History

Stores a representation of the history in an HDF5 file.

Parameters

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

__init__(*id: str*, *file: str*, *options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*)
Initialize self. See help(type(self)) for accurate signature.

finalize()
Finalize history. Called after a run.

get_chi2_trace() → Sequence[numpy.ndarray]
Chi2 values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_fval_trace() → Sequence[float]
Function values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_grad_trace() → Sequence[numpy.ndarray]
Gradients.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_hess_trace() → Sequence[numpy.ndarray]
Hessians.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_history_directory()

get_res_trace() → Sequence[numpy.ndarray]
Residuals.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

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

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_sres_trace() → Sequence[numpy.ndarray]
Residual sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_time_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Cumulative execution times.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_x_trace() → Sequence[numpy.ndarray]
Parameters.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

static load(*id: str*, *file: str*)
Loads the History object from memory.

property n_fval
Number of function evaluations.

property n_grad
Number of gradient evaluations.

property n_hess
Number of Hessian evaluations.

property n_res
Number of residual evaluations.

property n_sres
Number or residual sensitivity evaluations.

property trace_save_iter

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

Parameters

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

class pypesto.**History**(*options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*)
Bases: pypesto.objective.history.HistoryBase

Tracks numbers of function evaluations only, no trace.

Parameters **options** – History options.

__init__(*options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*)
Initialize self. See help(type(self)) for accurate signature.

finalize()
Finalize history. Called after a run.

property n_fval
Number of function evaluations.

property n_grad
Number of gradient evaluations.

property n_hess
Number of Hessian evaluations.

property n_res
Number of residual evaluations.

property n_sres
Number or residual sensitivity evaluations.

property start_time
Start time.

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

Parameters

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

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

Abstract base class for history objects.

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

finalize()
Finalize history. Called after a run.

get_chi2_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[float], float]
Chi2 values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_fval_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[float], float]
Function values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_grad_trace(*ix: Optional[Union[int, Sequence[int]]] = None*) → Union[Sequence[Union[numpy.ndarray, np.nan]], numpy.ndarray, np.nan]
Gradients.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_hess_trace(*ix: Optional[Union[int, Sequence[int]]] = None*) → Union[Sequence[Union[numpy.ndarray, np.nan]], numpy.ndarray, np.nan]
Hessians.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_res_trace(*ix: Optional[Union[int, Sequence[int]]] = None*) → Union[Sequence[Union[numpy.ndarray, np.nan]], numpy.ndarray, np.nan]
Residuals.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_schi2_trace(*ix: Optional[Union[int, Sequence[int]]] = None*) → Union[Sequence[Union[numpy.ndarray, np.nan]], numpy.ndarray, np.nan]
Chi2 sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_sres_trace(*ix: Optional[Union[int, Sequence[int]]] = None*) → Union[Sequence[Union[numpy.ndarray, np.nan]], numpy.ndarray, np.nan]
Residual sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_time_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[float], float]
Cumulative execution times.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_x_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[numpy.ndarray], numpy.ndarray]

Parameters.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

property n_fval

Number of function evaluations.

property n_grad

Number of gradient evaluations.

property n_hess

Number of Hessian evaluations.

property n_res

Number of residual evaluations.

property n_sres

Number or residual sensitivity evaluations.

property start_time

Start time.

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

Update history after a function evaluation.

Parameters

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

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

Bases: dict

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

In addition implements a factory pattern to generate history objects.

Parameters

- **trace_record** – Flag indicating whether to record the trace of function calls. The trace_record_* flags only become effective if trace_record is True. Default: False.
- **trace_record_grad** – Flag indicating whether to record the gradient in the trace. Default: True.
- **trace_record_hess** – Flag indicating whether to record the Hessian in the trace. Default: False.
- **trace_record_res** – Flag indicating whether to record the residual in the trace. Default: False.

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

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

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

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

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

Parameters

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

class pypesto.**MemoryHistory**(*options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*)
Bases: pypesto.objective.history.History

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

Parameters **options** – History options.

__init__(*options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*)
Initialize self. See help(type(self)) for accurate signature.

get_chi2_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Chi2 values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_fval_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Function values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_grad_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Gradients.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_hess_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Hessians.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_res_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Residuals.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_schi2_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Chi2 sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_sres_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Residual sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_time_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Cumulative execution times.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_x_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Parameters.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

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

Parameters

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

class pypesto.**NegLogPriors**(*objectives: Sequence[pypesto.objective.base.ObjectiveBase]*, *x_names: Optional[Sequence[str]] = None*)
Bases: pypesto.objective.aggregated.AggregatedObjective

Aggregates different forms of negative log-prior distributions.

Allows to distinguish priors from the likelihood by testing the type of an objective.

Consists basically of a list of individual negative log-priors, given in self.objectives.

class pypesto.**Objective**(*fun: Optional[Callable] = None*, *grad: Optional[Union[Callable, bool]] = None*, *hess: Optional[Callable] = None*, *hessp: Optional[Callable] = None*, *res: Optional[Callable] = None*, *sres: Optional[Union[Callable, bool]] = None*, *x_names: Optional[Sequence[str]] = None*)

Bases: pypesto.objective.base.ObjectiveBase

The objective class allows the user explicitely specify functions that compute the function value and/or residuals as well as respective derivatives.

Parameters

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- **x_names** – Parameter names. None if no names provided, otherwise a list of str, length dim_full (as in the Problem class). Can be read by the problem.

__init__(*fun: Optional[Callable] = None*, *grad: Optional[Union[Callable, bool]] = None*, *hess: Optional[Callable] = None*, *hessp: Optional[Callable] = None*, *res: Optional[Callable] = None*, *sres: Optional[Union[Callable, bool]] = None*, *x_names: Optional[Sequence[str]] = None*)

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

call_unprocessed(*x*, *sensi_orders*, *mode*)

Call objective function without pre- or post-processing and formatting.

Returns A dict containing the results.

Return type result

check_mode(*mode*)
Check if the objective is able to compute in the requested mode.

Parameters **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether mode is supported

Return type flag

check_sensi_orders(*sensi_orders*, *mode*)
Check if the objective is able to compute the requested sensitivities.

Parameters

- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether combination of sensi_orders and mode is supported

Return type flag

property has_fun

property has_grad

property has_hess

property has_hessp

property has_res

property has_sres

class pypesto.**ObjectiveBase**(*x_names: Optional[Sequence[str]] = None*)
Bases: abc.ABC

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

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

Parameters **x_names** – Parameter names that can be optionally used in, e.g., history or gradient checks

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

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

__call__(*x: numpy.ndarray*, *sensi_orders: Tuple[int, ...] = (0)*, *mode: str = 'mode_fun'*, *return_dict: bool = False*, ***kwargs*) → Union[float, numpy.ndarray, Tuple, Dict[str, Union[float, numpy.ndarray, Dict]]]
Method to obtain arbitrary sensitivities. This is the central method which is always called, also by the get_* methods.

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

Parameters

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

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

Return type result

__init__(*x_names: Optional[Sequence[str]] = None*)
Initialize self. See help(type(self)) for accurate signature.

abstract call_unprocessed(*x: numpy.ndarray*, *sensi_orders: Tuple[int, ...]*, *mode: str*, ***kwargs*) → Dict[str, Union[float, numpy.ndarray, Dict]]
Call objective function without pre- or post-processing and formatting.

Parameters

- **x** – The parameters for which to evaluate the objective function.
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns A dict containing the results.

Return type result

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

Parameters

- **x** – The parameters for which to evaluate the gradient.
- **x_indices** – Indices for which to compute gradients. Default: all.
- **eps** – Finite differences step size.
- **verbosity** – Level of verbosity for function output. 0: no output, 1: summary for all parameters, 2: summary for individual parameters.
- **mode** – Residual (MODE_RES) or objective function value (MODE_FUN) computation mode.
- **detailed** – Toggle whether additional values are returned. Additional values are function values, and the central difference weighted by the difference in output from all methods (standard deviation and mean).

Returns gradient, finite difference approximations and error estimates.

Return type result

check_grad_multi_eps(**args*, *multi_eps: Optional[Iterable] = None*, *label: str = 'rel_err'*, ***kwargs*)

Equivalent to the *ObjectiveBase.check_grad* method, except multiple finite difference step sizes are tested. The result contains the lowest finite difference for each parameter, and the corresponding finite difference step size.

Parameters

- **ObjectiveBase.check_grad method parameters.** (*All*) –
- **multi_eps** – The finite difference step sizes to be tested.
- **label** – The label of the column that will be minimized for each parameter. Valid options are the column labels of the dataframe returned by the *ObjectiveBase.check_grad* method.

abstract check_mode(*mode*) → bool

Check if the objective is able to compute in the requested mode.

Parameters **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether mode is supported

Return type flag

abstract check_sensi_orders(*sensi_orders*, *mode*) → bool

Check if the objective is able to compute the requested sensitivities.

Parameters

- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether combination of sensi_orders and mode is supported

Return type flag

get_fval(*x: numpy.ndarray*) → float

Get the function value at x.

get_grad(*x: numpy.ndarray*) → numpy.ndarray

Get the gradient at x.

get_hess(*x: numpy.ndarray*) → numpy.ndarray

Get the Hessian at x.

get_res(*x: numpy.ndarray*) → numpy.ndarray

Get the residuals at x.

get_sres(*x: numpy.ndarray*) → numpy.ndarray

Get the residual sensitivities at x.

property has_fun

property has_grad

property has_hess

property has_hessp

property has_res

property has_sres

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

static output_to_tuple(*sensi_orders: Tuple[int, ...]*, *mode: str*, ***kwargs: Union[float, numpy.ndarray]*) → Tuple
Return values as requested by the caller, since usually only a subset is demanded. One output is returned as-is, more than one output are returned as a tuple in order (fval, grad, hess).

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

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

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

Parameters

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

class pypesto.**OptimizeResult**
Bases: object

Result of the minimize() function.

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

append(*optimizer_result: optimize.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.**OptimizerHistory**(*history: pypesto.objective.history.History*, *x0: numpy.ndarray*, *generate_from_history: bool = False*)
Bases: object

Objective call history. Container around a History object, which keeps track of optimal values.

fval0, fval_min
Initial and best function value found.

chi20, chi2_min
Initial and best chi2 value found.

x0, x_min
Initial and best parameters found.

grad_min
gradient for best parameters

hess_min
hessian (approximation) for best parameters

res_min
residuals for best parameters

sres_min
residual sensitivities for best parameters

Parameters

- **history** – History object to attach to this container. This history object implements the storage of the actual history.
- **x0** – Initial values for optimization
- **generate_from_history** – If set to true, this function will try to fill attributes of this function based on the provided history

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

extract_from_history(*var*, *ix*)

finalize()

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

class pypesto.**PredictionConditionResult**(*timepoints: numpy.ndarray*, *output_ids: Sequence[str]*, *output: Optional[numpy.ndarray] = None*, *output_sensi: Optional[numpy.ndarray] = None*, *x_names: Optional[Sequence[str]] = None*)
Bases: object

This class is a light-weight wrapper for the prediction of one simulation condition of an amici model. It should provide a common api how amici predictions should look like in pyPESTO.

__init__(*timepoints: numpy.ndarray*, *output_ids: Sequence[str]*, *output: Optional[numpy.ndarray] = None*, *output_sensi: Optional[numpy.ndarray] = None*, *x_names: Optional[Sequence[str]] = None*)
Constructor.

Parameters

- **timepoints** – Output timepoints for this simulation condition
- **output_ids** – IDs of outputs for this simulation condition
- **outputs** – Postprocessed outputs (ndarray)

- **outputs_sensi** – Sensitivities of postprocessed outputs (ndarray)
- **x_names** – IDs of model parameter w.r.t to which sensitivities were computed

class pypesto.**PredictionResult**(*conditions: Sequence[Union[pypesto.predict.result.PredictionConditionResult, Dict]]*, *condition_ids: Optional[Sequence[str]] = None*, *comment: Optional[str] = None*)

Bases: object

This class is a light-weight wrapper around predictions from pyPESTO made via an amici model. It's only purpose is to have fixed format/api, how prediction results should be stored, read, and handled: as predictions are a very flexible format anyway, they should at least have a common definition, which allows to work with them in a reasonable way.

__init__(*conditions: Sequence[Union[pypesto.predict.result.PredictionConditionResult, Dict]]*, *condition_ids: Optional[Sequence[str]] = None*, *comment: Optional[str] = None*)

Constructor.

Parameters

- **conditions** – A list of PredictionConditionResult objects or dicts
- **condition_ids** – IDs or names of the simulation conditions, which belong to this prediction (e.g., PEtab uses tuples of preequilibration condition and simulation conditions)
- **comment** – An additional note, which can be attached to this prediction

write_to_csv(*output_file: str*)

This method saves predictions to a csv file.

Parameters **output_file** – path to file/folder to which results will be written

write_to_h5(*output_file: str*, *base_path: Optional[str] = None*)

This method saves predictions to an h5 file. It appends to the file if the file already exists.

Parameters

- **output_file** – path to file/folder to which results will be written
- **base_path** – base path in the h5 file

class pypesto.**Problem**(*objective: pypesto.objective.base.ObjectiveBase*, *lb: Union[numpy.ndarray, List[float]]*, *ub: Union[numpy.ndarray, List[float]]*, *dim_full: Optional[int] = None*, *x_fixed_indices: Optional[Union[Iterable[SupportsInt], SupportsInt]] = None*, *x_fixed_vals: Optional[Union[Iterable[SupportsFloat], SupportsFloat]] = None*, *x_guesses: Optional[Iterable[float]] = None*, *startpoint_method: Optional[Callable] = None*, *x_names: Optional[Iterable[str]] = None*, *x_scales: Optional[Iterable[str]] = None*, *x_priors_defs: Optional[pypesto.objective.priors.NegLogPriors] = None*, *lb_init: Optional[Union[numpy.ndarray, List[float]]] = None*, *ub_init: Optional[Union[numpy.ndarray, List[float]]] = 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.
- **lb** – The lower and upper bounds for optimization. For unbounded directions set to +-inf.
- **ub** – The lower and upper bounds for optimization. For unbounded directions set to +-inf.

- **lb_init** – The lower and upper bounds for initialization, typically for defining search start points. If not set, set to lb, ub.
- **ub_init** – The lower and upper bounds for initialization, typically for defining search start points. If not set, set to lb, ub.
- **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.
- **startpoint_method** – Callable. *startpoint_method(n_starts)* returns a n_starts x n_free_indices array of initial values for 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.
- **x_scales** – Parameter scales can be optionally given and are used e.g. in visualisation and prior generation. Currently the scales 'lin', 'log`and 'log10' are supported.
- **x_priors_defs** – Definitions of priors for parameters. Types of priors, and their required and optional parameters, are described in the *Prior* class.

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.

__init__(*objective: pypesto.objective.base.ObjectiveBase*, *lb: Union[numpy.ndarray, List[float]]*, *ub: Union[numpy.ndarray, List[float]]*, *dim_full: Optional[int] = None*, *x_fixed_indices: Optional[Union[Iterable[SupportsInt], SupportsInt]] = None*, *x_fixed_vals: Optional[Union[Iterable[SupportsFloat], SupportsFloat]] = None*, *x_guesses: Optional[Iterable[float]] = None*, *startpoint_method: Optional[Callable] = None*, *x_names: Optional[Iterable[str]] = None*, *x_scales: Optional[Iterable[str]] = None*, *x_priors_defs: Optional[pypesto.objective.priors.NegLogPriors] = None*, *lb_init: Optional[Union[numpy.ndarray, List[float]]] = None*, *ub_init: Optional[Union[numpy.ndarray, List[float]]] = None*)

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

property dim

fix_parameters(*parameter_indices: Union[Iterable[SupportsInt], SupportsInt]*, *parameter_vals: Union[Iterable[SupportsFloat], SupportsFloat]*) → None

Fix specified parameters to specified values

full_index_to_free_index(*full_index: int*)

Calculate index in reduced vector from index in full vector.

Parameters **full_index** (*The index in the full vector.*) –

Returns free_index

Return type The index in the free vector.

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: Optional[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]*, *x_indices: Optional[List[int]] = None*) → Optional[numpy.ndarray]
Keep only those elements, which indices are specified in x_indices If x_indices is not provided, delete fixed indices.

Parameters

- **x_full** (*array_like, ndim=1*) – The vector in dimension dim_full.
- **x_indices** – indices of x_full that should remain

property lb

property lb_init

normalize() → 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 parameter boundaries.

property ub

property ub_init

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

property x_free_indices

property x_guesses

class pypesto.**ProfileResult**
Bases: object

Result of the profile() function.

It holds a list of profile lists. Each profile list consists of a list of *ProfilerResult* objects, one for each parameter.

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

append_empty_profile_list() → int
Append an empty profile list to the list of profile lists.

Returns The index of the created profile list.

Return type index

append_profiler_result(*profiler_result:* profile.ProfilerResult = *None*, *profile_list:* *int* = *None*) → None
Append the profiler result to the profile list.

Parameters

- **profiler_result** – The result of one profiler run for a parameter, or None if to be left empty.
- **profile_list** – Index specifying the profile list to which we want to append. Defaults to the last list.

get_profiler_result(*i_par:* *int*, *profile_list:* *Optional[int]* = *None*)
Get theprofiler result at parameter index *i_par* of profile list *profile_list*.

Parameters

- **i_par** – Integer specifying the profile index.
- **profile_list** – Index specifying the profile list. Defaults to the last list.

set_profiler_result(*profiler_result:* profile.ProfilerResult, *i_par:* *int*, *profile_list:* *int* = *None*) → None
Write a profiler result to the result object at *i_par* of profile list *profile_list*.

Parameters

- **profiler_result** – The result of one (local) profiler run.
- **i_par** – Integer specifying the parameter index.
- **profile_list** – Index specifying the profile list. Defaults to the last list.

class pypesto.**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.

__init__(*problem=None*)
Initialize self. See help(type(self)) for accurate signature.

class pypesto.**SampleResult**
Bases: object

Result of the sample() function.

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

4.2 Objective

class pypesto.objective.**AggregatedObjective**(*objectives: Sequence[pypesto.objective.base.ObjectiveBase]*, *x_names: Optional[Sequence[str]] = None*)
Bases: pypesto.objective.base.ObjectiveBase

This class aggregates multiple objectives into one objective.

__init__(*objectives: Sequence[pypesto.objective.base.ObjectiveBase]*, *x_names: Optional[Sequence[str]] = None*)
Constructor.

Parameters

- **objectives** – Sequence of pypesto.ObjectiveBase instances
- **x_names** – Sequence of names of the (optimized) parameters. (Details see documentation of x_names in *pypesto.ObjectiveBase*)

call_unprocessed(*x*, *sensi_orders*, *mode*) → Dict[str, Union[float, numpy.ndarray, Dict]]
Call objective function without pre- or post-processing and formatting.

Parameters

- **x** – The parameters for which to evaluate the objective function.
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns A dict containing the results.

Return type result

check_mode(*mode*) → bool
Check if the objective is able to compute in the requested mode.

Parameters **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether mode is supported

Return type flag

check_sensi_orders(*sensi_orders*, *mode*) → bool
Check if the objective is able to compute the requested sensitivities.

Parameters

- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether combination of sensi_orders and mode is supported

Return type flag

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

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

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

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

Called within the AmiciObjective.__call__() method.

Parameters

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

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

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

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

Allows to build AMICI model, solver, and edatas.

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

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

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

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

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

Bases: pypesto.objective.base.ObjectiveBase

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

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

Constructor.

Parameters

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

- **calculator** – Performs the actual calculation of the function values and derivatives.

apply_custom_timepoints()
Apply custom timepoints, if applicable.

See the *set_custom_timepoints* method for more information.

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

call_unprocessed(*x*, *sensi_orders*, *mode*, *edatas=None*, *parameter_mapping=None*)
Call objective function without pre- or post-processing and formatting.

Parameters

- **x** – The parameters for which to evaluate the objective function.
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns A dict containing the results.

Return type result

check_mode(*mode*)
Check if the objective is able to compute in the requested mode.

Parameters **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether mode is supported

Return type flag

check_sensi_orders(*sensi_orders*, *mode*) → bool
Check if the objective is able to compute the requested sensitivities.

Parameters

- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether combination of sensi_orders and mode is supported

Return type flag

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

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

reset_steadystate_guesses()
Resets all steadystate guess data

set_custom_timepoints(*timepoints: Optional[Sequence[Sequence[Union[float, int]]]] = None*, *timepoints_global: Optional[Sequence[Union[float, int]]] = None*) → pypesto.objective.amici.AmiciObjective
Create a copy of this objective that will be evaluated at custom timepoints.

The intended use is to aid in predictions at unmeasured timepoints.

Parameters

- **timepoints** – The outer sequence should contain a sequence of timepoints for each experimental condition.
- **timepoints_global** – A sequence of timepoints that will be used for all experimental conditions.

Returns

Return type The customized copy of this objective.

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

class pypesto.objective.**CsvHistory**(*file: str*, *x_names: Optional[Sequence[str]] = None*, *options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*, *load_from_file: bool = False*)
Bases: pypesto.objective.history.History

Stores a representation of the history in a CSV file.

Parameters

- **file** – CSV file name.
- **x_names** – Parameter names.
- **options** – History options.
- **load_from_file** – If True, history will be initialized from data in the specified file

__init__(*file: str*, *x_names: Optional[Sequence[str]] = None*, *options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*, *load_from_file: bool = False*)
Initialize self. See help(type(self)) for accurate signature.

finalize()
Finalize history. Called after a run.

get_chi2_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Chi2 values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_fval_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Function values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_grad_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Gradients.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_hess_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Hessians.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_res_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Residuals.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_schi2_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Chi2 sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_sres_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Residual sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_time_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Cumulative execution times.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_x_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Parameters.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

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

Parameters

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

class pypesto.objective.**Hdf5History**(*id: str*, *file: str*, *options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*)
Bases: pypesto.objective.history.History

Stores a representation of the history in an HDF5 file.

Parameters

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

- **options** – History options.

__init__(*id: str*, *file: str*, *options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*)
Initialize self. See help(type(self)) for accurate signature.

finalize()
Finalize history. Called after a run.

get_chi2_trace() → Sequence[numpy.ndarray]
Chi2 values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_fval_trace() → Sequence[float]
Function values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_grad_trace() → Sequence[numpy.ndarray]
Gradients.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_hess_trace() → Sequence[numpy.ndarray]
Hessians.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_history_directory()

get_res_trace() → Sequence[numpy.ndarray]
Residuals.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

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

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_sres_trace() → Sequence[numpy.ndarray]
Residual sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_time_trace(*t: Optional[int] = None*) → Sequence[numpy.ndarray]
Cumulative execution times.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_x_trace() → Sequence[numpy.ndarray]
Parameters.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

static load(*id: str*, *file: str*)
Loads the History object from memory.

property n_fval
Number of function evaluations.

property n_grad
Number of gradient evaluations.

property n_hess
Number of Hessian evaluations.

property n_res
Number of residual evaluations.

property n_sres
Number or residual sensitivity evaluations.

property trace_save_iter

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

Parameters

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

class pypesto.objective.**History**(*options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*)
Bases: pypesto.objective.history.HistoryBase

Tracks numbers of function evaluations only, no trace.

Parameters **options** – History options.

__init__(*options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*)
Initialize self. See help(type(self)) for accurate signature.

finalize()
Finalize history. Called after a run.

property n_fval
Number of function evaluations.

property n_grad
Number of gradient evaluations.

property n_hess
Number of Hessian evaluations.

property n_res
Number of residual evaluations.

property n_sres
Number or residual sensitivity evaluations.

property start_time
Start time.

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

Parameters

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

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

Abstract base class for history objects.

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

finalize()
Finalize history. Called after a run.

get_chi2_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[float], float]
Chi2 values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_fval_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[float], float]
Function values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_grad_trace(*ix: Optional[Union[int, Sequence[int]]] = None*) → Union[Sequence[Union[numpy.ndarray, np.nan]], numpy.ndarray, np.nan]
Gradients.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_hess_trace(*ix: Optional[Union[int, Sequence[int]]] = None*) → Union[Sequence[Union[numpy.ndarray, np.nan]], numpy.ndarray, np.nan]
Hessians.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_res_trace(*ix: Optional[Union[int, Sequence[int]]] = None*) → Union[Sequence[Union[numpy.ndarray, np.nan]], numpy.ndarray, np.nan]
Residuals.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_schi2_trace(*ix: Optional[Union[int, Sequence[int]]] = None*) → Union[Sequence[Union[numpy.ndarray, np.nan]], numpy.ndarray, np.nan]
Chi2 sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_sres_trace(*ix: Optional[Union[int, Sequence[int]]] = None*) → Union[Sequence[Union[numpy.ndarray, np.nan]], numpy.ndarray, np.nan]
Residual sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_time_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[float], float]
Cumulative execution times.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_x_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[numpy.ndarray], numpy.ndarray]
Parameters.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

property n_fval
Number of function evaluations.

property n_grad
Number of gradient evaluations.

property n_hess
Number of Hessian evaluations.

property n_res
Number of residual evaluations.

property n_sres
Number or residual sensitivity evaluations.

property start_time
Start time.

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

Parameters

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

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

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

In addition implements a factory pattern to generate history objects.

Parameters

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

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

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

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

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

Parameters

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

class pypesto.objective.**MemoryHistory**(*options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*)
Bases: pypesto.objective.history.History

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

Parameters **options** – History options.

__init__(*options: Optional[Union[pypesto.objective.history.HistoryOptions, Dict]] = None*)
Initialize self. See help(type(self)) for accurate signature.

get_chi2_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Chi2 values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_fval_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Function values.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_grad_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Gradients.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_hess_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Hessians.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_res_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Residuals.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_schi2_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Chi2 sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_sres_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Residual sensitivities.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_time_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Cumulative execution times.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

get_x_trace(*ix: Optional[Union[Sequence[int], int]] = None*) → Union[Sequence[Union[float, numpy.ndarray, np.nan]], float, numpy.ndarray, np.nan]
Parameters.

Takes as parameter an index or indices and returns corresponding trace values. If only a single value is requested, the list is flattened.

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

Parameters

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

class pypesto.objective.**NegLogParameterPriors**(*prior_list: List[Dict]*, *x_names: Optional[Sequence[str]] = None*)

Bases: pypesto.objective.base.ObjectiveBase

This class implements Negative Log Priors on Parameters.

Contains a list of prior dictionaries for the individual parameters of the format

{'index': [int], 'density_fun': [Callable], 'density_dx': [Callable], 'density_ddx': [Callable]}

A prior instance can be added to e.g. an objective, that gives the likelihood, by an AggregatedObjective.

Notes

All callables should correspond to log-densities. That is, they return log-densities and their corresponding derivatives. Internally, values are multiplied by -1, since pyPESTO expects the Objective function to be of a negative log-density type.

__init__(*prior_list: List[Dict]*, *x_names: Optional[Sequence[str]] = None*)

Constructor

Parameters

- **prior_list** – List of dicts containing the individual parameter priors. Format see above.
- **x_names** – Sequence of parameter names (optional).

call_unprocessed(*x: numpy.ndarray*, *sensi_orders: Tuple[int, ...]*, *mode: str*) → Dict[str, Union[float, numpy.ndarray, Dict]]

Call objective function without pre- or post-processing and formatting.

Parameters

- **x** – The parameters for which to evaluate the objective function.
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns A dict containing the results.

Return type result

check_mode(*mode*) → bool

Check if the objective is able to compute in the requested mode.

Parameters **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether mode is supported

Return type flag

check_sensi_orders(*sensi_orders: Tuple[int, ...]*, *mode: str*) → bool

Check if the objective is able to compute the requested sensitivities.

Parameters

- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether combination of sensi_orders and mode is supported

Return type flag

gradient_neg_log_density(*x*)
Computes the gradient of the negative log-density for a parameter vector x.

hessian_neg_log_density(*x*)
Computes the hessian of the negative log-density for a parameter vector x.

hessian_vp_neg_log_density(*x*, *p*)
Computes the hessian vector product of the hessian of the negative log-density for a parameter vector x with a vector p.

neg_log_density(*x*)
Computes the negative log-density for a parameter vector x.

class pypesto.objective.**NegLogPriors**(*objectives: Sequence[pypesto.objective.base.ObjectiveBase]*, *x_names: Optional[Sequence[str]] = None*)

Bases: pypesto.objective.aggregated.AggregatedObjective

Aggregates different forms of negative log-prior distributions.

Allows to distinguish priors from the likelihood by testing the type of an objective.

Consists basically of a list of individual negative log-priors, given in self.objectives.

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

Bases: pypesto.objective.base.ObjectiveBase

The objective class allows the user explicitely specify functions that compute the function value and/or residuals as well as respective derivatives.

Parameters

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- **x_names** – Parameter names. None if no names provided, otherwise a list of str, length dim_full (as in the Problem class). Can be read by the problem.

__init__ (*fun: Optional[Callable] = None*, *grad: Optional[Union[Callable, bool]] = None*, *hess: Optional[Callable] = None*, *hessp: Optional[Callable] = None*, *res: Optional[Callable] = None*, *sres: Optional[Union[Callable, bool]] = None*, *x_names: Optional[Sequence[str]] = None*)

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

call_unprocessed (*x*, *sensi_orders*, *mode*)

Call objective function without pre- or post-processing and formatting.

Returns A dict containing the results.

Return type result

check_mode (*mode*)

Check if the objective is able to compute in the requested mode.

Parameters **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether mode is supported

Return type flag

check_sensi_orders (*sensi_orders*, *mode*)

Check if the objective is able to compute the requested sensitivities.

Parameters

- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether combination of sensi_orders and mode is supported

Return type flag

property has_fun

property has_grad

property has_hess

property has_hessp

property has_res

property has_sres

class pypesto.objective.**ObjectiveBase**(*x_names: Optional[Sequence[str]] = None*)
Bases: abc.ABC

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

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

Parameters **x_names** – Parameter names that can be optionally used in, e.g., history or gradient checks

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

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

__call__(*x: numpy.ndarray*, *sensi_orders: Tuple[int, ...] = (0)*, *mode: str = 'mode_fun'*, *return_dict: bool = False*, ***kwargs*) → Union[float, numpy.ndarray, Tuple, Dict[str, Union[float, numpy.ndarray, Dict]]]
Method to obtain arbitrary sensitivities. This is the central method which is always called, also by the get_* methods.

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

Parameters

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

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

Return type result

__init__(*x_names: Optional[Sequence[str]] = None*)
Initialize self. See help(type(self)) for accurate signature.

abstract call_unprocessed(*x: numpy.ndarray*, *sensi_orders: Tuple[int, ...]*, *mode: str*, ***kwargs*) → Dict[str, Union[float, numpy.ndarray, Dict]]
Call objective function without pre- or post-processing and formatting.

Parameters

- **x** – The parameters for which to evaluate the objective function.
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns A dict containing the results.

Return type result

check_grad(*x: numpy.ndarray*, *x_indices: Optional[Sequence[int]] = None*, *eps: float = 1e-05*, *verbosity: int = 1*, *mode: str = 'mode_fun'*, *detailed: bool = False*) → pandas.core.frame.DataFrame

Compare gradient evaluation: Firstly approximate via finite differences, and secondly use the objective gradient.

Parameters

- **x** – The parameters for which to evaluate the gradient.
- **x_indices** – Indices for which to compute gradients. Default: all.
- **eps** – Finite differences step size.
- **verbosity** – Level of verbosity for function output. 0: no output, 1: summary for all parameters, 2: summary for individual parameters.
- **mode** – Residual (MODE_RES) or objective function value (MODE_FUN) computation mode.
- **detailed** – Toggle whether additional values are returned. Additional values are function values, and the central difference weighted by the difference in output from all methods (standard deviation and mean).

Returns gradient, finite difference approximations and error estimates.

Return type result

check_grad_multi_eps(**args*, *multi_eps: Optional[Iterable] = None*, *label: str = 'rel_err'*, ***kwargs*)

Equivalent to the *ObjectiveBase.check_grad* method, except multiple finite difference step sizes are tested. The result contains the lowest finite difference for each parameter, and the corresponding finite difference step size.

Parameters

- **ObjectiveBase.check_grad method parameters.** (*All*) –
- **multi_eps** – The finite difference step sizes to be tested.
- **label** – The label of the column that will be minimized for each parameter. Valid options are the column labels of the dataframe returned by the *ObjectiveBase.check_grad* method.

abstract check_mode(*mode*) → bool

Check if the objective is able to compute in the requested mode.

Parameters **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether mode is supported

Return type flag

abstract check_sensi_orders(*sensi_orders*, *mode*) → bool

Check if the objective is able to compute the requested sensitivities.

Parameters

- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether combination of sensi_orders and mode is supported

Return type flag

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

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

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

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

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

property has_fun

property has_grad

property has_hess

property has_hessp

property has_res

property has_sres

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

static output_to_tuple(*sensi_orders: Tuple[int, ...]*, *mode: str*, ***kwargs: Union[float, numpy.ndarray]*) → Tuple
Return values as requested by the caller, since usually only a subset is demanded. One output is returned as-is, more than one output are returned as a tuple in order (fval, grad, hess).

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

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

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

Parameters

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

class pypesto.objective.**OptimizerHistory**(*history: pypesto.objective.history.History*, *x0: numpy.ndarray*, *generate_from_history: bool = False*)

Bases: object

Objective call history. Container around a History object, which keeps track of optimal values.

fval0, fval_min
Initial and best function value found.

chi20, chi2_min
Initial and best chi2 value found.

x0, x_min
Initial and best parameters found.

grad_min
gradient for best parameters

hess_min
hessian (approximation) for best parameters

res_min
residuals for best parameters

sres_min
residual sensitivities for best parameters

Parameters

- **history** – History object to attach to this container. This history object implements the storage of the actual history.
- **x0** – Initial values for optimization
- **generate_from_history** – If set to true, this function will try to fill attributes of this function based on the provided history

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

extract_from_history(*var*, *ix*)

finalize()

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

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

```
chi2 = sum(res**2)
```

which is consistent with the AMICI definition but NOT the 'Linear' formulation in scipy.

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

4.3 Problem

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

class pypesto.problem.**NegLogPriors**(*objectives: Sequence[pypesto.objective.base.ObjectiveBase]*, *x_names: Optional[Sequence[str]] = None*)

Bases: pypesto.objective.aggregated.AggregatedObjective

Aggregates different forms of negative log-prior distributions.

Allows to distinguish priors from the likelihood by testing the type of an objective.

Consists basically of a list of individual negative log-priors, given in self.objectives.

class pypesto.problem.**ObjectiveBase**(*x_names: Optional[Sequence[str]] = None*)

Bases: abc.ABC

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

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

Parameters **x_names** – Parameter names that can be optionally used in, e.g., history or gradient checks

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

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

__call__(*x: numpy.ndarray*, *sensi_orders: Tuple[int, ...] = (0)*, *mode: str = 'mode_fun'*, *return_dict: bool = False*, ***kwargs*) → Union[float, numpy.ndarray, Tuple, Dict[str, Union[float, numpy.ndarray, Dict]]]

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

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

Parameters

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

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

Return type result

__init__(*x_names: Optional[Sequence[str]] = None*)
Initialize self. See help(type(self)) for accurate signature.

abstract call_unprocessed(*x: numpy.ndarray*, *sensi_orders: Tuple[int, ...]*, *mode: str*, ***kwargs*) → Dict[str, Union[float, numpy.ndarray, Dict]]
Call objective function without pre- or post-processing and formatting.

Parameters

- **x** – The parameters for which to evaluate the objective function.
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns A dict containing the results.

Return type result

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

Parameters

- **x** – The parameters for which to evaluate the gradient.
- **x_indices** – Indices for which to compute gradients. Default: all.
- **eps** – Finite differences step size.
- **verbosity** – Level of verbosity for function output. 0: no output, 1: summary for all parameters, 2: summary for individual parameters.
- **mode** – Residual (MODE_RES) or objective function value (MODE_FUN) computation mode.
- **detailed** – Toggle whether additional values are returned. Additional values are function values, and the central difference weighted by the difference in output from all methods (standard deviation and mean).

Returns gradient, finite difference approximations and error estimates.

Return type result

check_grad_multi_eps(**args*, *multi_eps: Optional[Iterable] = None*, *label: str = 'rel_err'*, ***kwargs*)
Equivalent to the *ObjectiveBase.check_grad* method, except multiple finite difference step sizes are tested. The result contains the lowest finite difference for each parameter, and the corresponding finite difference step size.

Parameters

- **ObjectiveBase.check_grad method parameters.** (*All*) –
- **multi_eps** – The finite difference step sizes to be tested.
- **label** – The label of the column that will be minimized for each parameter. Valid options are the column labels of the dataframe returned by the *ObjectiveBase.check_grad* method.

abstract check_mode(*mode*) → bool
Check if the objective is able to compute in the requested mode.

Parameters **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether mode is supported

Return type flag

abstract check_sensi_orders(*sensi_orders*, *mode*) → bool
Check if the objective is able to compute the requested sensitivities.

Parameters

- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.

Returns Boolean indicating whether combination of sensi_orders and mode is supported

Return type flag

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

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

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

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

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

property has_fun

property has_grad

property has_hess

property has_hessp

property has_res

property has_sres

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

static output_to_tuple(*sensi_orders: Tuple[int, ...]*, *mode: str*, ***kwargs: Union[float, numpy.ndarray]*) → Tuple
Return values as requested by the caller, since usually only a subset is demanded. One output is returned as-is, more than one output are returned as a tuple in order (fval, grad, hess).

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

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

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

Parameters

- **dim_full** – Dimension of the full vector including fixed parameters.

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

class pypesto.problem.**Problem**(*objective: pypesto.objective.base.ObjectiveBase, lb: Union[numpy.ndarray, List[float]], ub: Union[numpy.ndarray, List[float]], dim_full: Optional[int] = None, x_fixed_indices: Optional[Union[Iterable[SupportsInt], SupportsInt]] = None, x_fixed_vals: Optional[Union[Iterable[SupportsFloat], SupportsFloat]] = None, x_guesses: Optional[Iterable[float]] = None, startpoint_method: Optional[Callable] = None, x_names: Optional[Iterable[str]] = None, x_scales: Optional[Iterable[str]] = None, x_priors_defs: Optional[pypesto.objective.priors.NegLogPriors] = None, lb_init: Optional[Union[numpy.ndarray, List[float]]] = None, ub_init: Optional[Union[numpy.ndarray, List[float]]] = 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.
- **lb** – The lower and upper bounds for optimization. For unbounded directions set to +-inf.
- **ub** – The lower and upper bounds for optimization. For unbounded directions set to +-inf.
- **lb_init** – The lower and upper bounds for initialization, typically for defining search start points. If not set, set to lb, ub.
- **ub_init** – The lower and upper bounds for initialization, typically for defining search start points. If not set, set to lb, ub.
- **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.
- **startpoint_method** – Callable. *startpoint_method(n_starts)* returns a n_starts x n_free_indices array of initial values for 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.
- **x_scales** – Parameter scales can be optionally given and are used e.g. in visualisation and prior generation. Currently the scales 'lin', 'log`and 'log10' are supported.

- **x_priors_defs** – Definitions of priors for parameters. Types of priors, and their required and optional parameters, are described in the *Prior* class.

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.

__init__(*objective: pypesto.objective.base.ObjectiveBase*, *lb: Union[numpy.ndarray, List[float]]*, *ub: Union[numpy.ndarray, List[float]]*, *dim_full: Optional[int] = None*, *x_fixed_indices: Optional[Union[Iterable[SupportsInt], SupportsInt]] = None*, *x_fixed_vals: Optional[Union[Iterable[SupportsFloat], SupportsFloat]] = None*, *x_guesses: Optional[Iterable[float]] = None*, *startpoint_method: Optional[Callable] = None*, *x_names: Optional[Iterable[str]] = None*, *x_scales: Optional[Iterable[str]] = None*, *x_priors_defs: Optional[pypesto.objective.priors.NegLogPriors] = None*, *lb_init: Optional[Union[numpy.ndarray, List[float]]] = None*, *ub_init: Optional[Union[numpy.ndarray, List[float]]] = None*)
Initialize self. See help(type(self)) for accurate signature.

property dim

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

full_index_to_free_index(*full_index: int*)
Calculate index in reduced vector from index in full vector.

Parameters **full_index** (*The index in the full vector.*) –

Returns free_index

Return type The index in the free vector.

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: Optional[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]*, *x_indices: Optional[List[int]] = None*) → Optional[numpy.ndarray]

Keep only those elements, which indices are specified in x_indices If x_indices is not provided, delete fixed indices.

Parameters

- **x_full** (*array_like, ndim=1*) – The vector in dimension dim_full.
- **x_indices** – indices of x_full that should remain

property lb

property lb_init

normalize() → 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 parameter boundaries.

property ub

property ub_init

unfix_parameters(*parameter_indices: Union[Iterable[SupportsInt], SupportsInt]*) → None

Free specified parameters

property x_free_indices

property x_guesses

class pypesto.problem.**SupportsFloat**(**args*, ***kwargs*)

Bases: Protocol

An ABC with one abstract method __float__.

__init__(**args*, ***kwargs*)

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

class pypesto.problem.**SupportsInt**(**args*, ***kwargs*)

Bases: Protocol

An ABC with one abstract method __int__.

__init__(**args*, ***kwargs*)

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

4.4 Prediction

Generate predictions from simulations with specified parameter vectors, with optional post-processing.

class pypesto.predict.**AmiciPredictor**(*amici_objective: pypesto.objective.amici.AmiciObjective*, *amici_output_fields: Optional[Sequence[str]] = None*, *post_processor: Optional[Callable] = None*, *post_processor_sensi: Optional[Callable] = None*, *post_processor_time: Optional[Callable] = None*, *max_chunk_size: Optional[int] = None*, *output_ids: Optional[Sequence[str]] = None*, *condition_ids: Optional[Sequence[str]] = None*)

Bases: object

Do forward simulations (predictions) with parameter vectors, for an AMICI model. The user may supply post-processing methods. If post-processing methods are supplied, and a gradient of the prediction is requested, then the sensitivities of the AMICI model must also be post-processed. There are no checks here to ensure that the sensitivities are correctly post-processed, this is explicitly left to the user. There are also no safeguards if the postprocessor routines fail. This may happen if, e.g., a call to Amici fails, and no timepoints, states or observables are returned. As the AmiciPredictor is agnostic about the dimension of the postprocessor and also the dimension of the postprocessed output, these checks are also left to the user. An example for such a check is provided in the default output (see _default_output()).

__call__(*x: numpy.ndarray*, *sensi_orders: Tuple[int, ...] = (0)*, *mode: str = 'mode_fun'*, *output_file: str = ''*, *output_format: str = 'csv'*) → pypesto.predict.result.PredictionResult

Simulate a model for a certain prediction function. This method relies on the AmiciObjective, which is underlying, but allows the user to apply any post-processing of the results, the sensitivities, and the timepoints.

Parameters

- **x** – The parameters for which to evaluate the prediction function.
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.
- **mode** – Whether to compute function values or residuals.
- **output_file** – Path to an output file.
- **output_format** – Either 'csv', 'h5'. If an output file is specified, this routine will return a csv file, created from a DataFrame, or an h5 file, created from a dict.

Returns PredictionResult object containing timepoints, outputs, and output_sensitivities if requested

Return type results

__init__(*amici_objective: pypesto.objective.amici.AmiciObjective*, *amici_output_fields: Optional[Sequence[str]] = None*, *post_processor: Optional[Callable] = None*, *post_processor_sensi: Optional[Callable] = None*, *post_processor_time: Optional[Callable] = None*, *max_chunk_size: Optional[int] = None*, *output_ids: Optional[Sequence[str]] = None*, *condition_ids: Optional[Sequence[str]] = None*)

Constructor.

Parameters

- **amici_objective** – An objective object, which will be used to get model simulations
- **amici_output_fields** – keys that exist in the return data object from AMICI, which should be available for the post-processors
- **post_processor** – A callable function which applies postprocessing to the simulation results and possibly defines different outputs than those of the amici model. Default are the observables (*pypesto.predict.constants.AMICI_Y*) of the AMICI model. This method takes a list of dicts (with the returned fields *pypesto.predict.constants.AMICI_T*, *pypesto.predict.constants.AMICI_X*, and *pypesto.predict.constants.AMICI_Y* of the AMICI ReturnData objects) as input. Safeguards for, e.g., failure of AMICI are left to the user.
- **post_processor_sensi** – A callable function which applies postprocessing to the sensitivities of the simulation results. Defaults to the observable sensitivities of the AMICI model. This method takes a list of dicts (with the returned fields *pypesto.predict.constants.AMICI_T*, *pypesto.predict.constants.AMICI_X*, *pypesto.predict.constants.AMICI_Y*, *pypesto.predict.constants.AMICI_SX*, and *pypesto.predict.constants.AMICI_SY* of the AMICI ReturnData objects) as input. Safeguards for, e.g., failure of AMICI are left to the user.

- **post_processor_time** – A callable function which applies postprocessing to the timepoints of the simulations. Defaults to the timepoints of the amici model. This method takes a list of dicts (with the returned field *pypesto.predict.constants.AMICI_T* of the amici ReturnData objects) as input. Safeguards for, e.g., failure of AMICI are left to the user.
- **max_chunk_size** – In some cases, we don't want to compute all predictions at once when calling the prediction function, as this might not fit into the memory for large datasets and models. Here, the user can specify a maximum chunk size of conditions, which should be simulated at a time. Defaults to None, meaning that all conditions will be simulated.
- **output_ids** – IDs of outputs, as post-processing allows the creation of customizable outputs, which may not coincide with those from the AMICI model (defaults to AMICI observables).
- **condition_ids** – List of identifiers for the conditions of the edata objects of the amici objective, will be passed to the PredictionResult at call.

class pypesto.predict.**PredictionConditionResult**(*timepoints: numpy.ndarray*, *output_ids: Sequence[str]*, *output: Optional[numpy.ndarray] = None*, *output_sensi: Optional[numpy.ndarray] = None*, *x_names: Optional[Sequence[str]] = None*)

Bases: object

This class is a light-weight wrapper for the prediction of one simulation condition of an amici model. It should provide a common api how amici predictions should look like in pyPESTO.

__init__(*timepoints: numpy.ndarray*, *output_ids: Sequence[str]*, *output: Optional[numpy.ndarray] = None*, *output_sensi: Optional[numpy.ndarray] = None*, *x_names: Optional[Sequence[str]] = None*)

Constructor.

Parameters

- **timepoints** – Output timepoints for this simulation condition
- **output_ids** – IDs of outputs for this simulation condition
- **outputs** – Postprocessed outputs (ndarray)
- **outputs_sensi** – Sensitivities of postprocessed outputs (ndarray)
- **x_names** – IDs of model parameter w.r.t to which sensitivities were computed

class pypesto.predict.**PredictionResult**(*conditions: Sequence[Union[pypesto.predict.result.PredictionConditionResu Dict]]*, *condition_ids: Optional[Sequence[str]] = None*, *comment: Optional[str] = None*)

Bases: object

This class is a light-weight wrapper around predictions from pyPESTO made via an amici model. It's only purpose is to have fixed format/api, how prediction results should be stored, read, and handled: as predictions are a very flexible format anyway, they should at least have a common definition, which allows to work with them in a reasonable way.

__init__(*conditions: Sequence[Union[pypesto.predict.result.PredictionConditionResult, Dict]]*, *condition_ids: Optional[Sequence[str]] = None*, *comment: Optional[str] = None*)

Constructor.

Parameters

- **conditions** – A list of PredictionConditionResult objects or dicts

- **condition_ids** – IDs or names of the simulation conditions, which belong to this prediction (e.g., PEtab uses tuples of preequilibration condition and simulation conditions)
- **comment** – An additional note, which can be attached to this prediction

write_to_csv(*output_file: str*)
This method saves predictions to a csv file.

Parameters **output_file** – path to file/folder to which results will be written

write_to_h5(*output_file: str*, *base_path: Optional[str] = None*)
This method saves predictions to an h5 file. It appends to the file if the file already exists.

Parameters

- **output_file** – path to file/folder to which results will be written
- **base_path** – base path in the h5 file

class pypesto.predict.**PredictorTask**(*predictor: pypesto.predict.Predictor*, *x: Sequence[float]*, *sensi_orders: Tuple[int, ...]*, *mode: str*, *id: str*)
Bases: pypesto.engine.task.Task

Perform a single prediction with *pypesto.engine.Task*.

Designed for use with *pypesto.ensemble.Ensemble*.

predictor
The predictor to use.

x
The parameter vector to compute predictions with.

sensi_orders
Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad.

mode
Whether to compute function values or residuals.

id
The input ID.

__init__(*predictor: pypesto.predict.Predictor*, *x: Sequence[float]*, *sensi_orders: Tuple[int, ...]*, *mode: str*, *id: str*)
Initialize self. See help(type(self)) for accurate signature.

execute() → *pypesto.predict.PredictionResult*
Execute the task and return its results.

4.5 PEtab

pyPESTO support for the PEtab data format.

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

MODEL_BASE_DIR = 'amici_models'

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

petab_problem: Managing access to the model and data.

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

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

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

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

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

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

Parameters

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

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

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

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

Parameters

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

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

Return type objective

create_predictor(*objective: Optional[pypesto.objective.amici.AmiciObjective] = None*, *amici_output_fields: Optional[Sequence[str]] = None*, *post_processor: Optional[Callable] = None*, *post_processor_sensi: Optional[Callable] = None*, *post_processor_time: Optional[Callable] = None*, *max_chunk_size: Optional[int] = None*, *output_ids: Optional[Sequence[str]] = None*, *condition_ids: Optional[Sequence[str]] = None*) → pypesto.predict.amici_predictor.AmiciPredictor
Create a *pypesto.predict.AmiciPredictor*.

The *AmiciPredictor* facilitates generation of predictions from parameter vectors.

Parameters

- **objective** – An objective object, which will be used to get model simulations
- **amici_output_fields** – keys that exist in the return data object from AMICI, which should be available for the post-processors

- **post_processor** – A callable function which applies postprocessing to the simulation results. Default are the observables of the AMICI model. This method takes a list of ndarrays (as returned in the field ['y'] of amici ReturnData objects) as input.
- **post_processor_sensi** – A callable function which applies postprocessing to the sensitivities of the simulation results. Default are the observable sensitivities of the AMICI model. This method takes two lists of ndarrays (as returned in the fields ['y'] and ['sy'] of amici ReturnData objects) as input.
- **post_processor_time** – A callable function which applies postprocessing to the timepoints of the simulations. Default are the timepoints of the amici model. This method takes a list of ndarrays (as returned in the field ['t'] of amici ReturnData objects) as input.
- **max_chunk_size** – In some cases, we don't want to compute all predictions at once when calling the prediction function, as this might not fit into the memory for large datasets and models. Here, the user can specify a maximum number of conditions, which should be simulated at a time. Default is 0 meaning that all conditions will be simulated. Other values are only applicable, if an output file is specified.
- **output_ids** – IDs of outputs, if post-processing is used
- **condition_ids** – IDs of conditions, if post-processing is used

Returns A *pypesto.predict.AmiciPredictor* for the model, using the outputs of the AMICI model and the timepoints from the PEtab data

Return type predictor

create_prior() → pypesto.objective.priors.NegLogParameterPriors
Creates a prior from the parameter table. Returns None, if no priors are defined.

create_problem(*objective: Optional[pypesto.objective.amici.AmiciObjective] = None*, *x_guesses: Optional[Iterable[float]] = None*, ***kwargs*) → *pypesto.problem.Problem*
Create a *pypesto.Problem*.

Parameters

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

Returns A *pypesto.Problem* for the objective.

Return type problem

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

create_startpoint_method()
Creates a startpoint method, if the PEtab problem specifies an initializationPrior. Returns None, if no initializationPrior is specified.

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

prediction_to_petab_measurement_df(*prediction: pypesto.predict.result.PredictionResult*, *predictor: Optional[pypesto.predict.amici_predictor.AmiciPredictor] = None*) → pandas.core.frame.DataFrame
If a PEtab problem is simulated without post-processing, then the result can be cast into a PEtab measurement or simulation dataframe

Parameters

- **prediction** – A prediction result as produced by an AmiciPredictor
- **predictor** – The AmiciPredictor function

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

Return type measurement_df

prediction_to_petab_simulation_df(*prediction: pypesto.predict.result.PredictionResult*, *predictor: Optional[pypesto.predict.amici_predictor.AmiciPredictor] = None*) → pandas.core.frame.DataFrame

Same as *prediction_to_petab_measurement_df*, except a PEtab simulation dataframe is created, i.e. the measurement column label is adjusted.

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

Create a measurement dataframe in the petab format from the passed *rdatas* and own information.

Parameters

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

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

Return type measurement_df

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

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

class pypesto.petab.**PetabImporterPysb**(*petab_problem: amici.petab_import_pysb.PysbPetabProblem*, *output_folder: str = None*)

Bases: pypesto.petab.importer.PetabImporter

Import for experimental PySB-based PEtab problems

__init__(*petab_problem: amici.petab_import_pysb.PysbPetabProblem*, *output_folder: str = None*)

petab_problem: Managing access to the model and data.

output_folder: Folder to contain the amici model.

compile_model(***kwargs*)

Compile the model. If the output folder exists already, it is first deleted.

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

4.6 Optimize

Multistart optimization with support for various optimizers.

class pypesto.optimize.**CmaesOptimizer**(*par_sigma0: float = 0.25*, *options: Optional[Dict] = None*)

Bases: pypesto.optimize.optimizer.Optimizer

Global optimization using cma-es. Package homepage: https://pypi.org/project/cma-es/

__init__(*par_sigma0: float = 0.25*, *options: Optional[Dict] = None*)

Parameters

- **par_sigma0** – scalar, initial standard deviation in each coordinate. par_sigma0 should be about 1/4th of the search domain width (where the optimum is to be expected)
- **options** – Optimizer options that are directly passed on to cma.

is_least_squares()

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

class pypesto.optimize.**DlibOptimizer**(*options: Optional[Dict] = None*)

Bases: pypesto.optimize.optimizer.Optimizer

Use the Dlib toolbox for optimization.

__init__(*options: Optional[Dict] = None*)
Default constructor.

get_default_options()
Create default options specific for the optimizer.

is_least_squares()

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

class pypesto.optimize.**FidesOptimizer**(*hessian_update: Optional[fides.HessianApproximation] = 'Hybrid'*, *options: Optional[Dict] = None*, *verbose: Optional[int] = 20*)

Bases: pypesto.optimize.optimizer.Optimizer

Global/Local optimization using the trust region optimizer fides. Package Homepage: https://fides-optimizer.readthedocs.io/en/latest

__init__(*hessian_update: Optional[fides.HessianApproximation] = 'Hybrid'*, *options: Optional[Dict] = None*, *verbose: Optional[int] = 20*)

Parameters

- **options** – Optimizer options.
- **hessian_update** – Hessian update strategy. If this is None, Hessian (approximation) computed by problem.objective will be used.

is_least_squares()

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

class pypesto.optimize.**IpoptOptimizer**(*options: Optional[Dict] = None*)

Bases: pypesto.optimize.optimizer.Optimizer

Use IpOpt (https://pypi.org/project/ipopt/) for optimization.

__init__(*options: Optional[Dict] = None*)

Parameters **options** – Options are directly passed on to *cyipopt.minimize_ipopt*.

is_least_squares()

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

class pypesto.optimize.**NLoptOptimizer**(*method=None*, *local_method=None*, *options: Optional[Dict] = None*, *local_options: Optional[Dict] = None*)

Bases: pypesto.optimize.optimizer.Optimizer

Global/Local optimization using NLopt. Package homepage: https://nlopt.readthedocs.io/en/latest/

__init__(*method=None*, *local_method=None*, *options: Optional[Dict] = None*, *local_options: Optional[Dict] = None*)

Parameters

- **method** – Local or global Optimizer to use for minimization.
- **local_method** – Local method to use in combination with the global optimizer (for the MLSL family of solvers) or to solve a subproblem (for the AUGLAG family of solvers)
- **options** – Optimizer options. scipy option *maxiter* is automatically transformed into *maxeval* and takes precedence.
- **local_options** – Optimizer options for the local method

is_least_squares()

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

class pypesto.optimize.**OptimizeOptions**(*startpoint_resample: bool = False*, *allow_failed_starts: bool = True*)

Bases: dict

Options for the multistart optimization.

Parameters

- **startpoint_resample** – Flag indicating whether initial points are supposed to be resampled if function evaluation fails at the initial point
- **allow_failed_starts** – Flag indicating whether we tolerate that exceptions are thrown during the minimization process.

__init__(*startpoint_resample: bool = False*, *allow_failed_starts: bool = True*)
Initialize self. See help(type(self)) for accurate signature.

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

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

class pypesto.optimize.**Optimizer**

Bases: abc.ABC

This is the optimizer base class, not functional on its own. An optimizer takes a problem, and possibly a start point, and then performs an optimization. It returns an OptimizerResult.

__init__()
Default constructor.

get_default_options()
Create default options specific for the optimizer.

abstract is_least_squares()

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

class pypesto.optimize.**OptimizerResult**(*id: Optional[str] = None*, *x: Optional[numpy.ndarray] = None*, *fval: Optional[float] = None*, *grad: Optional[numpy.ndarray] = None*, *hess: Optional[numpy.ndarray] = None*, *res: Optional[numpy.ndarray] = None*, *sres: Optional[numpy.ndarray] = None*, *n_fval: Optional[int] = None*, *n_grad: Optional[int] = None*, *n_hess: Optional[int] = None*, *n_res: Optional[int] = None*, *n_sres: Optional[int] = None*, *x0: Optional[numpy.ndarray] = None*, *fval0: Optional[float] = None*, *history: Optional[pypesto.objective.history.History] = None*, *exitflag: Optional[int] = None*, *time: Optional[float] = None*, *message: Optional[str] = None*)

Bases: dict

The result of an optimizer run. Used as a standardized return value to map from the individual result objects returned by the employed optimizers to the format understood by pypesto.

Can be used like a dict.

id
Id of the optimizer run. Usually the start index.

x
The best found parameters.

fval
The best found function value, *fun(x)*.

grad
The gradient at *x*.

hess
The Hessian at *x*.

res
The residuals at *x*.

sres
The residual sensitivities at *x*.

n_fval
Number of function evaluations.

n_grad
Number of gradient evaluations.

n_hess
Number of Hessian evaluations.

n_res
Number of residuals evaluations.

n_sres
Number of residual sensitivity evaluations.

x0
The starting parameters.

fval0
The starting function value, *fun(x0)*.

history
Objective history.

exitflag
The exitflag of the optimizer.

time
Execution time.

message
Textual comment on the optimization result.

Type str

Notes

Any field not supported by the optimizer is filled with None.

__init__(*id: Optional[str] = None*, *x: Optional[numpy.ndarray] = None*, *fval: Optional[float] = None*, *grad: Optional[numpy.ndarray] = None*, *hess: Optional[numpy.ndarray] = None*, *res: Optional[numpy.ndarray] = None*, *sres: Optional[numpy.ndarray] = None*, *n_fval: Optional[int] = None*, *n_grad: Optional[int] = None*, *n_hess: Optional[int] = None*, *n_res: Optional[int] = None*, *n_sres: Optional[int] = None*, *x0: Optional[numpy.ndarray] = None*, *fval0: Optional[float] = None*, *history: Optional[pypesto.objective.history.History] = None*, *exitflag: Optional[int] = None*, *time: Optional[float] = None*, *message: Optional[str] = None*)
Initialize self. See help(type(self)) for accurate signature.

update_to_full(*problem:* pypesto.problem.Problem) → None
Updates values to full vectors/matrices

Parameters **problem** – problem which contains info about how to convert to full vectors or matrices

class pypesto.optimize.**PyswarmOptimizer**(*options: Optional[Dict] = None*)
Bases: pypesto.optimize.optimizer.Optimizer

Global optimization using pyswarm.

__init__(*options: Optional[Dict] = None*)
Default constructor.

is_least_squares()

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

class pypesto.optimize.**PyswarmsOptimizer**(*par_popsize: float = 10*, *options: Optional[Dict] = None*)
Bases: pypesto.optimize.optimizer.Optimizer

Global optimization using pyswarms. Package homepage: https://pyswarms.readthedocs.io/en/latest/index.html

Parameters

- **par_popsize** – number of particles in the swarm, default value 10

- **options** – Optimizer options that are directly passed on to pyswarms. c1: cognitive parameter c2: social parameter w: inertia parameter Default values are (c1,c2,w) = (0.5, 0.3, 0.9)

Examples

Arguments that can be passed to options:

maxiter: used to calculate the maximal number of funcion evaluations. Default: 1000

__init__(*par_popsize: float = 10*, *options: Optional[Dict] = None*)
Default constructor.

is_least_squares()

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

class pypesto.optimize.**ScipyDifferentialEvolutionOptimizer**(*options: Optional[Dict] = None*)
Bases: pypesto.optimize.optimizer.Optimizer

Global optimization using scipy's differential evolution optimizer. Package homepage: https://docs.scipy.org/doc/scipy/reference/generated /scipy.optimize.differential_evolution.html

Parameters **options** – Optimizer options that are directly passed on to scipy's optimizer.

Examples

Arguments that can be passed to options:

maxiter: used to calculate the maximal number of funcion evaluations by maxfevals = (maxiter + 1) * popsize * len(x) Default: 100

popsize: population size, default value 15

__init__(*options: Optional[Dict] = None*)
Default constructor.

is_least_squares()

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

class pypesto.optimize.**ScipyOptimizer**(*method: str = 'L-BFGS-B'*, *tol: float = 1e-09*, *options: Optional[Dict] = None*)
Bases: pypesto.optimize.optimizer.Optimizer

Use the SciPy optimizers. Find details on the optimizer and configuration options at: https://docs.scipy.org/doc/scipy/reference/generated/scipy. optimize.minimize.html#scipy.optimize.minimize

__init__(*method: str = 'L-BFGS-B'*, *tol: float = 1e-09*, *options: Optional[Dict] = None*)
Default constructor.

get_default_options()
Create default options specific for the optimizer.

is_least_squares()

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

pypesto.optimize.**minimize**(*problem: pypesto.problem.Problem, optimizer: Optional[pypesto.optimize.optimizer.Optimizer] = None, n_starts: int = 100, ids: Optional[Iterable[str]] = None, startpoint_method: Optional[Union[Callable, bool]] = None, result: Optional[pypesto.result.Result] = None, engine: Optional[pypesto.engine.base.Engine] = None, progress_bar: bool = True, options: Optional[pypesto.optimize.options.OptimizeOptions] = None, history_options: Optional[pypesto.objective.history.HistoryOptions] = None*) → *pypesto.result.Result*

This is the main function to call to do multistart optimization.

Parameters

- **problem** – The problem to be solved.
- **optimizer** – The optimizer to be used n_starts times.
- **n_starts** – Number of starts of the optimizer.
- **ids** – Ids assigned to the startpoints.
- **startpoint_method** – Method for how to choose start points. False means the optimizer does not require start points, e.g. for the 'PyswarmOptimizer'.
- **result** – A result object to append the optimization results to. For example, one might append more runs to a previous optimization. If None, a new object is created.
- **engine** – Parallelization engine. Defaults to sequential execution on a SingleCoreEngine.
- **progress_bar** – Whether to display a progress bar.
- **options** – Various options applied to the multistart optimization.
- **history_options** – Optimizer history options.

Returns Result object containing the results of all multistarts in *result.optimize_result*.

Return type result

4.7 Profile

class pypesto.profile.**ProfileOptions**(*default_step_size: float = 0.01, min_step_size: float = 0.001, max_step_size: float = 1.0, step_size_factor: float = 1.25, delta_ratio_max: float = 0.1, ratio_min: float = 0.145, reg_points: int = 10, reg_order: int = 4, magic_factor_obj_value: float = 0.5*)

Bases: dict

Options for optimization based profiling.

Parameters

- **default_step_size** – Default step size of the profiling routine along the profile path (adaptive step lengths algorithms will only use this as a first guess and then refine the update).
- **min_step_size** – Lower bound for the step size in adaptive methods.
- **max_step_size** – Upper bound for the step size in adaptive methods.

- **step_size_factor** – Adaptive methods recompute the likelihood at the predicted point and try to find a good step length by a sort of line search algorithm. This factor controls step handling in this line search.
- **delta_ratio_max** – Maximum allowed drop of the posterior ratio between two profile steps.
- **ratio_min** – Lower bound for likelihood ratio of the profile, based on inverse chi2-distribution. The default 0.145 is slightly lower than the 95% quantile 0.1465 of a chi2 distribution with one degree of freedom.
- **reg_points** – Number of profile points used for regression in regression based adaptive profile points proposal.
- **reg_order** – Maximum degree of regression polynomial used in regression based adaptive profile points proposal.
- **magic_factor_obj_value** – There is this magic factor in the old profiling code which slows down profiling at small ratios (must be >= 0 and < 1).

__init__(*default_step_size: float = 0.01*, *min_step_size: float = 0.001*, *max_step_size: float = 1.0*, *step_size_factor: float = 1.25*, *delta_ratio_max: float = 0.1*, *ratio_min: float = 0.145*, *reg_points: int = 10*, *reg_order: int = 4*, *magic_factor_obj_value: float = 0.5*)

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

static create_instance(*maybe_options: Union[pypesto.profile.options.ProfileOptions, Dict]*) → pypesto.profile.options.ProfileOptions

Returns a valid options object.

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

class pypesto.profile.**ProfilerResult**(*x_path: numpy.ndarray*, *fval_path: numpy.ndarray*, *ratio_path: numpy.ndarray*, *gradnorm_path: Optional[numpy.ndarray] = None*, *exitflag_path: Optional[numpy.ndarray] = None*, *time_path: Optional[numpy.ndarray] = None*, *time_total: float = 0.0*, *n_fval: int = 0*, *n_grad: int = 0*, *n_hess: int = 0*, *message: Optional[str] = None*)

Bases: dict

The result of a profiler run. The standardized return return value from pypesto.profile, which can either be initialized from an OptimizerResult or from an existing ProfilerResult (in order to extend the computation).

Can be used like a dict.

x_path
The path of the best found parameters along the profile (Dimension: n_par x n_profile_points)

fval_path
The function values, fun(x), along the profile.

ratio_path
The ratio of the posterior function along the profile.

gradnorm_path
The gradient norm along the profile.

exitflag_path
The exitflags of the optimizer along the profile.

time_path
The computation time of the optimizer runs along the profile.

time_total
The total computation time for the profile.

n_fval
Number of function evaluations.

n_grad
Number of gradient evaluations.

n_hess
Number of Hessian evaluations.

message
Textual comment on the profile result.

Notes

Any field not supported by the profiler or the profiling optimizer is filled with None. Some fields are filled by pypesto itself.

__init__(*x_path: numpy.ndarray*, *fval_path: numpy.ndarray*, *ratio_path: numpy.ndarray*, *gradnorm_path: Optional[numpy.ndarray] = None*, *exitflag_path: Optional[numpy.ndarray] = None*, *time_path: Optional[numpy.ndarray] = None*, *time_total: float = 0.0*, *n_fval: int = 0*, *n_grad: int = 0*, *n_hess: int = 0*, *message: Optional[str] = None*)
Initialize self. See help(type(self)) for accurate signature.

append_profile_point(*x: numpy.ndarray*, *fval: float*, *ratio: float*, *gradnorm: float = nan*, *time: float = nan*, *exitflag: float = nan*, *n_fval: int = 0*, *n_grad: int = 0*, *n_hess: int = 0*) → None
This function appends a new point to the profile path.

Parameters

- **x** – The parameter values.
- **fval** – The function value at *x*.
- **ratio** – The ratio of the function value at *x* by the optimal function value.
- **gradnorm** – The gradient norm at *x*.
- **time** – The computation time to find *x*.
- **exitflag** – The exitflag of the optimizer (useful if an optimization was performed to find *x*).
- **n_fval** – Number of function evaluations performed to find *x*.
- **n_grad** – Number of gradient evaluations performed to find *x*.
- **n_hess** – Number of Hessian evaluations performed to find *x*.

flip_profile() → None
This function flips the profiling direction (left-right) Profiling direction needs to be changed once (if the profile is new), or twice if we append to an existing profile.

All profiling paths are flipped in-place.

pypesto.profile.**approximate_parameter_profile**(*problem:* pypesto.problem.Problem, *result:* pypesto.result.Result, *profile_index: Optional[Iterable[int]] = None*, *profile_list: Optional[int] = None*, *result_index: int = 0*, *n_steps: int = 100*) → *pypesto.result.Result*

Calculate profiles based on an approximation via a normal likelihood centered at the chosen optimal parameter value, with the covariance matrix being the Hessian or FIM.

Parameters

- **problem** – The problem to be solved.
- **result** – A result object to initialize profiling and to append the profiling results to. For example, one might append more profiling runs to a previous profile, in order to merge these. The existence of an optimization result is obligatory.
- **profile_index** – List with the profile indices to be computed (by default all of the free parameters).
- **profile_list** – Integer which specifies whether a call to the profiler should create a new list of profiles (default) or should be added to a specific profile list.
- **result_index** – Index from which optimization result profiling should be started (default: global optimum, i.e., index = 0).
- **n_steps** – Number of profile steps in each dimension.

Returns The profile results are filled into *result.profile_result*.

Return type result

pypesto.profile.**calculate_approximate_ci**(*xs: numpy.ndarray*, *ratios: numpy.ndarray*, *confidence_ratio: float*) → Tuple[float, float]

Calculate approximate confidence interval based on profile. Interval bounds are linerly interpolated.

Parameters

- **xs** – The ordered parameter values along the profile for the coordinate of interest.
- **ratios** – The likelihood ratios corresponding to the parameter values.
- **confidence_ratio** – Minimum confidence ratio to base the confidence interval upon, as obtained via *pypesto.profile.chi2_quantile_to_ratio*.

Returns Bounds of the approximate confidence interval.

Return type lb, ub

pypesto.profile.**chi2_quantile_to_ratio**(*alpha: float = 0.95*, *df: int = 1*)

Transform lower tail probability *alpha* for a chi2 distribution with *df* degrees of freedom to a profile likelihood ratio threshold.

Parameters

- **alpha** – Lower tail probability, defaults to 95% interval.
- **df** – Degrees of freedom. Defaults to 1.

Returns Corresponds to a likelihood ratio.

Return type ratio

pypesto.profile.**parameter_profile**(*problem: pypesto.problem.Problem, result: pypesto.result.Result, optimizer: pypesto.optimize.optimizer.Optimizer, engine: Optional[pypesto.engine.base.Engine] = None, profile_index: Optional[Iterable[int]] = None, profile_list: Optional[int] = None, result_index: int = 0, next_guess_method: Union[Callable, str] = 'adaptive_step_regression', profile_options: Optional[pypesto.profile.options.ProfileOptions] = None, progress_bar: bool = True*) → *pypesto.result.Result*

This is the main function to call to do parameter profiling.

Parameters

- **problem** – The problem to be solved.
- **result** – A result object to initialize profiling and to append the profiling results to. For example, one might append more profiling runs to a previous profile, in order to merge these. The existence of an optimization result is obligatory.
- **optimizer** – The optimizer to be used along each profile.
- **engine** – The engine to be used.
- **profile_index** – List with the parameter indices to be profiled (by default all free indices).
- **profile_list** – Integer which specifies whether a call to the profiler should create a new list of profiles (default) or should be added to a specific profile list.
- **result_index** – Index from which optimization result profiling should be started (default: global optimum, i.e., index = 0).
- **next_guess_method** – Function handle to a method that creates the next starting point for optimization in profiling.
- **profile_options** – Various options applied to the profile optimization.
- **progress_bar** – Whether to display a progress bar.

Returns The profile results are filled into *result.profile_result*.

Return type result

4.8 Sample

Draw samples from the distribution, with support for various samplers.

class pypesto.sample.**AdaptiveMetropolisSampler**(*options: Optional[Dict] = None*)

Bases: pypesto.sample.metropolis.MetropolisSampler

Metropolis-Hastings sampler with adaptive proposal covariance.

__init__(*options: Optional[Dict] = None*)

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

classmethod default_options()

Convenience method to set/get default options.

Returns Default sampler options.

Return type default_options

initialize(*problem:* pypesto.problem.Problem, *x0:* *numpy.ndarray*)
Initialize the sampler.

Parameters

- **problem** – The problem for which to sample.
- **x0** – Should, but is not required to, be used as initial parameter.

class pypesto.sample.**AdaptiveParallelTemperingSampler**(*internal_sampler: pypesto.sample.sampler.InternalSampler*, *betas: Optional[Sequence[float]] = None*, *n_chains: Optional[int] = None*, *options: Optional[Dict] = None*)
Bases: pypesto.sample.parallel_tempering.ParallelTemperingSampler

Parallel tempering sampler with adaptive temperature adaptation.

adjust_betas(*i_sample: int*, *swapped: Sequence[bool]*)
Update temperatures as in Vousden2016.

classmethod default_options() → Dict
Convenience method to set/get default options.

Returns Default sampler options.

Return type default_options

class pypesto.sample.**EmceeSampler**(*nwalkers: int = 1*, *sampler_args: Optional[dict] = None*, *run_args: Optional[dict] = None*)
Bases: pypesto.sample.sampler.Sampler

Use emcee for sampling.

Wrapper around https://emcee.readthedocs.io/en/stable/, see there for details.

__init__(*nwalkers: int = 1*, *sampler_args: Optional[dict] = None*, *run_args: Optional[dict] = None*)

Parameters

- **nwalkers**(*The number of walkers in the ensemble.*) –
- **sampler_args** – Further keyword arguments that are passed on to emcee.EnsembleSampler.__init__.
- **run_args** – Further keyword arguments that are passed on to emcee.EnsembleSampler.run_mcmc.

get_samples() → pypesto.sample.result.McmcPtResult
Get the generated samples.

initialize(*problem:* pypesto.problem.Problem, *x0: Union[numpy.ndarray, List[numpy.ndarray]]*) → None
Initialize the sampler.

Parameters

- **problem** – The problem for which to sample.
- **x0** – Should, but is not required to, be used as initial parameter.

sample(*n_samples: int*, *beta: float = 1.0*) → None
Perform sampling.

Parameters

- **n_samples** – Number of samples to generate.
- **beta** – Inverse of the temperature to which the system is elevated.

class pypesto.sample.**InternalSampler**(*options: Optional[Dict] = None*)

Bases: pypesto.sample.sampler.Sampler

Sampler to be used inside a parallel tempering sampler.

The last sample can be obtained via *get_last_sample* and set via *set_last_sample*.

abstract get_last_sample() → pypesto.sample.sampler.InternalSample

Get the last sample in the chain.

Returns The last sample in the chain in the exchange format.

Return type internal_sample

make_internal(*temper_lpost: bool*)

This function can be called by parallel tempering samplers during initialization to allow the inner samplers to adjust to them being used as inner samplers. Default: Do nothing.

Parameters **temper_lpost** – Whether to temperate the posterior or only the likelihood.

abstract set_last_sample(*sample: pypesto.sample.sampler.InternalSample*)

Set the last sample in the chain to the passed value.

Parameters **sample** – The sample that will replace the last sample in the chain.

class pypesto.sample.**McmcPtResult**(*trace_x: numpy.ndarray*, *trace_neglogpost: numpy.ndarray*, *trace_neglogprior: numpy.ndarray*, *betas: Iterable[float]*, *burn_in: Optional[int] = None*, *time: float = 0.0*, *auto_correlation: Optional[float] = None*, *effective_sample_size: Optional[float] = None*, *message: Optional[str] = None*)

Bases: dict

The result of a sampler run using Markov-chain Monte Carlo, and optionally parallel tempering.

Can be used like a dict.

Parameters

- **trace_x** (*[n_chain, n_iter, n_par]*) – Parameters.
- **trace_neglogpost** (*[n_chain, n_iter]*) – Negative log posterior values.
- **trace_neglogprior** (*[n_chain, n_iter]*) – Negative log prior values.
- **betas** (*[n_chain]*) – The associated inverse temperatures.
- **burn_in** (*[n_chain]*) – The burn in index.
- **time** (*[n_chain]*) – The computation time.
- **auto_correlation** (*[n_chain]*) – The estimated chain autcorrelation.
- **effective_sample_size** (*[n_chain]*) – The estimated effective sample size.
- **message** (*str*) – Textual comment on the profile result.
- **Here** –
- **denotes the number of chains** (*n_chain*) –
- **the number of** (*n_iter*) –
- **(i.e.** (*iterations*) –

- **chain length)** (*the*) –
- **n_par the number of parameters.** (*and*) –

__init__(*trace_x: numpy.ndarray, trace_neglogpost: numpy.ndarray, trace_neglogprior: numpy.ndarray, betas: Iterable[float], burn_in: Optional[int] = None, time: float = 0.0, auto_correlation: Optional[float] = None, effective_sample_size: Optional[float] = None, message: Optional[str] = None*)
Initialize self. See help(type(self)) for accurate signature.

class pypesto.sample.**MetropolisSampler**(*options: Optional[Dict] = None*)
Bases: pypesto.sample.sampler.InternalSampler

Simple Metropolis-Hastings sampler with fixed proposal variance.

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

classmethod default_options()
Convenience method to set/get default options.

Returns Default sampler options.

Return type default_options

get_last_sample() → pypesto.sample.sampler.InternalSample
Get the last sample in the chain.

Returns The last sample in the chain in the exchange format.

Return type internal_sample

get_samples() → pypesto.sample.result.McmcPtResult
Get the generated samples.

initialize(*problem: pypesto.problem.Problem, x0: numpy.ndarray*)
Initialize the sampler.

Parameters

- **problem** – The problem for which to sample.
- **x0** – Should, but is not required to, be used as initial parameter.

make_internal(*temper_lpost: bool*)
This function can be called by parallel tempering samplers during initialization to allow the inner samplers to adjust to them being used as inner samplers. Default: Do nothing.

Parameters **temper_lpost** – Whether to temperate the posterior or only the likelihood.

sample(*n_samples: int, beta: float = 1.0*)
Perform sampling.

Parameters

- **n_samples** – Number of samples to generate.
- **beta** – Inverse of the temperature to which the system is elevated.

set_last_sample(*sample: pypesto.sample.sampler.InternalSample*)
Set the last sample in the chain to the passed value.

Parameters **sample** – The sample that will replace the last sample in the chain.

class pypesto.sample.**ParallelTemperingSampler**(*internal_sampler: pypesto.sample.sampler.InternalSampler*, *betas: Optional[Sequence[float]] = None*, *n_chains: Optional[int] = None*, *options: Optional[Dict] = None*)

Bases: pypesto.sample.sampler.Sampler

Simple parallel tempering sampler.

__init__(*internal_sampler: pypesto.sample.sampler.InternalSampler*, *betas: Optional[Sequence[float]] = None*, *n_chains: Optional[int] = None*, *options: Optional[Dict] = None*)

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

adjust_betas(*i_sample: int*, *swapped: Sequence[bool]*)

Adjust temperature values. Default: Do nothing.

classmethod default_options() → Dict

Convenience method to set/get default options.

Returns Default sampler options.

Return type default_options

get_samples() → pypesto.sample.result.McmcPtResult

Concatenate all chains.

initialize(*problem:* pypesto.problem.Problem, *x0: Union[numpy.ndarray, List[numpy.ndarray]]*)

Initialize the sampler.

Parameters

- **problem** – The problem for which to sample.
- **x0** – Should, but is not required to, be used as initial parameter.

sample(*n_samples: int*, *beta: float = 1.0*)

Perform sampling.

Parameters

- **n_samples** – Number of samples to generate.
- **beta** – Inverse of the temperature to which the system is elevated.

swap_samples() → Sequence[bool]

Swap samples as in Vousden2016.

class pypesto.sample.**Pymc3Sampler**(*step_function=None*, ***kwargs*)

Bases: pypesto.sample.sampler.Sampler

Wrapper around Pymc3 samplers.

Parameters

- **step_function** – A pymc3 step function, e.g. NUTS, Slice. If not specified, pymc3 determines one automatically (preferable).
- ****kwargs** – Options are directly passed on to *pymc3.sample*.

__init__(*step_function=None*, ***kwargs*)

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

get_samples() → pypesto.sample.result.McmcPtResult

Get the generated samples.

initialize(*problem:* pypesto.problem.Problem, *x0:* *numpy.ndarray*)
Initialize the sampler.

Parameters

- **problem** – The problem for which to sample.
- **x0** – Should, but is not required to, be used as initial parameter.

sample(*n_samples:* *int*, *beta:* *float* = *1.0*)
Perform sampling.

Parameters

- **n_samples** – Number of samples to generate.
- **beta** – Inverse of the temperature to which the system is elevated.

classmethod translate_options(*options*)
Convenience method to translate options and fill in defaults.

Parameters **options** – Options configuring the sampler.

class pypesto.sample.**Sampler**(*options: Optional[Dict]* = *None*)
Bases: abc.ABC

Sampler base class, not functional on its own.

The sampler maintains an internal chain, which is initialized in *initialize*, and updated in *sample*.

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

classmethod default_options() → Dict
Convenience method to set/get default options.

Returns Default sampler options.

Return type default_options

abstract get_samples() → pypesto.sample.result.McmcPtResult
Get the generated samples.

abstract initialize(*problem:* pypesto.problem.Problem, *x0:* *Union[numpy.ndarray, List[numpy.ndarray]]*)
Initialize the sampler.

Parameters

- **problem** – The problem for which to sample.
- **x0** – Should, but is not required to, be used as initial parameter.

abstract sample(*n_samples:* *int*, *beta:* *float* = *1.0*)
Perform sampling.

Parameters

- **n_samples** – Number of samples to generate.
- **beta** – Inverse of the temperature to which the system is elevated.

classmethod translate_options(*options*)
Convenience method to translate options and fill in defaults.

Parameters **options** – Options configuring the sampler.

pypesto.sample.**auto_correlation**(*result:* pypesto.result.Result) → float
Calculates the autocorrelation of the MCMC chains.

Parameters **result** – The pyPESTO result object with filled sample result.

Returns Estimate of the integrated autocorrelation time of the MCMC chains.

Return type auto_correlation

pypesto.sample.**calculate_ci_mcmc_sample**(*result:* pypesto.result.Result, *ci_level: float = 0.95*, *exclude_burn_in: bool = True*) → Tuple[numpy.ndarray, numpy.ndarray]
Calculate parameter credibility intervals based on MCMC samples.

Parameters

- **result** – The pyPESTO result object with filled sample result.
- **ci_level** – Lower tail probability, defaults to 95% interval.

Returns Bounds of the MCMC percentile-based confidence interval.

Return type lb, ub

pypesto.sample.**calculate_ci_mcmc_sample_prediction**(*simulated_values: numpy.ndarray*, *ci_level: float = 0.95*) → Tuple[numpy.ndarray, numpy.ndarray]
Calculate prediction credibility intervals based on MCMC samples.

Parameters

- **simulated_values** – Simulated model states or model observables.
- **ci_level** – Lower tail probability, defaults to 95% interval.

Returns Bounds of the MCMC-based prediction confidence interval.

Return type lb, ub

pypesto.sample.**effective_sample_size**(*result:* pypesto.result.Result) → float
Calculate the effective sample size of the MCMC chains.

Parameters **result** – The pyPESTO result object with filled sample result.

Returns Estimate of the effective sample size of the MCMC chains.

Return type ess

pypesto.sample.**geweke_test**(*result:* pypesto.result.Result, *zscore: float = 2.0*) → int
Calculates the burn-in of MCMC chains.

Parameters

- **result** – The pyPESTO result object with filled sample result.
- **zscore** – The Geweke test threshold. Default 2.

Returns Iteration where the first and the last fraction of the chain do not differ significantly regarding Geweke test -> Burn-In

Return type burn_in

pypesto.sample.**sample**(*problem: pypesto.problem.Problem, n_samples: int, sampler: Optional[pypesto.sample.sampler.Sampler] = None, x0: Optional[Union[numpy.ndarray, List[numpy.ndarray]]] = None, result: Optional[pypesto.result.Result] = None*) → *pypesto.result.Result*

This is the main function to call to do parameter sampling.

Parameters

- **problem** – The problem to be solved. If None is provided, a pypesto.AdaptiveMetropolisSampler is used.
- **n_samples** – Number of samples to generate.
- **sampler** – The sampler to perform the actual sampling.
- **x0** – Initial parameter for the Markov chain. If None, the best parameter found in optimization is used. Note that some samplers require an initial parameter, some may ignore it. x0 can also be a list, to have separate starting points for parallel tempering chains.
- **result** – A result to write to. If None provided, one is created from the problem.

Returns A result with filled in sample_options part.

Return type result

4.9 Result

The pypesto.Result object contains all results generated by the pypesto components. It contains sub-results for optimization, profiling, sampling.

class pypesto.result.**OptimizeResult**

Bases: object

Result of the minimize() function.

__init__()

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

append(*optimizer_result: optimize.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.

It holds a list of profile lists. Each profile list consists of a list of *ProfilerResult* objects, one for each parameter.

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

append_empty_profile_list() → int
Append an empty profile list to the list of profile lists.

Returns The index of the created profile list.

Return type index

append_profiler_result(*profiler_result: profile.ProfilerResult = None*, *profile_list: int = None*) → None
Append the profiler result to the profile list.

Parameters

- **profiler_result** – The result of one profiler run for a parameter, or None if to be left empty.
- **profile_list** – Index specifying the profile list to which we want to append. Defaults to the last list.

get_profiler_result(*i_par: int*, *profile_list: Optional[int] = None*)
Get theprofiler result at parameter index *i_par* of profile list *profile_list*.

Parameters

- **i_par** – Integer specifying the profile index.
- **profile_list** – Index specifying the profile list. Defaults to the last list.

set_profiler_result(*profiler_result: profile.ProfilerResult*, *i_par: int*, *profile_list: int = None*) → None
Write a profiler result to the result object at *i_par* of profile list *profile_list*.

Parameters

- **profiler_result** – The result of one (local) profiler run.
- **i_par** – Integer specifying the parameter index.
- **profile_list** – Index specifying the profile list. Defaults to the last list.

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.

__init__(*problem=None*)
Initialize self. See help(type(self)) for accurate signature.

class pypesto.result.**SampleResult**
Bases: object

Result of the sample() function.

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

4.10 Visualize

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

class pypesto.visualize.**ReferencePoint**(*reference=None*, *x=None*, *fval=None*, *color=None*, *legend=None*)
Bases: dict

Reference point for plotting. Should contain a parameter value and an objective function value, may also contain a color and a legend.

Can be used like a dict.

x
Reference parameters.

Type ndarray

fval
Function value, fun(x), for reference parameters.

Type float

color
Color which should be used for reference point.

Type RGBA, optional

auto_color
flag indicating whether color for this reference point should be assigned automatically or whether it was assigned by user

Type boolean

legend
legend text for reference point

Type str

__init__(*reference=None*, *x=None*, *fval=None*, *color=None*, *legend=None*)
Initialize self. See help(type(self)) for accurate signature.

pypesto.visualize.**assign_clustered_colors**(*vals*, *balance_alpha=True*, *highlight_global=True*)
Cluster and assign colors.

Parameters

- **vals** (*numeric list or array*) – List to be clustered and assigned colors.
- **balance_alpha** (*bool (optional)*) – Flag indicating whether alpha for large clusters should be reduced to avoid overplotting (default: True)
- **highlight_global** (*bool (optional)*) – flag indicating whether global optimum should be highlighted

Returns **colors** – One for each element in 'vals'.

Return type list of RGBA

pypesto.visualize.**assign_clusters**(*vals*)
Find clustering.

Parameters **vals** (*numeric list or array*) – List to be clustered.

Returns

- **clust** (*numeric list*) – Indicating the corresponding cluster of each element from 'vals'.
- **clustsize** (*numeric list*) – Size of clusters, length equals number of clusters.

pypesto.visualize.**assign_colors**(*vals*, *colors=None*, *balance_alpha=True*, *highlight_global=True*)
Assign colors or format user specified colors.

Parameters

- **vals** (*numeric list or array*) – List to be clustered and assigned colors.
- **colors** (*list, or RGBA, optional*) – list of colors, or single color
- **balance_alpha** (*bool (optional)*) – Flag indicating whether alpha for large clusters should be reduced to avoid overplotting (default: True)
- **highlight_global** (*bool (optional)*) – flag indicating whether global optimum should be highlighted

Returns **colors** – One for each element in 'vals'.

Return type list of RGBA

pypesto.visualize.**create_references**(*references=None*, *x=None*, *fval=None*, *color=None*, *legend=None*) → List[pypesto.visualize.reference_points.ReferencePoint]
This function creates a list of reference point objects from user inputs

Parameters

- **references** (*ReferencePoint or dict or list, optional*) – Will be converted into a list of RefPoints
- **x** (*ndarray, optional*) – Parameter vector which should be used for reference point
- **fval** (*float, optional*) – Objective function value which should be used for reference point
- **color** (*RGBA, optional*) – Color which should be used for reference point.
- **legend** (*str*) – legend text for reference point

Returns **colors** – One for each element in 'vals'.

Return type list of RGBA

pypesto.visualize.**delete_nan_inf**(*fvals: numpy.ndarray*, *x: Optional[numpy.ndarray] = None*, *xdim: Optional[int] = 1*) → Tuple[numpy.ndarray, numpy.ndarray]
Delete nan and inf values in fvals. If parameters 'x' are passed, also the corresponding entries are deleted.

Parameters

- **x** – array of parameters
- **fvals** – array of fval

- **xdim** – dimension of x, in case x dimension cannot be inferred

Returns

- *x* – array of parameters without nan or inf
- *fvals* – array of fval without nan or inf

pypesto.visualize.**ensemble_crosstab_scatter_lowlevel**(*dataset: numpy.ndarray*, *component_labels: Optional[Sequence[str]] = None*, ***kwargs*)

Plot cross-classification table of scatter plots for different coordinates. Lowlevel routine for multiple UMAP and PCA plots, but can also be used to visualize, e.g., parameter traces across optimizer runs

Parameters

- **dataset** – array of data points to be shown as scatter plot
- **component_labels** – labels for the x-axes and the y-axes

Returns A dictionary of plot axes.

Return type axs

pypesto.visualize.**ensemble_identifiability**(*ensemble: pypesto.ensemble.ensemble.Ensemble*, *ax: Optional[matplotlib.axes._axes.Axes] = None*, *size: Optional[Tuple[float]] = (12, 6)*)

Plots an overview about how many parameters hit the parameter bounds based on a ensemble of parameters. confidence intervals/credible ranges are computed via the ensemble mean plus/minus 1 standard deviation. This highlevel routine expects a ensemble object as input.

Parameters

- **ensemble** – ensemble of parameter vectors (from pypesto.ensemble)
- **ax** – Axes object to use.
- **size** – Figure size (width, height) in inches. Is only applied when no ax object is specified

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**ensemble_scatter_lowlevel**(*dataset*, *ax: Optional[matplotlib.axes._axes.Axes] = None*, *size: Optional[Tuple[float]] = (12, 6)*, *x_label: str = 'component 1'*, *y_label: str = 'component 2'*, *color_by: Optional[Sequence[float]] = None*, *color_map: str = 'viridis'*, *background_color: Tuple[float, float, float, float] = (0.0, 0.0, 0.0, 1.0)*, *marker_type: str = '.'*, *scatter_size: float = 0.5*, *invert_scatter_order: bool = False*)

Create a scatter plot

Parameters

- **dataset** – array of data points in reduced dimension
- **ax** – Axes object to use.
- **size** – Figure size (width, height) in inches. Is only applied when no ax object is specified

- **x_label** – The x-axis label
- **y_label** – The y-axis label
- **color_by** – A sequence/list of floats, which specify the color in the colormap
- **color_map** – A colormap name known to pyplot
- **background_color** – Background color of the axes object (defaults to black)
- **marker_type** – Type of plotted markers
- **scatter_size** – Size of plotted markers
- **invert_scatter_order** – Specifies the order of plotting the scatter points, can be important in case of overplotting

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**optimization_run_properties_one_plot**(*results:* pypesto.result.Result, *properties_to_plot: Optional[List[str]] = None*, *size: Tuple[float, float] = (18.5, 10.5)*, *start_indices: Optional[Union[int, Iterable[int]]] = None*, *colors: Optional[Union[List[float], List[List[float]]]] = None*, *legends: Optional[Union[str, List[str]]] = None*, *plot_type: str = 'line'*) → matplotlib.axes._axes.Axes

Plot stats for all optimization properties specified in properties_to_plot on one plot.

Parameters

- **results** – Optimization result obtained by 'optimize.py' or list of those
- **properties_to_plot** – Optimization run properties that should be plotted
- **size** – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **start_indices** – List of integers specifying the multistarts to be plotted or int specifying up to which start index should be plotted
- **colors** – List of RGBA colors (one color per property in properties_to_plot), or single RGBA color. If not set and one result, clustering is done and colors are assigned automatically
- **legends** – Labels, one label per optimization property
- **plot_type** – Specifies plot type. Possible values: 'line' and 'hist'

Examples

optimization_properties_per_multistart(result1, properties_to_plot=['time'], colors=[.5, .9, .9, .3])

optimization_properties_per_multistart(result1, properties_to_plot=['time', 'n_grad'], colors=[[.5, .9, .9, .3], [.2, .1, .9, .5]])

pypesto.visualize.**optimization_run_properties_per_multistart**(*results: Union[pypesto.result.Result, Sequence[pypesto.result.Result]], properties_to_plot: Optional[List[str]] = None, size: Tuple[float, float] = (18.5, 10.5), start_indices: Optional[Union[int, Iterable[int]]] = None, colors: Optional[Union[List[float], List[List[float]]]] = None, legends: Optional[Union[str, List[str]]] = None, plot_type: str = 'line'*) → Dict[str, matplotlib.axes._subplots.AxesSubplot]

One plot per optimization property in properties_to_plot.

Parameters

- **results** – Optimization result obtained by 'optimize.py' or list of those
- **properties_to_plot** – Optimization run properties that should be plotted
- **size** – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **start_indices** – List of integers specifying the multistarts to be plotted or int specifying up to which start index should be plotted
- **colors** – List of RGBA colors (one color per result in results), or single RGBA color. If not set and one result, clustering is done and colors are assigned automatically
- **legends** – Labels for line plots, one label per result object
- **plot_type** – Specifies plot type. Possible values: 'line' and 'hist'

Returns

- *ax*
- *The plot axes.*

Examples

optimization_properties_per_multistart(result1, properties_to_plot=['time'], colors=[.5, .9, .9, .3])

optimization_properties_per_multistart([result1, result2], properties_to_plot=['time'], colors=[[.5, .9, .9, .3], [.2, .1, .9, .5]])

optimization_properties_per_multistart(result1, properties_to_plot=['time', 'n_grad'], colors=[.5, .9, .9, .3])

optimization_properties_per_multistart([result1, result2], properties_to_plot=['time', 'n_fval'], colors=[[.5, .9, .9, .3], [.2, .1, .9, .5]])

pypesto.visualize.**optimization_run_property_per_multistart**(*results: Union[pypesto.result.Result, Sequence[pypesto.result.Result]], opt_run_property: str, axes: Optional[matplotlib.axes._axes.Axes] = None, size: Tuple[float, float] = (18.5, 10.5), start_indices: Optional[Union[int, Iterable[int]]] = None, colors: Optional[Union[List[float], List[List[float]]]] = None, legends: Optional[Union[str, List[str]]] = None, plot_type: str = 'line'*) → matplotlib.axes._axes.Axes

Plot stats for an optimization run property specified by opt_run_property. It is possible to plot a histogram or a line plot. In a line plot, on the x axis are the numbers of the multistarts, where the multistarts are ordered with respect to a function value. On the y axis of the line plot the value of the corresponding parameter for each multistart is displayed.

Parameters

- **opt_run_property** – optimization run property to plot. One of the 'time', 'n_fval', 'n_grad', 'n_hess', 'n_res', 'n_sres'
- **results** – Optimization result obtained by 'optimize.py' or list of those
- **axes** – Axes object to use
- **size** – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **start_indices** – List of integers specifying the multistarts to be plotted or int specifying up to which start index should be plotted
- **colors** – List of RGBA colors (one color per result in results), or single RGBA color. If not set and one result, clustering is done and colors are assigned automatically
- **legends** – Labels for line plots, one label per result object
- **plot_type** – Specifies plot type. Possible values: 'line', 'hist', 'both'

Returns The plot axes.

Return type ax

pypesto.visualize.**optimizer_convergence**(*result: pypesto.result.Result, ax: Optional[matplotlib.axes._axes.Axes] = None, xscale: str = 'symlog', yscale: str = 'log', size: Tuple[float] = (18.5, 10.5)*) → matplotlib.axes._axes.Axes

Scatter plot of function values and gradient values at the end of optimization. Optimizer exit-message is encoded by color. Can help identifying convergence issues in optimization and guide tolerance refinement etc.

Parameters

- **result** – Optimization result obtained by 'optimize.py'
- **ax** – Axes object to use.
- **size** – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **xscale** – Scale for x-axis
- **yscale** – Scale for y-axis

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**optimizer_history**(*results, ax=None, size=(18.5, 10.5), trace_x='steps', trace_y='fval', scale_y='log10', offset_y=None, colors=None, y_limits=None, start_indices=None, reference=None, legends=None*)

Plot history of optimizer. Can plot either the history of the cost function or of the gradient norm, over either the optimizer steps or the computation time.

Parameters

- **results** (*pypesto.Result or list*) – Optimization result obtained by 'optimize.py' or list of those
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **size** (*tuple, optional*) – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **trace_x** (*str, optional*) – What should be plotted on the x-axis? Possibilities: 'time', 'steps' Default: 'steps'
- **trace_y** (*str, optional*) – What should be plotted on the y-axis? Possibilities: 'fval', 'gradnorm', 'stepsize' Default: 'fval'
- **scale_y** (*str, optional*) – May be logarithmic or linear ('log10' or 'lin')
- **offset_y** (*float, optional*) – Offset for the y-axis-values, as these are plotted on a log10-scale Will be computed automatically if necessary
- **colors** (*list, or RGBA, optional*) – list of colors, or single color color or list of colors for plotting. If not set, clustering is done and colors are assigned automatically
- **y_limits** (*float or ndarray, optional*) – maximum value to be plotted on the y-axis, or y-limits
- **start_indices** (*list or int*) – list of integers specifying the multistart to be plotted or int specifying up to which start index should be plotted
- **reference** (*list, optional*) – List of reference points for optimization results, containing et least a function value fval

- **legends** (*list or str*) – Labels for line plots, one label per result object

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**optimizer_history_lowlevel**(*vals*, *scale_y='log10'*, *colors=None*, *ax=None*, *size=(18.5, 10.5)*, *x_label='Optimizer steps'*, *y_label='Objective value'*, *legend_text=None*)

Plot optimizer history using list of numpy arrays.

Parameters

- **vals** (*list of numpy arrays*) – list of 2xn-arrays (x_values and y_values of the trace)
- **scale_y** (*str, optional*) – May be logarithmic or linear ('log10' or 'lin')
- **colors** (*list, or RGBA, optional*) – list of colors, or single color color or list of colors for plotting. If not set, clustering is done and colors are assigned automatically
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **size** (*tuple, optional*) – see waterfall
- **x_label** (*str*) – label for x-axis
- **y_label** (*str*) – label for y-axis
- **legend_text** (*str*) – Label for line plots

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**parameter_hist**(*result: pypesto.result.Result*, *parameter_name: str*, *bins: Union[int, str] = 'auto'*, *ax: Optional[matplotlib.Axes] = None*, *size: Optional[Tuple[float]] = (18.5, 10.5)*, *color: Optional[List[float]] = None*, *start_indices: Optional[Union[int, List[int]]] = None*)

Plot parameter values as a histogram.

Parameters

- **result** – Optimization result obtained by 'optimize.py'
- **parameter_name** – The name of the parameter that should be plotted
- **bins** – Specifies bins of the histogram
- **ax** – Axes object to use
- **size** – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **color** – RGBA color.
- **start_indices** – List of integers specifying the multistarts to be plotted or int specifying up to which start index should be plotted

Returns

- *ax*
- *The plot axes.*

pypesto.visualize.**parameters**(*results: Union[pypesto.result.Result, Sequence[pypesto.result.Result]], ax: Optional[matplotlib.axes._axes.Axes] = None, parameter_indices: Union[str, Sequence[int]] = 'free_only', lb: Optional[Union[numpy.ndarray, List[float]]] = None, ub: Optional[Union[numpy.ndarray, List[float]]] = None, size: Optional[Tuple[float, float]] = None, reference: Optional[List[pypesto.visualize.reference_points.ReferencePoint]] = None, colors: Optional[Union[List[float], List[List[float]]]] = None, legends: Optional[Union[str, List[str]]] = None, balance_alpha: bool = True, start_indices: Optional[Union[int, Iterable[int]]] = None*) → matplotlib.axes._axes.Axes

Plot parameter values.

Parameters

- **results** – Optimization result obtained by 'optimize.py' or list of those
- **ax** – Axes object to use.
- **parameter_indices** – Specifies which parameters should be plotted. Allowed string values are 'all' (both fixed and free parameters will be plotted) and 'free_only' (only free parameters will be plotted)
- **lb** – If not None, override result.problem.lb, problem.problem.ub. Dimension either result.problem.dim or result.problem.dim_full.
- **ub** – If not None, override result.problem.lb, problem.problem.ub. Dimension either result.problem.dim or result.problem.dim_full.
- **size** – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **reference** – List of reference points for optimization results, containing at least a function value fval
- **colors** – list of RGBA colors, or single RGBA color If not set, clustering is done and colors are assigned automatically
- **legends** – Labels for line plots, one label per result object
- **balance_alpha** – Flag indicating whether alpha for large clusters should be reduced to avoid overplotting (default: True)
- **start_indices** – list of integers specifying the multistarts to be plotted or int specifying up to which start index should be plotted

Returns The plot axes.

Return type ax

pypesto.visualize.**parameters_lowlevel**(*xs: Sequence[Union[numpy.ndarray, List[float]]], fvals: Union[numpy.ndarray, List[float]], lb: Optional[Union[numpy.ndarray, List[float]]] = None, ub: Optional[Union[numpy.ndarray, List[float]]] = None, x_labels: Optional[Iterable[str]] = None, ax: Optional[matplotlib.axes._axes.Axes] = None, size: Optional[Tuple[float, float]] = None, colors: Optional[Sequence[Union[numpy.ndarray, List[float]]]] = None, linestyle: str = '-', legend_text: Optional[str] = None, balance_alpha: bool = True*) → matplotlib.axes._axes.Axes

Plot parameters plot using list of parameters.

Parameters

- **xs** – Including optimized parameters for each startpoint. Shape: (n_starts, dim).
- **fvals** – Function values. Needed to assign cluster colors.
- **lb** – The lower and upper bounds.
- **ub** – The lower and upper bounds.
- **x_labels** – Labels to be used for the parameters.
- **ax** – Axes object to use.
- **size** – see parameters
- **colors** – One for each element in 'fvals'.
- **linestyle** – linestyle argument for parameter plot
- **legend_text** – Label for line plots
- **balance_alpha** – Flag indicating whether alpha for large clusters should be reduced to avoid overplotting (default: True)

Returns The plot axes.

Return type ax

pypesto.visualize.**process_offset_y**(*offset_y: Optional[float]*, *scale_y: str*, *min_val: float*) → float

compute offset for y-axis, depend on user settings

Parameters

- **offset_y** – value for offsetting the later plotted values, in order to ensure positivity if a semilog-plot is used
- **scale_y** – Can be 'lin' or 'log10', specifying whether values should be plotted on linear or on log10-scale
- **min_val** – Smallest value to be plotted

Returns **offset_y** – value for offsetting the later plotted values, in order to ensure positivity if a semilog-plot is used

Return type float

pypesto.visualize.**process_result_list**(*results*, *colors=None*, *legends=None*)

assigns colors and legends to a list of results, check user provided lists

Parameters

- **results** (*list or pypesto.Result*) – list of pypesto.Result objects or a single pypesto.Result
- **colors** (*list, optional*) – list of RGBA colors
- **legends** (*str or list*) – labels for line plots

Returns

- **results** (*list of pypesto.Result*) – list of pypesto.Result objects
- **colors** (*list of RGBA*) – One for each element in 'results'.
- **legends** (*list of str*) – labels for line plots

pypesto.visualize.**process_y_limits**(*ax*, *y_limits*)
apply user specified limits of y-axis

Parameters

- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **y_limits** (*ndarray*) – y_limits, minimum and maximum, for current axes object

Returns **ax** – Axes object to use.

Return type matplotlib.Axes, optional

pypesto.visualize.**profile_cis**(*result: pypesto.result.Result*, *confidence_level: float = 0.95*, *profile_indices: Optional[Sequence[int]] = None*, *profile_list: int = 0*, *color: Union[str, tuple] = 'C0'*, *show_bounds: bool = False*, *ax: Optional[matplotlib.axes._axes.Axes] = None*) → matplotlib.axes._axes.Axes
Plot approximate confidence intervals based on profiles.

Parameters

- **result** – The result object after profiling.
- **confidence_level** – The confidence level in (0,1), which is translated to an approximate threshold assuming a chi2 distribution, using *pypesto.profile.chi2_quantile_to_ratio*.
- **profile_indices** – List of integer values specifying which profiles should be plotted. Defaults to the indices for which profiles were generated in profile list *profile_list*.
- **profile_list** – Index of the profile list to be used.
- **color** – Main plot color.
- **show_bounds** – Whether to show, and extend the plot to, the lower and upper bounds.
- **ax** – Axes object to use. Default: Create a new one.

pypesto.visualize.**profile_lowlevel**(*fvals*, *ax=None*, *size: Tuple[float, float] = (18.5, 6.5)*, *color=None*, *legend_text: Optional[str] = None*, *show_bounds: bool = False*, *lb: Optional[float] = None*, *ub: Optional[float] = None*)
Lowlevel routine for plotting one profile, working with a numpy array only

Parameters

- **fvals** (*numeric list or array*) – Values to plot.
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **size** (*tuple, optional*) – Figure size (width, height) in inches. Is only applied when no ax object is specified.
- **color** (*RGBA, optional*) – Color for profiles in plot.
- **legend_text** (*str*) – Label for line plots.
- **show_bounds** – Whether to show, and extend the plot to, the lower and upper bounds.
- **lb** – Lower bound.
- **ub** – Upper bound.

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**profiles**(*results: Union[pypesto.result.Result, Sequence[pypesto.result.Result]], ax=None, profile_indices: Optional[Sequence[int]] = None, size: Sequence[float] = (18.5, 6.5), reference: Optional[Union[pypesto.visualize.reference_points.ReferencePoint, Sequence[pypesto.visualize.reference_points.ReferencePoint]]] = None, colors=None, legends: Optional[Sequence[str]] = None, x_labels: Optional[Sequence[str]] = None, profile_list_ids: Union[int, Sequence[int]] = 0, ratio_min: float = 0.0, show_bounds: bool = False*)

Plot classical 1D profile plot (using the posterior, e.g. Gaussian like profile)

Parameters

- **results** (*list or pypesto.Result*) – List of or single *pypesto.Result* after profiling.
- **ax** (*list of matplotlib.Axes, optional*) – List of axes objects to use.
- **profile_indices** (*list of integer values*) – List of integer values specifying which profiles should be plotted.
- **size** (*tuple, optional*) – Figure size (width, height) in inches. Is only applied when no ax object is specified.
- **reference** (*list, optional*) – List of reference points for optimization results, containing at least a function value fval.
- **colors** (*list, or RGBA, optional*) – List of colors, or single color.
- **legends** (*list or str, optional*) – Labels for line plots, one label per result object.
- **x_labels** (*list of str*) – Labels for parameter value axes (e.g. parameter names).
- **profile_list_ids** (*int or list of ints, optional*) – Index or list of indices of the profile lists to be used for profiling.
- **ratio_min** – Minimum ratio below which to cut off.
- **show_bounds** – Whether to show, and extend the plot to, the lower and upper bounds.

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**profiles_lowlevel**(*fvals, ax=None, size: Tuple[float, float] = (18.5, 6.5), color=None, legend_text: Optional[str] = None, x_labels=None, show_bounds: bool = False, lb_full=None, ub_full=None*)

Lowlevel routine for profile plotting, working with a list of arrays only, opening different axes objects in case

Parameters

- **fvals** (*numeric list or array*) – Values to plot.
- **ax** (*list of matplotlib.Axes, optional*) – List of axes object to use.
- **size** (*tuple, optional*) – Figure size (width, height) in inches. Is only applied when no ax object is specified.
- **size** – Figure size (width, height) in inches. Is only applied when no ax object is specified.
- **color** (*RGBA, optional*) – Color for profiles in plot.

- **legend_text** (*List[str]*) – Label for line plots.
- **legend_text** – Label for line plots.
- **show_bounds** – Whether to show, and extend the plot to, the lower and upper bounds.
- **lb_full** – Lower bound.
- **ub_full** – Upper bound.

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**projection_scatter_pca**(*pca_coordinates: numpy.ndarray*, *components: Sequence[int] = (0, 1)*, ***kwargs*)

Plot a scatter plots for PCA coordinates. Creates either one or multiple scatter plots, depending on the number of coordinates passed to it.

Parameters

- **pca_coordinates** – array of pca coordinates (returned as first output by the routine get_pca_representation) to be shown as scatter plot
- **components** – Components to be plotted (corresponds to columns of pca_coordinates)

Returns Either one axes object, or a dictionary of plot axes (depending on the number of coordinates passed)

Return type axs

pypesto.visualize.**projection_scatter_umap**(*umap_coordinates: numpy.ndarray*, *components: Sequence[int] = (0, 1)*, ***kwargs*)

Plot a scatter plots for UMAP coordinates. Creates either one or multiple scatter plots, depending on the number of coordinates passed to it.

Parameters

- **umap_coordinates** – array of umap coordinates (returned as first output by the routine get_umap_representation) to be shown as scatter plot
- **components** – Components to be plotted (corresponds to columns of umap_coordinates)

Returns Either one axes object, or a dictionary of plot axes (depending on the number of coordinates passed)

Return type axs

pypesto.visualize.**projection_scatter_umap_original**(*umap_object: None*, *color_by: Optional[Sequence[float]] = None*, *components: Sequence[int] = (0, 1)*, ***kwargs*)

Wrapper around umap.plot.points. Similar to projection_scatter_umap, but uses the original plotting routine from umap.plot.

Parameters

- **umap_object** – umap object (returned as second output by get_umap_representation) to be shown as scatter plot
- **color_by** – A sequence/list of floats, which specify the color in the colormap
- **components** – Components to be plotted (corresponds to columns of umap_coordinates)

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**sampling_1d_marginals**(*result: pypesto.result.Result, i_chain: int = 0, par_indices: Optional[Sequence[int]] = None, stepsize: int = 1, plot_type: str = 'both', bw: str = 'scott', suptitle: Optional[str] = None, size: Optional[Tuple[float, float]] = None*)

Plot marginals.

Parameters

- **result** – The pyPESTO result object with filled sample result.
- **i_chain** – Which chain to plot. Default: First chain.
- **par_indices** (*list of integer values*) – List of integer values specifying which parameters to plot. Default: All parameters are shown.
- **stepsize** – Only one in *stepsize* values is plotted.
- **plot_type** (*{'hist'|'kde'|'both'}*) – Specify whether to plot a histogram ('hist'), a kernel density estimate ('kde'), or both ('both').
- **bw** (*{'scott', 'silverman' | scalar | pair of scalars}*) – Kernel bandwidth method.
- **suptitle** – Figure super title.
- **size** – Figure size in inches.

Returns ax

Return type matplotlib-axes

pypesto.visualize.**sampling_fval_traces**(*result: pypesto.result.Result, i_chain: int = 0, full_trace: bool = False, stepsize: int = 1, title: Optional[str] = None, size: Optional[Tuple[float, float]] = None, ax: Optional[matplotlib.axes._axes.Axes] = None*)

Plot log-posterior (=function value) over iterations.

Parameters

- **result** – The pyPESTO result object with filled sample result.
- **i_chain** – Which chain to plot. Default: First chain.
- **full_trace** – Plot the full trace including warm up. Default: False.
- **stepsize** – Only one in *stepsize* values is plotted.
- **title** – Axes title.
- **size** (*ndarray*) – Figure size in inches.
- **ax** – Axes object to use.

Returns The plot axes.

Return type ax

pypesto.visualize.**sampling_parameter_cis**(*result: pypesto.result.Result, alpha: Optional[Sequence[int]] = None, step: float = 0.05, show_median: bool = True, title: Optional[str] = None, size: Optional[Tuple[float, float]] = None, ax: Optional[matplotlib.axes._axes.Axes] = None*) → matplotlib.axes._axes.Axes

Plot MCMC-based parameter credibility intervals for one or more credibility levels.

Parameters

- **result** – The pyPESTO result object with filled sample result.
- **alpha** – List of lower tail probabilities, defaults to 95% interval.
- **step** – Height of boxes for projectile plot, defaults to 0.05.
- **show_median** – Plot the median of the MCMC chain. Default: True.
- **title** – Axes title.
- **size** (*ndarray*) – Figure size in inches.
- **ax** – Axes object to use.

Returns The plot axes.

Return type ax

pypesto.visualize.**sampling_parameter_traces**(*result: pypesto.result.Result*, *i_chain: int = 0*, *par_indices: Optional[Sequence[int]] = None*, *full_trace: bool = False*, *stepsize: int = 1*, *use_problem_bounds: bool = True*, *suptitle: Optional[str] = None*, *size: Optional[Tuple[float, float]] = None*, *ax: Optional[matplotlib.axes._axes.Axes] = None*)

Plot parameter values over iterations.

Parameters

- **result** – The pyPESTO result object with filled sample result.
- **i_chain** – Which chain to plot. Default: First chain.
- **par_indices** (*list of integer values*) – List of integer values specifying which parameters to plot. Default: All parameters are shown.
- **full_trace** – Plot the full trace including warm up. Default: False.
- **stepsize** – Only one in *stepsize* values is plotted.
- **use_problem_bounds** – Defines if the y-limits shall be the lower and upper bounds of parameter estimation problem.
- **suptitle** – Figure suptitle.
- **size** – Figure size in inches.
- **ax** – Axes object to use.

Returns The plot axes.

Return type ax

pypesto.visualize.**sampling_prediction_trajectories**(*ensemble_prediction: pypesto.ensemble.ensemble.EnsemblePrediction*, *levels: Union[float, Sequence[float]]*, *title: Optional[str] = None*, *size: Optional[Tuple[float, float]] = None*, *axes: Optional[matplotlib.axes._axes.Axes] = None*, *labels: Optional[Dict[str, str]] = None*, *axis_label_padding: int = 50*, *groupby: str = 'condition'*, *condition_gap: float = 0.01*, *condition_ids: Optional[Sequence[str]] = None*, *output_ids: Optional[Sequence[str]] = None*) → matplotlib.axes._axes.Axes

Plot MCMC-based prediction credibility intervals for the model states or outputs. One or various credibility levels can be depicted. Plots are grouped by condition.

Parameters

- **result** – The pyPESTO result object with filled sample result.
- **levels** – Credibility levels, e.g. [95] for a 95% credibility interval. See the `_get_level_percentiles()` method for a description of how these levels are handled, and current limitations.
- **title** – Axes title.
- **size** (*ndarray*) – Figure size in inches.
- **axes** – Axes object to use.
- **labels** – Keys should be ensemble output IDs, values should be the desired label for that output. Defaults to output IDs.
- **axis_label_padding** – Pixels between axis labels and plots.
- **groupby** – Group plots by *pypesto.predict.constants.OUTPUT* or *pypesto.predict.constants.CONDITION*.
- **condition_gap** – Gap between conditions when *groupby == pypesto.predict.constants.CONDITION*.
- **condition_ids** – If provided, only data for the provided condition IDs will be plotted.
- **output_ids** – If provided, only data for the provided output IDs will be plotted.

Returns The plot axes.

Return type axes

pypesto.visualize.**sampling_scatter**(*result:* pypesto.result.Result, *i_chain: int = 0*, *stepsize: int = 1*, *suptitle: Optional[str] = None*, *diag_kind: str = 'kde'*, *size: Optional[Tuple[float, float]] = None*)

Parameter scatter plot.

Parameters

- **result** – The pyPESTO result object with filled sample result.
- **i_chain** – Which chain to plot. Default: First chain.

- **stepsize** – Only one in *stepsize* values is plotted.
- **suptitle** – Figure super title.
- **diag_kind** – Visualization mode for marginal densities {'auto', 'hist', 'kde', None}
- **size** – Figure size in inches.

Returns The plot axes.

Return type ax

pypesto.visualize.**waterfall**(*results: Union[pypesto.result.Result, Sequence[pypesto.result.Result]], ax: Optional[matplotlib.axes._axes.Axes] = None, size: Optional[Tuple[float]] = (18.5, 10.5), y_limits: Optional[Tuple[float]] = None, scale_y: Optional[str] = 'log10', offset_y: Optional[float] = None, start_indices: Optional[Union[Sequence[int], int]] = None, reference: Optional[Sequence[pypesto.visualize.reference_points.ReferencePoint]] = None, colors: Optional[Union[Tuple[float, float, float, float], Sequence[Tuple[float, float, float, float]]]] = None, legends: Optional[Union[Sequence[str], str]] = None*)

Plot waterfall plot.

Parameters

- **results** – Optimization result obtained by 'optimize.py' or list of those
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **size** – Figure size (width, height) in inches. Is only applied when no ax object is specified
- **y_limits** (*float or ndarray, optional*) – maximum value to be plotted on the y-axis, or y-limits
- **scale_y** – May be logarithmic or linear ('log10' or 'lin')
- **offset_y** – offset for the y-axis, if it is supposed to be in log10-scale
- **start_indices** – Integers specifying the multistart to be plotted or int specifying up to which start index should be plotted
- **reference** – Reference points for optimization results, containing at least a function value fval
- **colors** – Colors or single color for plotting. If not set, clustering is done and colors are assigned automatically
- **legends** – Labels for line plots, one label per result object

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**waterfall_lowlevel**(*fvals, scale_y='log10', offset_y=0.0, ax=None, size=(18.5, 10.5), colors=None, legend_text=None*)

Plot waterfall plot using list of function values.

Parameters

- **fvals** (*numeric list or array*) – Including values need to be plotted.
- **scale_y** (*str, optional*) – May be logarithmic or linear ('log10' or 'lin')
- **offset_y** – offset for the y-axis, if it is supposed to be in log10-scale

- **ax** (*matplotlib.Axes, optional*) – Axes object to use.
- **size** (*tuple, optional*) – see waterfall
- **colors** (*list, or RGBA, optional*) – list of colors, or single color color or list of colors for plotting. If not set, clustering is done and colors are assigned automatically
- **legend_text** (*str*) – Label for line plots

Returns **ax** – The plot axes.

Return type matplotlib.Axes

4.11 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.

class pypesto.engine.**Engine**

Bases: abc.ABC

Abstract engine base class.

__init__()

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

abstract execute(*tasks: List[pypesto.engine.task.Task]*, *progress_bar: bool = True*)

Execute tasks.

Parameters

- **tasks** – List of tasks to execute.
- **progress_bar** – Whether to display a progress bar.

class pypesto.engine.**MultiProcessEngine**(*n_procs: Optional[int] = None*)

Bases: pypesto.engine.base.Engine

Parallelize the task execution using multiprocessing.

Parameters **n_procs** – The maximum number of processes to use in parallel. Defaults to the number of CPUs available on the system according to *os.cpu_count()*. The effectively used number of processes will be the minimum of *n_procs* and the number of tasks submitted.

__init__(*n_procs: Optional[int] = None*)

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

execute(*tasks: List[pypesto.engine.task.Task]*, *progress_bar: bool = True*)

Pickle tasks and distribute work over parallel processes.

Parameters

- **tasks** – List of tasks to execute.
- **progress_bar** – Whether to display a progress bar.

class pypesto.engine.**MultiThreadEngine**(*n_threads: Optional[int] = None*)

Bases: pypesto.engine.base.Engine

Parallelize the task execution using multithreading.

Parameters **n_threads** – The maximum number of threads to use in parallel. Defaults to the number of CPUs available on the system according to *os.cpu_count()*. The effectively used number of threads will be the minimum of *n_threads* and the number of tasks submitted.

__init__(*n_threads: Optional[int] = None*)
Initialize self. See help(type(self)) for accurate signature.

execute(*tasks: List[pypesto.engine.task.Task]*, *progress_bar: bool = True*)
Deepcopy tasks and distribute work over parallel threads.

Parameters

- **tasks** – List of tasks to execute.
- **progress_bar** – Whether to display a progress bar.

class pypesto.engine.**SingleCoreEngine**
Bases: pypesto.engine.base.Engine

Dummy engine for sequential execution on one core. Note that the objective itself may be multithreaded.

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

execute(*tasks: List[pypesto.engine.task.Task]*, *progress_bar: bool = True*)
Execute all tasks in a simple for loop sequentially.

Parameters

- **tasks** – List of tasks to execute.
- **progress_bar** – Whether to display a progress bar.

class pypesto.engine.**Task**
Bases: abc.ABC

A task is one of a list of independent execution tasks that are submitted to the execution engine to be executed using the execute() method, commonly in parallel.

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

abstract execute()
Execute the task and return its results.

4.12 Startpoint

Methods for selecting points that can be used as start points for multistart optimization. All methods have the form

```
method(**kwargs) -> startpoints
```

where the kwargs can/should include the following parameters, which are passed by pypesto:

n_starts: int Number of points to generate.

lb, ub: ndarray Lower and upper bound, may for most methods not contain nan or inf values.

x_guesses: ndarray, shape=(g, dim), optional Parameter guesses by the user, where g denotes the number of guesses. Note that these are only possibly taken as reference points to generate new start points (e.g. to maximize some distance) depending on the method, but regardless of g, there are always n_starts points generated and returned.

objective: pypesto.Objective, optional The objective can be used to evaluate the goodness of start points.

max_n_fval: int, optional The maximum number of evaluations of the objective function allowed.

pypesto.startpoint.**assign_startpoints**(*n_starts: int*, *startpoint_method: Callable*, *problem: pypesto.problem.Problem*, *startpoint_resample: bool*) → numpy.ndarray

Assign start points.

pypesto.startpoint.**latin_hypercube**(***kwargs*) → numpy.ndarray

Generate latin hypercube points.

Parameters

- **n_starts** – number of starting points to be sampled.
- **lb** – lower bound.
- **ub** – upper bound.
- **smooth** – indicates if a (uniformly chosen) random starting point within the hypercube [i/n_starts, (i+1)/n_starts] should be chosen (True) or the midpoint of the interval (False). Default is True.

pypesto.startpoint.**uniform**(***kwargs*) → numpy.ndarray

Generate uniform points.

4.13 Storage

Saving and loading traces and results objects.

class pypesto.store.**OptimizationResultHDF5Reader**(*storage_filename: str*)

Bases: object

Reader of the HDF5 result files written by class OptimizationResultHDF5Writer.

storage_filename

HDF5 result file name

__init__(*storage_filename: str*)

Parameters **storage_filename** (*str*) – HDF5 result file name

read() → *pypesto.result.Result*

Read HDF5 result file and return pyPESTO result object.

class pypesto.store.**OptimizationResultHDF5Writer**(*storage_filename: str*)

Bases: object

Writer of the HDF5 result files.

storage_filename

HDF5 result file name

__init__(*storage_filename: str*)

Parameters **storage_filename** (*str*) – HDF5 result file name

write(*result: pypesto.result.Result*, *overwrite=False*)

Write HDF5 result file from pyPESTO result object.

class pypesto.store.**ProblemHDF5Reader**(*storage_filename: str*)

Bases: object

Reader of the HDF5 problem files written by class ProblemHDF5Writer.

storage_filename
HDF5 problem file name

__init__(*storage_filename: str*)

Parameters **storage_filename** (*str*) – HDF5 problem file name

read(*objective: Optional[pypesto.objective.base.ObjectiveBase] = None*) → *pypesto.problem.Problem*
Read HDF5 problem file and return pyPESTO problem object.

Parameters **objective** – Objective function which is currently not saved to storage.

Returns A problem instance with all attributes read in.

Return type problem

class pypesto.store.**ProblemHDF5Writer**(*storage_filename: str*)
Bases: object

Writer of the HDF5 problem files.

storage_filename
HDF5 result file name

__init__(*storage_filename: str*)

Parameters **storage_filename** (*str*) – HDF5 problem file name

write(*problem*, *overwrite: bool = False*)
Write HDF5 problem file from pyPESTO problem object.

class pypesto.store.**ProfileResultHDF5Reader**(*storage_filename: str*)
Bases: object

Reader of the HDF5 result files written by class OptimizationResultHDF5Writer.

storage_filename
HDF5 result file name

__init__(*storage_filename: str*)

Parameters **storage_filename** (*str*) – HDF5 result file name

read() → *pypesto.result.Result*
Read HDF5 result file and return pyPESTO result object.

class pypesto.store.**ProfileResultHDF5Writer**(*storage_filename: str*)
Bases: object

Writer of the HDF5 result files.

storage_filename
HDF5 result file name

__init__(*storage_filename: str*)

Parameters **storage_filename** (*str*) – HDF5 result file name

write(*result:* pypesto.result.Result, *overwrite: bool = False*)
Write HDF5 result file from pyPESTO result object.

class pypesto.store.**SamplingResultHDF5Reader**(*storage_filename: str*)
Bases: object

Reader of the HDF5 result files written by class SamplingResultHDF5Writer.

storage_filename
HDF5 result file name

__init__(*storage_filename: str*)

Parameters **storage_filename** (*str*) – HDF5 result file name

read() → *pypesto.result.Result*
Read HDF5 result file and return pyPESTO result object.

class pypesto.store.**SamplingResultHDF5Writer**(*storage_filename: str*)
Bases: object

Writer of the HDF5 sampling files.

storage_filename
HDF5 result file name

__init__(*storage_filename: str*)

Parameters **storage_filename** (*str*) – HDF5 result file name

write(*result:* pypesto.result.Result, *overwrite: bool = False*)
Write HDF5 sampling file from pyPESTO result object.

pypesto.store.**read_result**(*filename: str*, *problem: bool = True*, *optimize: bool = False*, *profile: bool = False*, *sample: bool = False*) → *pypesto.result.Result*
This is a function that saves the whole pypesto.Result object in an HDF5 file. With booleans one can choose more detailed what to save.

Parameters

- **filename** – The HDF5 filename.
- **problem** – Read the problem.
- **optimize** – Read the optimize result.
- **profile** – Read the profile result.
- **sample** – Read the sample result.

Returns Result object containing the results stored in HDF5 file.

Return type result

pypesto.store.**write_result**(*result:* pypesto.result.Result, *filename: str*, *overwrite: bool = False*, *problem: bool = True*, *optimize: bool = True*, *profile: bool = True*, *sample: bool = True*)
Save whole pypesto.Result to hdf5 file.

Boolean indicators allow specifying what to save.

Parameters

- **result** – The *pypesto.Result* object to be saved.
- **filename** – The HDF5 filename.
- **overwrite** – Boolean, whether already existing results should be overwritten.
- **problem** – Read the problem.
- **optimize** – Read the optimize result.
- **profile** – Read the profile result.
- **sample** – Read the sample result.

4.14 Logging

Logging convenience functions.

pypesto.logging.**log**(*name: str = 'pypesto'*, *level: int = 10*, *console: bool = False*, *filename: str = ''*)
Log messages from a specified name with a specified level to any combination of console and file.

Parameters

- **name** – The name of the logger.
- **level** – The output level to use.
- **console** – If True, messages are logged to console.
- **filename** – If specified, messages are logged to a file with this name.

pypesto.logging.**log_to_console**(*level: int = 10*)
Log to console.

Parameters the log method. (*See*) –

pypesto.logging.**log_to_file**(*level: int = 10*, *filename: str = '.pypesto_logging.log'*)
Log to file.

Parameters the log method. (*See*) –

4.15 Ensemble

class pypesto.ensemble.**Ensemble**(*x_vectors: numpy.ndarray*, *x_names: Optional[Sequence[str]] = None*, *vector_tags: Optional[Sequence[Tuple[int, int]]] = None*, *ensemble_type: Optional[pypesto.ensemble.constants.EnsembleType] = None*, *predictions: Optional[Sequence[pypesto.ensemble.ensemble.EnsemblePrediction]] = None*, *lower_bound: Optional[numpy.ndarray] = None*, *upper_bound: Optional[numpy.ndarray] = None*)
Bases: object

A ensemble is a wrapper around an numpy array. It comes with some convenience functionality: It allows to map parameter values via identifiers to the correct parameters, it allows to compute summaries of the parameter vectors (mean, standard deviation, median, percentiles) more easily, and it can store predictions made by pyPESTO, such that the parameter ensemble and the predictions are linked to each other.

__init__(*x_vectors: numpy.ndarray*, *x_names: Optional[Sequence[str]] = None*, *vector_tags: Optional[Sequence[Tuple[int, int]]] = None*, *ensemble_type: Optional[pypesto.ensemble.constants.EnsembleType] = None*, *predictions: Optional[Sequence[pypesto.ensemble.ensemble.EnsemblePrediction]] = None*, *lower_bound: Optional[numpy.ndarray] = None*, *upper_bound: Optional[numpy.ndarray] = None*)
Constructor.

Parameters

- **x_vectors** – parameter vectors of the ensemble, in the format n_parameter x n_vectors
- **x_names** – Names or identifiers of the parameters
- **vector_tags** – Additional tag, which adds information about the the parameter vectors of the form (optimization_run, optimization_step) if the ensemble is created from an

optimization result or (sampling_chain, sampling_step) if the ensemble is created from a sampling result.

- **ensemble_type** – Type of ensemble: Ensemble (default), sample, or unprocessed_chain Samples are meant to be representative, ensembles can be any ensemble of parameters, and unprocessed chains still have burn-ins
- **predictions** – List of EnsemblePrediction objects
- **lower_bound** – array of potential lower bounds for the parameters
- **upper_bound** – array of potential upper bounds for the parameters

check_identifiability() → pandas.core.frame.DataFrame

Use ensemble mean and standard deviation to assess (in a rudimentary way) whether or not parameters are identifiable. Returns a dataframe with tuples, which specify whether or not the lower and the upper bounds are violated.

Returns DataFrame indicating parameter identifiability based on mean plus/minus standard deviations and parameter bounds

Return type parameter_identifiability

compute_summary(*percentiles_list: Sequence[int] = (5, 20, 80, 95)*)

This function computes the mean, the median, the standard deviation and possibly percentiles for the parameters of the ensemble. Those summary results are added as a data member to the EnsemblePrediction object.

Parameters **percentiles_list** – List or tuple of percent numbers for the percentiles

Returns Dict with mean, std, median, and percentiles of parameter vectors

Return type summary

static from_sample(*result: pypesto.result.Result*, *remove_burn_in: bool = True*, *chain_slice: Optional[slice] = None*, ***kwargs*)

Construct an ensemble from a sample.

Parameters

- **result** – A pyPESTO result that contains a sample result.
- **remove_burn_in** – Exclude parameter vectors from the ensemble if they are in the "burn-in".
- **chain_slice** – Subset the chain with a slice. Any "burn-in" removal occurs first.

Returns

Return type The ensemble.

predict(*predictor: Callable*, *prediction_id: Optional[str] = None*, *sensi_orders: Tuple = (0)*, *default_value: Optional[float] = None*, *mode: str = 'mode_fun'*, *engine: Optional[pypesto.engine.base.Engine] = None*, *progress_bar: bool = True*) → pypesto.ensemble.ensemble.EnsemblePrediction

Convenience function to run predictions for a full ensemble: User needs to hand over a predictor function and settings, then all results are grouped as EnsemblePrediction for the whole ensemble

Parameters

- **predictor** – Prediction function, e.g., an AmiciPredictor
- **prediction_id** – Identifier for the predictions
- **sensi_orders** – Specifies which sensitivities to compute, e.g. (0,1) -> fval, grad

- **default_value** – If parameters are needed in the mapping, which are not found in the parameter source, it can make sense to fill them up with this default value (e.g. *np.nan*) in some cases (to be used with caution though).
- **mode** – Whether to compute function values or residuals.
- **engine** – Parallelization engine. Defaults to sequential execution on a *SingleCoreEngine*.
- **progress_bar** – Whether to display a progress bar.

Returns

Return type The prediction of the ensemble.

class pypesto.ensemble.**EnsemblePrediction**(*predictor: Callable[[Sequence], pypesto.predict.result.PredictionResult], prediction_id: Optional[str] = None, prediction_results: Optional[Sequence[pypesto.predict.result.PredictionResult]] = None, lower_bound: Optional[Sequence[numpy.ndarray]] = None, upper_bound: Optional[Sequence[numpy.ndarray]] = None*)

Bases: object

A ensemble prediction consists of an ensemble, i.e., a set of parameter vectors and their identifiers such as a sample, and a prediction function. It can be attached to a ensemble-type object

__init__(*predictor: Callable[[Sequence], pypesto.predict.result.PredictionResult], prediction_id: Optional[str] = None, prediction_results: Optional[Sequence[pypesto.predict.result.PredictionResult]] = None, lower_bound: Optional[Sequence[numpy.ndarray]] = None, upper_bound: Optional[Sequence[numpy.ndarray]] = None*)

Constructor.

Parameters

- **predictor** – Prediction function, e.g., an AmiciPredictor, which takes a parameter vector as input and outputs a PredictionResult object
- **prediction_id** – Identifier for the predictions
- **prediction_results** – List of Prediction results
- **lower_bound** – Array of potential lower bounds for the predictions, should have the same shape as the output of the predictions, i.e., a list of numpy array (one list entry per condition), with the arrays having the shape of n_timepoints x n_outputs for each condition.
- **upper_bound** – array of potential upper bounds for the parameters

compute_summary(*percentiles_list: Sequence[int] = (5, 20, 80, 95)*) → Dict

Compute the mean, the median, the standard deviation and possibly percentiles from the ensemble prediction results. Those summary results are added as a data member to the EnsemblePrediction object.

Parameters **percentiles_list** – List or tuple of percent numbers for the percentiles

Returns dictionary of predictions results with the keys mean, std, median, percentiles, ...

Return type summary

condense_to_arrays()
This functions reshapes the predictions results to an array and adds them as a member to the EnsemblePrediction objects. It's meant to be used only if all conditions of a prediction have the same observables, as this is often the case for large-scale data sets taken from online databases or similar.

class pypesto.ensemble.**EnsembleType**(*value*)
Bases: enum.Enum

An enumeration.

ensemble = 1

sample = 2

unprocessed_chain = 3

class pypesto.ensemble.**Enum**(*value*)
Bases: object

Generic enumeration.

Derive from this class to define new enumerations.

name
The name of the Enum member.

value
The value of the Enum member.

pypesto.ensemble.**get_covariance_matrix_parameters**(*ens: pypesto.ensemble.ensemble.Ensemble*) → numpy.ndarray
Compute the covariance of ensemble parameters.

Parameters **ens** – Ensemble object containing a set of parameter vectors

Returns covariance matrix of ensemble parameters

Return type covariance_matrix

pypesto.ensemble.**get_covariance_matrix_predictions**(*ens: Union[pypesto.ensemble.ensemble.Ensemble, pypesto.ensemble.ensemble.EnsemblePrediction], prediction_index: int = 0*) → numpy.ndarray
Compute the covariance of ensemble predictions.

Parameters

- **ens** – Ensemble object containing a set of parameter vectors and a set of predictions or EnsemblePrediction object containing only predictions
- **prediction_index** – index telling which prediction from the list should be analyzed

Returns covariance matrix of ensemble predictions

Return type covariance_matrix

pypesto.ensemble.**get_pca_representation_parameters**(*ens: pypesto.ensemble.ensemble.Ensemble, n_components: int = 2, rescale_data: bool = True, rescaler: Optional[Callable] = None*) → Tuple
Compute the representation with reduced dimensionality via principal component analysis (with a given number

of principal components) of the parameter ensemble.

Parameters

- **ens** – Ensemble objects containing a set of parameter vectors
- **n_components** – number of components for the dimension reduction
- **rescale_data** – flag indicating whether the principal components should be rescaled using a rescaler function (e.g., an arcsinh function)
- **rescaler** – callable function to rescale the output of the PCA (defaults to numpy.arcsinh)

Returns

- *principal_components* – principal components of the parameter vector ensemble
- *pca_object* – returned fitted pca object from sklearn.decomposition.PCA()

pypesto.ensemble.**get_pca_representation_predictions**(*ens: Union[pypesto.ensemble.ensemble.Ensemble, pypesto.ensemble.ensemble.EnsemblePrediction]*, *prediction_index: int = 0*, *n_components: int = 2*, *rescale_data: bool = True*, *rescaler: Optional[Callable] = None*) → Tuple

Compute the representation with reduced dimensionality via principal component analysis (with a given number of principal components) of the ensemble prediction.

Parameters

- **ens** – Ensemble objects containing a set of parameter vectors and a set of predictions or EnsemblePrediction object containing only predictions
- **prediction_index** – index telling which prediction from the list should be analyzed
- **n_components** – number of components for the dimension reduction
- **rescale_data** – flag indicating whether the principal components should be rescaled using a rescaler function (e.g., an arcsinh function)
- **rescaler** – callable function to rescale the output of the PCA (defaults to numpy.arcsinh)

Returns

- *principal_components* – principal components of the parameter vector ensemble
- *pca_object* – returned fitted pca object from sklearn.decomposition.PCA()

pypesto.ensemble.**get_percentile_label**(*percentile: Union[float, int, str]*) → str

Convert a percentile to a label.

Labels for percentiles are used at different locations (e.g. ensemble prediction code, and visualization code). This method ensures that the same percentile is labeled identically everywhere.

The percentile is rounded to two decimal places in the label representation if it is specified to more decimal places. This is for readability in plotting routines, and to avoid float to string conversion issues related to float precision.

Parameters **percentile** – The percentile value that will be used to generate a label.

Returns

Return type The label of the (possibly rounded) percentile.

pypesto.ensemble.**get_spectral_decomposition_lowlevel**(*matrix: numpy.ndarray*, *normalize: bool = False*, *only_separable_directions: bool = False*, *cutoff_absolute_separable: float = 1e-16*, *cutoff_relative_separable: float = 1e-16*, *only_identifiable_directions: bool = False*, *cutoff_absolute_identifiable: float = 1e-16*, *cutoff_relative_identifiable: float = 1e-16*) → Tuple[numpy.ndarray, numpy.ndarray]

Compute the spectral decomposition of ensemble parameters or predictions.

Parameters

- **matrix** – symmetric matrix (typically a covariance matrix) of parameters or predictions
- **normalize** – flag indicating whether the returned Eigenvalues should be normalized with respect to the largest Eigenvalue
- **only_separable_directions** – return only separable directions according to cutoff_[absolute/relative]_separable
- **cutoff_absolute_separable** – Consider only eigenvalues of the covariance matrix above this cutoff (only applied when only_separable_directions is True)
- **cutoff_relative_separable** – Consider only eigenvalues of the covariance matrix above this cutoff, when rescaled with the largest eigenvalue (only applied when only_separable_directions is True)
- **only_identifiable_directions** – return only identifiable directions according to cutoff_[absolute/relative]_identifiable
- **cutoff_absolute_identifiable** – Consider only low eigenvalues of the covariance matrix with inverses above of this cutoff (only applied when only_identifiable_directions is True)
- **cutoff_relative_identifiable** – Consider only low eigenvalues of the covariance matrix when rescaled with the largest eigenvalue with inverses above of this cutoff (only applied when only_identifiable_directions is True)

Returns

- *eigenvalues* – Eigenvalues of the covariance matrix
- *eigenvectors* – Eigenvectors of the covariance matrix

pypesto.ensemble.**get_spectral_decomposition_parameters**(*ens: pypesto.ensemble.ensemble.Ensemble*, *normalize: bool = False*, *only_separable_directions: bool = False*, *cutoff_absolute_separable: float = 1e-16*, *cutoff_relative_separable: float = 1e-16*, *only_identifiable_directions: bool = False*, *cutoff_absolute_identifiable: float = 1e-16*, *cutoff_relative_identifiable: float = 1e-16*) → Tuple[numpy.ndarray, numpy.ndarray]

Compute the spectral decomposition of ensemble parameters.

Parameters

- **ens** – Ensemble object containing a set of parameter vectors
- **normalize** – flag indicating whether the returned Eigenvalues should be normalized with respect to the largest Eigenvalue
- **only_separable_directions** – return only separable directions according to cutoff_[absolute/relative]_separable
- **cutoff_absolute_separable** – Consider only eigenvalues of the covariance matrix above this cutoff (only applied when only_separable_directions is True)
- **cutoff_relative_separable** – Consider only eigenvalues of the covariance matrix above this cutoff, when rescaled with the largest eigenvalue (only applied when only_separable_directions is True)
- **only_identifiable_directions** – return only identifiable directions according to cutoff_[absolute/relative]_identifiable
- **cutoff_absolute_identifiable** – Consider only low eigenvalues of the covariance matrix with inverses above of this cutoff (only applied when only_identifiable_directions is True)
- **cutoff_relative_identifiable** – Consider only low eigenvalues of the covariance matrix when rescaled with the largest eigenvalue with inverses above of this cutoff (only applied when only_identifiable_directions is True)

Returns

- *eigenvalues* – Eigenvalues of the covariance matrix
- *eigenvectors* – Eigenvectors of the covariance matrix

pypesto.ensemble.**get_spectral_decomposition_predictions**(*ens: pypesto.ensemble.ensemble.Ensemble*, *normalize: bool = False*, *only_separable_directions: bool = False*, *cutoff_absolute_separable: float = 1e-16*, *cutoff_relative_separable: float = 1e-16*, *only_identifiable_directions: bool = False*, *cutoff_absolute_identifiable: float = 1e-16*, *cutoff_relative_identifiable: float = 1e-16*) → Tuple[numpy.ndarray, numpy.ndarray]

Compute the spectral decomposition of ensemble predictions.

Parameters

- **ens** – Ensemble object containing a set of parameter vectors and a set of predictions or EnsemblePrediction object containing only predictions
- **normalize** – flag indicating whether the returned Eigenvalues should be normalized with respect to the largest Eigenvalue
- **only_separable_directions** – return only separable directions according to cutoff_[absolute/relative]_separable
- **cutoff_absolute_separable** – Consider only eigenvalues of the covariance matrix above this cutoff (only applied when only_separable_directions is True)
- **cutoff_relative_separable** – Consider only eigenvalues of the covariance matrix above this cutoff, when rescaled with the largest eigenvalue (only applied when only_separable_directions is True)
- **only_identifiable_directions** – return only identifiable directions according to cutoff_[absolute/relative]_identifiable
- **cutoff_absolute_identifiable** – Consider only low eigenvalues of the covariance matrix with inverses above of this cutoff (only applied when only_identifiable_directions is True)
- **cutoff_relative_identifiable** – Consider only low eigenvalues of the covariance matrix when rescaled with the largest eigenvalue with inverses above of this cutoff (only applied when only_identifiable_directions is True)

Returns

- *eigenvalues* – Eigenvalues of the covariance matrix
- *eigenvectors* – Eigenvectors of the covariance matrix

pypesto.ensemble.**get_umap_representation_parameters**(*ens: pypesto.ensemble.ensemble.Ensemble*, *n_components: int = 2*, *normalize_data: bool = False*, ***kwargs*) → Tuple

Compute the representation with reduced dimensionality via umap (with a given number of umap components) of the parameter ensemble. Allows to pass on additional keyword arguments to the umap routine.

Parameters

- **ens** – Ensemble objects containing a set of parameter vectors
- **n_components** – number of components for the dimension reduction
- **normalize_data** – flag indicating whether the parameter ensemble should be rescaled with mean and standard deviation

Returns

- *umap_components* – first components of the umap embedding
- *umap_object* – returned fitted umap object from umap.UMAP()

pypesto.ensemble.**get_umap_representation_predictions**(*ens: Union[pypesto.ensemble.ensemble.Ensemble, pypesto.ensemble.ensemble.EnsemblePrediction]*, *prediction_index: int = 0*, *n_components: int = 2*, *normalize_data: bool = False*, ***kwargs*) → Tuple

Compute the representation with reduced dimensionality via umap (with a given number of umap components) of the ensemble predictions. Allows to pass on additional keyword arguments to the umap routine.

Parameters

- **ens** – Ensemble objects containing a set of parameter vectors and a set of predictions or EnsemblePrediction object containing only predictions
- **prediction_index** – index telling which prediction from the list should be analyzed
- **n_components** – number of components for the dimension reduction
- **normalize_data** – flag indicating whether the parameter ensemble should be rescaled with mean and standard deviation

Returns

- *umap_components* – first components of the umap embedding
- *umap_object* – returned fitted umap object from umap.UMAP()

pypesto.ensemble.**read_from_csv**(*path: str*, *sep: str = '\t'*, *index_col: int = 0*, *headline_parser: Optional[Callable] = None*, *ensemble_type: Optional[pypesto.ensemble.constants.EnsembleType] = None*, *lower_bound: Optional[numpy.ndarray] = None*, *upper_bound: Optional[numpy.ndarray] = None*)

function for creating an ensemble from a csv file

Parameters

- **path** – path to csv file to read in parameter ensemble
- **sep** – separator in csv file
- **index_col** – index column in csv file
- **headline_parser** – A function which reads in the headline of the csv file and converts it into vector_tags (see constructor of Ensemble for more details)
- **ensemble_type** – Ensemble type: representative sample or random ensemble
- **lower_bound** – array of potential lower bounds for the parameters
- **upper_bound** – array of potential upper bounds for the parameters

Returns Ensemble object of parameter vectors

Return type result

pypesto.ensemble.**read_from_df**(*dataframe: pandas.core.frame.DataFrame*, *headline_parser: Optional[Callable] = None*, *ensemble_type: Optional[pypesto.ensemble.constants.EnsembleType] = None*, *lower_bound: Optional[numpy.ndarray] = None*, *upper_bound: Optional[numpy.ndarray] = None*)

function for creating an ensemble from a csv file

Parameters

- **dataframe** – pandas.DataFrame to read in parameter ensemble
- **headline_parser** – A function which reads in the headline of the csv file and converts it into vector_tags (see constructor of Ensemble for more details)
- **ensemble_type** – Ensemble type: representative sample or random ensemble
- **lower_bound** – array of potential lower bounds for the parameters
- **upper_bound** – array of potential upper bounds for the parameters

Returns Ensemble object of parameter vectors

Return type result

pypesto.ensemble.**write_ensemble_prediction_to_h5**(*ensemble_prediction: pypesto.ensemble.ensemble.EnsemblePrediction*, *output_file: str*, *base_path: Optional[str] = None*)

CHAPTER

FIVE

CONTRIBUTE

5.1 Workflow

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

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

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

5.2 Environment

If you contribute to the development of pyPESTO, install developer requirements via:

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

This installs the tools described below.

5.2.1 Pre-commit hooks

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

```
pre-commit install
```

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

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

5.2.2 Tox

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

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

with optional `-e` options specifying the environments to run, see `tox.ini` for details.

5.3 GitHub Actions

For automatic continuous integration testing, we use GitHub Actions. All tests are run there on pull requests and required to pass. The configuration is specified in `.github/workflows/ci.yml`.

5.4 Documentation

To make pyPESTO easily usable, we try to provide good documentation, including code annotation and usage examples. The documentation is hosted at pypesto.readthedocs.io and updated automatically on merges to the main branches. To create the documentation locally, first install the requirements via:

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

and then compile the documentation via:

```
cd doc
make html
```

The documentation is then under `doc/_build`.

Alternatively, the documentation can be compiled and tested via a single line:

```
tox -e doc
```

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

For docstrings, we follow the numpy docstring standard. Check here for a detailed explanation.

5.5 Unit tests

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

```
tox -e base
```

with `base` covering basic tests, but some parts (`optimize`, `petab`, `...`) being in separate subfolders. This boils mostly down to e.g.:

```
pytest test/base
```

You can also run only specific tests.

Unit tests can be written with pytest or unittest.

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

5.6 PEP8

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

```
tox -e flake8
```

CHAPTER

SIX

DEPLOY

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

6.1 Versions

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

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

as suggested by the Python packaging guide.

6.2 Create a new release

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

6.2.1 Merge into main

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

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

6.2.2 Create a release on GitHub

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

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

6.3 Upload to PyPI

The upload to the python package index PyPI has been automatized via GitHub Actions and is triggered whenever a new release tag is published.

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

```
python setup.py bdist_wheel
```

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

This archive is uploaded using twine:

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

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

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

CHAPTER

SEVEN

RELEASE NOTES

7.1 0.2 series

7.1.1 0.2.5 (2021-05-04)

- **Objectives:**
  - New Aesara objectve (#623, #629, #635)
- **Sampling:**
  - New Emcee sampler (#606)
  - Fix compatibility to new Theano version (#650)
- **Storage:**
  - Improve hdf5 storage documentation (#612)
  - Hdf5 history for MultiProcessEngine (#650)
  - Minor fixes (#637, #638, #645, #649)
- **Visualization:**
  - Fix bounds of parameter plots (#601)
  - Fix waterfall plots with multiple results (#611)
- **CI:**
  - Move CI tests on GitHub Actions to python 3.9 (#598)
  - Add issue template (#604)
  - Update BionetGen Link (#630)
  - Introduce project.toml (#634)
- **General:**
  - Introduce progress bar for optimization, profiles and ensembles (#641)
  - Extend gradient checking functionality (#644)
- **Minor fixes:**
  - Fix installation of ipopt (#599)
  - Fix Zenodo link (#601)
  - Fix duplicates in documentation (#603)

  - Fix least squares optimizers (#617 #631 #632)
  - Fix trust region options (#616)
  - Fix slicing for new AMICI release (#621)
  - Refactor and document latin hypercube sampling (#647)
  - Fix missing SBML name in PEtab import (#648)

7.1.2 0.2.4 (2021-03-12)

- **Ensembles/Sampling:**
  - General ensemble analysis, visualization, storage (#557, #565, #568)
  - Calculate and plot MCMC parameter and prediction CIs via ensemble definition, parallelize ensemble predictions (#490)
- **Optimization:**
  - New optimizer: SciPy Differential Evolution (#543)
  - Set fides default to hybrid (#578)
- **AMICI:**
  - Make *guess_steadystate* less restrictive (#561) and have a more intuitive default behavior (#562, #582)
  - Customize time points (#490)
- **Storage:**
  - Save HDF5 history with SingleCoreEngine (#564)
  - Add read/write function for whole results (#589)
- **Engines:**
  - MPI based distributed parallelization (#542)
- **Visualization:**
  - Speed up waterfall plots by resizing scales only once (#577)
  - Change waterfall default offset to 1 - minimum (#593)
- **CI:**
  - Move GHA CI tests to pull request level for better cooperability (#574)
  - Streamline test environments using tox and pre-commit hooks (#579)
  - Test profile and sampling storage (#585)
  - Update for Ubuntu 20.04, add rerun on failure (#587)
- Minor fixes (release notes #558, nlop tests #559, close files #495, visualization #554, deployment #560, flakiness #570, aggregated deepcopy #572, respect user-provided offsets #576, update to SWIG 4 #591, check overwrite in profile writing #566)

7.1.3 0.2.3 (2021-01-18)

- **New optimizers:**
  - FIDES (#506, #503 # 500)
  - NLopt (#493)
- **Extended PEtab support:**
  - PySB import (#437)
  - Support of PEtab's initializationPriors (#535)
  - Support of prior parameterScale{Normal,Laplace} (#520)
  - Example notebook for synthetic data generation (#482)
- **General new and improved functionality:**
  - Predictions (#544)
  - Move tests to GitHub Actions (#524)
  - Parallelize profile calculation (#532)
  - Save *x_guesses* in *pypesto.problem* (#494)
  - Improved finite difference gradients (#464)
  - Support of unconstrained optimization (#519)
  - Additional NaN check for fval, grad and hessian (#521)
  - Add sanity checks for optimizer bounds (#516)
- **Improvements in storage:**
  - Fix hdf5 export of optimizer history (#536)
  - Fix reading *x_names* from hdf5 history (#528)
  - Storage does not save empty arrays (#489)
  - hdf5 storage sampling (#546)
  - hdf5 storage parameter profiles (#546)
- **Improvements in the visualization routines:**
  - Plot parameter values as histogram (#485)
  - Fix y axis limits in waterfall plots (#503)
  - Fix color scheme in visualization (#498)
  - Improved visualization of optimization results (#486)
- Several small bug fixes (#547, #541, #538, #533, #512, #508)

7.1.4 0.2.2 (2020-10-05)

- New optimizer: CMA-ES (#457)
- New plot: Optimizer convergence summary (#446)
- **Fixes in visualization:**
  - Type checks for reference points (#460)
  - y_limits in waterfall plots with multiple results (#475)
- Support of new amici release (#469)
- **Multiple fixes in optimization code:**
  - Remove unused argument for dlib optimizer (#466)
  - Add check for installation of ipopt (#470)
  - Add maxiter as default option of dlib (#474)
- Numpy based subindexing in amici_util (#462)
- Check amici/PEtab installation (#477)

7.1.5 0.2.1 (2020-09-07)

- Example Notebook for prior functionality (#438)
- Changed parameter indexing in profiling routines (#419)
- Basic sanity checking for parameter fixing (#420)
- **Bug fixes in:**
  - Displaying of multi start optimization (#430)
  - AMICI error output (#428)
  - Axes scaling/limits in waterfall plots (#441)
  - Priors (PEtab import, error handling) (#448, #452, #454)
- Improved sampling diagnostics (e.g. effective samples size) (#426)
- Improvements and bug fixes in parameter plots (#425)

7.1.6 0.2.0 (2020-06-17)

Major:

- Modularize import, to import optimization, sampling and profiling separately (#413)

Minor:

- **Bug fixes in**
  - sampling (#412)
  - visualization (#405)
  - PEtab import (#403)
  - Hessian computation (#390)

- Improve hdf5 error output (#409)
- Outlaw large new files in GitHub commits (#388)

7.2 0.1 series

7.2.1 0.1.0 (2020-06-17)

Objective

- Write solver settings to stream to enable serialization for distributed systems (#308)
- **Refactor objective function (#347)**
  - Removes necessity for all of the nasty binding/undbinding in AmiciObjective
  - Substantially reduces the complexity of the AggregatedObjective class
  - Aggregation of functions with inconsistent sensi_order/mode support
  - Introduce ObjectiveBase as an abstract Objective class
  - Introduce FunctionObjective for objectives from functions
- Implement priors with gradients, integrate with PEtab (#357)
- Fix minus sign in AmiciObjective.get_error_output (#361)
- Implement a prior class, derivatives for standard models, interface with PEtab (#357)
- Use *amici.import_model_module* to resolve module loading failure (#384)

Problem

- Tidy up problem vectors using properties (#393)

Optimization

- Interface IpOpt optimizer (#373)

Profiles

- Tidy up profiles (#356)
- Refactor profiles; add locally approximated profiles (#369)
- Fix profiling and visualization with fixed parameters (#393)

Sampling

- Geweke test for sampling convergence (#339)
- Implement basic Pymc3 sampler (#351)
- Make theano for pymc3 an optional dependency (allows using pypesto without pymc3) (#356)
- Progress bar for MCMC sampling (#366)
- Fix Geweke test crash for small sample sizes (#376)
- In parallel tempering, allow to only temperate the likelihood, not the prior (#396)

History and storage

- Allow storing results in a pre-filled hdf5 file (#290)
- Various fixes of the history (reduced vs. full parameters, read-in from file, chi2 values) (#315)

- Fix proper dimensions in result for failed start (#317)
- Create required directories before creating hdf5 file (#326)
- Improve storage and docs documentation (#328)
- Fix storing x_free_indices in hdf5 result (#334)
- Fix problem hdf5 return format (#336)
- Implement partial trace extraction, simplify History API (#337)
- Save really all attributes of a Problem to hdf5 (#342)

Visualization

- Customizable xLabels and tight layout for profile plots (#331)
- Fix non-positive bottom ylim on a log-scale axis in waterfall plots (#348)
- Fix "palette list has the wrong number of colors" in sampling plots (#372)
- Allow to plot multiple profiles from one result (#399)

Logging

- Allow easier specification of only logging for submodules (#398)

Tests

- Speed up travis build (#329)
- Update travis test system to latest ubuntu and python 3.8 (#330)
- Additional code quality checks, minor simplifications (#395)

7.3 0.0 series

7.3.1 0.0.13 (2020-05-03)

- Tidy up and speed up tests (#265 and others).
- Basic self-implemented Adaptive Metropolis and Adaptive Parallel Tempering sampling routines (#268).
- Fix namespace sample -> sampling (#275).
- Fix covariance matrix regularization (#275).
- Fix circular dependency *PetabImporter* - *PetabAmiciObjective* via *AmiciObjectBuilder*, *PetabAmiciObjective* becomes obsolete (#274).
- Define *AmiciCalculator* to separate the AMICI call logic (required for hierarchical optimization) (#277).
- Define initialize function for resetting steady states in *AmiciObjective* (#281).
- Fix scipy least squares options (#283).
- Allow failed starts by default (#280).
- Always copy parameter vector in objective to avoid side effects (#291).
- Add Dockerfile (#288).
- Fix header names in CSV history (#299).

Documentation:

- Use imported members in autodoc (#270).
- Enable python syntax highlighting in notebooks (#271).

7.3.2 0.0.12 (2020-04-06)

- Add typehints to global functions and classes.
- Add *PetabImporter.rdatas_to_simulation_df* function (all #235).
- Adapt y scale in waterfall plot if convergence was too good (#236).
- Clarify that *Objective* is of type negative log-posterior, for minimization (#243).
- Tidy up *AmiciObjective.parameter_mapping* as implemented in AMICI now (#247).
- Add *MultiThreadEngine* implementing multi-threading aside the *MultiProcessEngine* implementing multi-processing (#254).
- Fix copying and pickling of *AmiciObjective* (#252, #257).
- Remove circular dependence history-objective (#254).
- Fix problem of visualizing results with failed starts (#249).
- Rework history: make thread-safe, use factory methods, make context-specific (#256).
- Improve PEtab usage example (#258).
- Define history base contract, enabling different backends (#260).
- Store optimization results to HDF5 (#261).
- Simplify tests (#263).

Breaking changes:

- *HistoryOptions* passed to *pypesto.minimize* instead of *Objective* (#256).
- *GlobalOptimizer* renamed to *PyswarmOptimizer* (#235).

7.3.3 0.0.11 (2020-03-17)

- Rewrite AmiciObjective and PetabAmiciObjective simulation routine to directly use amici.petab_objective routines (#209, #219, #225).
- Implement petab test suite checks (#228).
- Various error fixes, in particular regarding PEtab and visualization.
- Improve trace structure.
- Fix conversion between fval and chi2, fix FIM (all #223).

7.3.4 0.0.10 (2019-12-04)

- Only compute FIM when sensitivities are available (#194).
- Fix documentation build (#197).
- Add support for pyswarm optimizer (#198).
- Run travis tests for documentation and notebooks only on pull requests (#199).

7.3.5 0.0.9 (2019-10-11)

- Update to AMICI 0.10.13, fix API changes (#185).
- Start using PEtab import from AMICI to be able to import constant species (#184, #185)
- Require PEtab>=0.0.0a16 (#183)

7.3.6 0.0.8 (2019-09-01)

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

7.3.7 0.0.7 (2019-03-21)

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

7.3.8 0.0.6 (2019-03-13)

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

7.3.9 0.0.5 (2019-03-11)

- Introduce AggregatedObjective to use multiple objectives at once.
- Estimate steady state in AmiciObjective.
- Check amici model build version in PetabImporter.
- Use Amici multithreading in AmiciObjective.
- Allow to sort multistarts by initial value.
- Show usage of visualization routines in notebooks.
- Various fixes, in particular to visualization.

7.3.10 0.0.4 (2019-02-25)

- Implement multi process parallelization engine for optimization.
- Introduce PrePostProcessor to more reliably handle pre- and post-processing.
- Fix problems with simulating for multiple conditions.
- Add more visualization routines and options for those (colors, reference points, plotting of lists of result obejcts)

7.3.11 0.0.3 (2019-01-30)

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

7.3.12 0.0.2 (2018-10-18)

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

7.3.13 0.0.1 (2018-07-25)

- Basic framework and implementation of the optimization

CHAPTER

EIGHT

AUTHORS

This package is mainly developed by:

- Yannik Schälte
- Jakob Vanhoefer
- Fabian Fröhlich
- Paul Stapor

with major contributions by:

- Dantong Wang
- Daniel Weindl
- Polina Lakrisenko
- Elba Raimúndez
- Leonard Schmiester
- Dilan Pathirana
- Paul Jonas Jost
- Lorenzo Contento
- Erika Dudkin
- Simon Merkt
- Carolin Loos
- Philipp Städter

CHAPTER

NINE

CONTACT

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

- Yannik Schälte: yannik.schaelte@gmail.com

CHAPTER

TEN

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

ELEVEN

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

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

A

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X