pyPESTO Documentation

Release 0.0.2

The pyPESTO developers

Jan 31, 2019

User's guide

Version: 0.0.2
Source code: https://github.com/icb-dcm/pypesto

CHAPTER 1

Install and upgrade

1.1 Requirements

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

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

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

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

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

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

1.2 Install from PIP

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

```
pip3 install pypesto
```

1.3 Install from GIT

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

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

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

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

and then install from the local repository via:

```
cd pypesto
pip3 install .
```

1.4 Upgrade

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

1.5 Install optional packages

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

  ```
  pip3 install amici
  ```

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

  ```
  pip3 install dlib
  ```

CHAPTER 2

Examples

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

2.1 Rosenbrock banana

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

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

%matplotlib inline
```

2.1.1 Define the objective and problem

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

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

dim_full = 10
lb = -2 * np.ones((dim_full, 1))
ub = 2 * np.ones((dim_full, 1))
```

(continues on next page)

(continued from previous page)

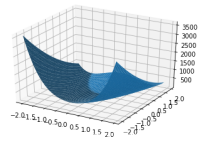
```
problem1 = pypesto.Problem(objective=objective1, lb=lb, ub=ub)
problem2 = pypesto.Problem(objective=objective2, lb=lb, ub=ub)
```

2.1.2 Illustration

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

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X=x, Y=y, Z=z)
```

[3]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7fd15ad50278>
```



2.1.3 Run optimization

[4]:
```
optimizer = pypesto.ScipyOptimizer()
n_starts = 20

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

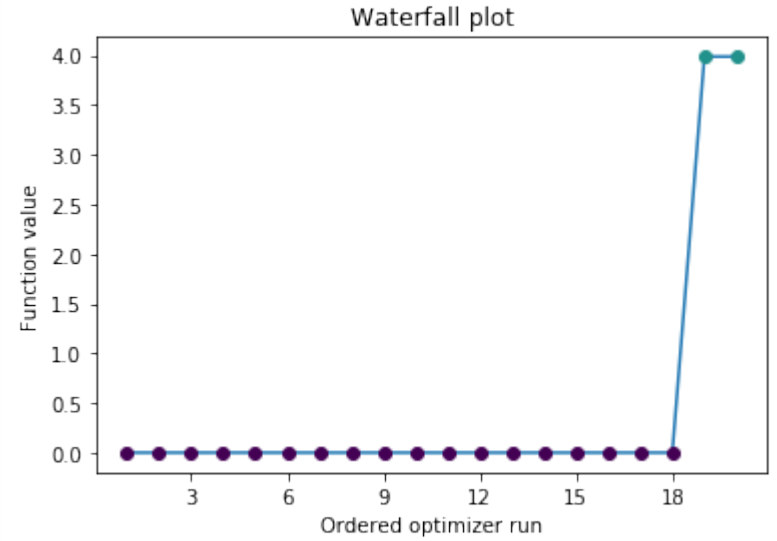
2.1.4 Visualize and analyze results

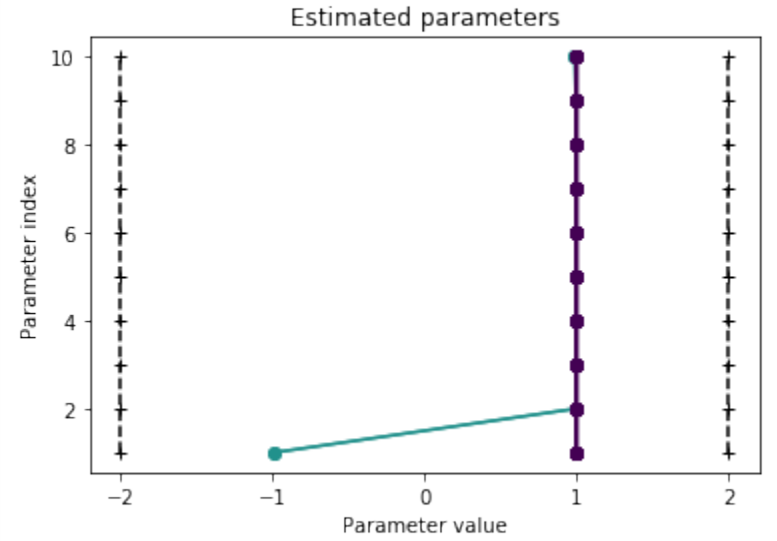
pypesto offers easy-to-use visualization routines:

[5]:
```
import pypesto.visualize

pypesto.visualize.waterfall(result1)
pypesto.visualize.parameters(result1)
```

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd15a929d68>
```





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

[6]:
```
result1.optimize_result.as_dataframe(['fval', 'n_fval', 'n_grad', 'n_hess', 'n_res',
↪'n_sres', 'time'])
```

[6]:

	fval	n_fval	n_grad	n_hess	n_res	n_sres	time
0	2.967459e-12	83	83	0	0	0	0.015485
1	4.049190e-12	87	87	0	0	0	0.014637
2	7.886079e-12	99	99	0	0	0	0.038658
3	9.738153e-12	78	78	0	0	0	0.010974
4	1.327963e-11	72	72	0	0	0	0.011126
5	1.548685e-11	60	60	0	0	0	0.008633
6	2.444331e-11	87	87	0	0	0	0.016484
7	4.726193e-11	103	103	0	0	0	0.014987
8	7.583920e-11	72	72	0	0	0	0.011051
9	8.009222e-11	73	73	0	0	0	0.011766
10	8.351843e-11	79	79	0	0	0	0.023473
11	8.571021e-11	86	86	0	0	0	0.016769

(continues on next page)

(continued from previous page)

```
12  9.786016e-11      77      77       0       0       0  0.010761
13  1.190891e-10      95      95       0       0       0  0.017382
14  1.233392e-10      73      73       0       0       0  0.010384
15  1.463236e-10      87      87       0       0       0  0.012187
16  1.952418e-10      63      63       0       0       0  0.010246
17  3.268263e-10      81      81       0       0       0  0.011533
18  3.986579e+00      71      71       0       0       0  0.010817
19  3.986579e+00      90      90       0       0       0  0.016129
```

2.2 Conversion reaction

[1]:
```
import amici
import amici.plotting
import pypesto
```

2.2.1 Compile AMICI model

[2]:
```
import importlib
import os
import sys
import numpy as np

# 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

# import sbml model, complile and generate amici module
sbml_importer = amici.SbmlImporter(sbml_file)
sbml_importer.sbml2amici(model_name,
                         model_output_dir,
                         verbose=False)
```

```
libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression refers to a compartment, species or parameter, it is necessary to know␣
↪the units of the object to establish unit consistency. In models where the units of␣
↪an object have not been declared, libSBML does not yet have the functionality to␣
↪accurately verify the consistency of the units in mathematical expressions␣
↪referring to that object.
 The units of the <parameter> 'R1_k2' cannot be fully checked. Unit consistency␣
↪reported as either no errors or further unit errors related to this object may not␣
↪be accurate.

libSBML Warning (SBML unit consistency): If the attribute 'units' on a given␣
↪Parameter object has not been set, then the unit of measurement associated with␣
↪that parameter's value is undefined.
Reference: L3V1 Section 4.7.3
 The <parameter> with id 'R1_k2' does not have a 'units' attribute.
```

(continues on next page)

(continued from previous page)

```
libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression refers to a compartment, species or parameter, it is necessary to know␣
↪the units of the object to establish unit consistency. In models where the units of␣
↪an object have not been declared, libSBML does not yet have the functionality to␣
↪accurately verify the consistency of the units in mathematical expressions␣
↪referring to that object.
 The units of the <parameter> 'R1_k1' cannot be fully checked. Unit consistency␣
↪reported as either no errors or further unit errors related to this object may not␣
↪be accurate.

libSBML Warning (SBML unit consistency): If the attribute 'units' on a given␣
↪Parameter object has not been set, then the unit of measurement associated with␣
↪that parameter's value is undefined.
Reference: L3V1 Section 4.7.3
 The <parameter> with id 'R1_k1' does not have a 'units' attribute.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <kineticLaw> <math> expression 'compartment * (R1_k1 * A - R1_k2 *␣
↪B)' cannot be fully checked. Unit consistency reported as either no errors or␣
↪further unit errors related to this object may not be accurate.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'R1_k2' does not have a 'units' attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'R1_k1' does not have a 'units' attribute.
```

2.2.2 Load AMICI model

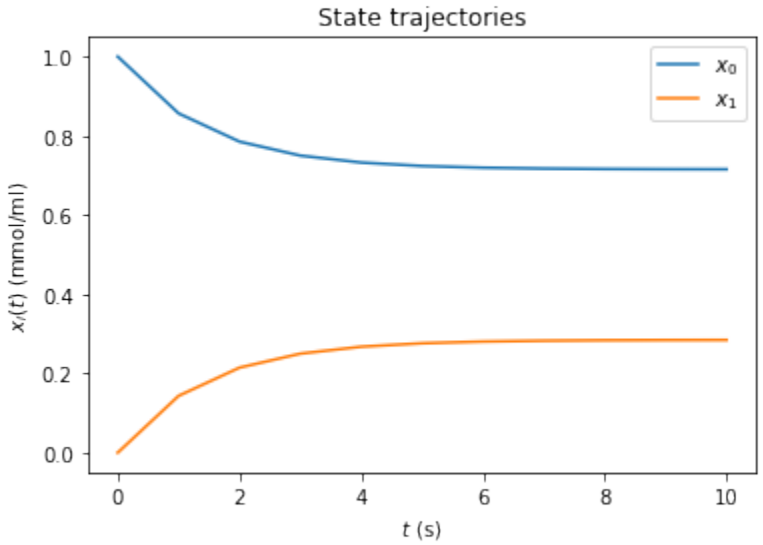
[3]:
```
# load amici module (the usual starting point later for the analysis)
sys.path.insert(0, os.path.abspath(model_output_dir))
model_module = importlib.import_module(model_name)
model = model_module.getModel()
model.requireSensitivitiesForAllParameters()
model.setTimepoints(amici.DoubleVector(np.linspace(0,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

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

(continues on next page)

(continued from previous page)

```
edata = amici.ExpData(rdata, 0.2, 0.0)
```



2.2.3 Optimize

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

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

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

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

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

# maybe lb, ub = inf
# other constraints: kwargs, class pesto.Constraints
# constraints on pams, states, esp. pesto.AmiciConstraints (e.g. pam1 + pam2<= const)
# if optimizer cannot handle -> error
# maybe also scaling / transformation of parameters encoded here

# do the optimization
```

(continues on next page)

(continued from previous page)

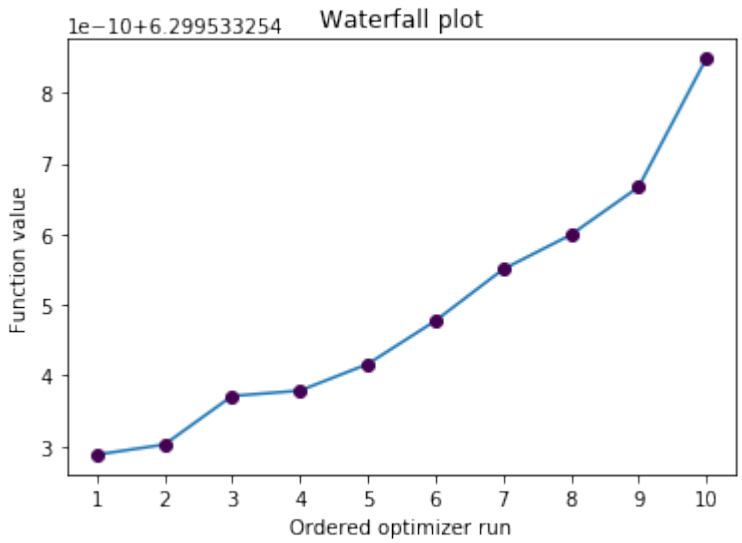
```
result = pypesto.minimize(problem=problem,
                          optimizer=optimizer,
                          n_starts=10)
# optimize is a function since it does not need an internal memory,
# just takes input and returns output in the form of a Result object
# 'result' parameter: e.g. some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

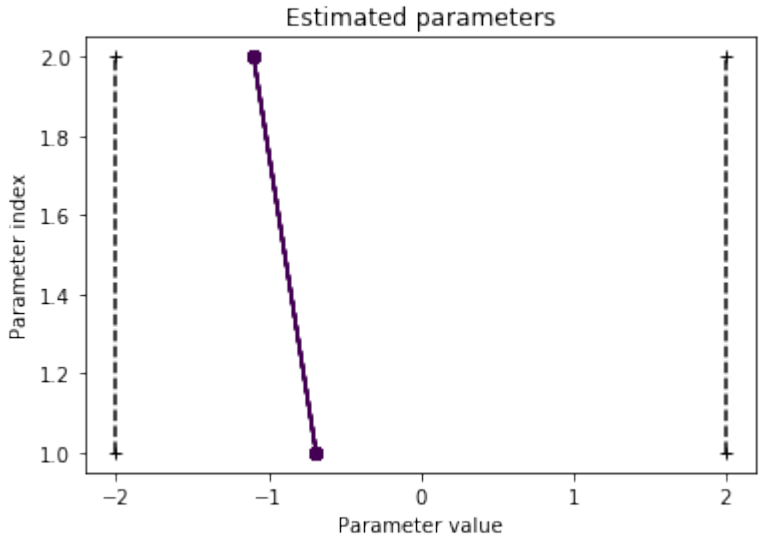
2.2.4 Visualize

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

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

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb9a6218748>
```





2.2.5 Data storage

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

2.2.6 Profiles

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

# profiler = pypesto.Profiler()

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

2.2.7 Sampling

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

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

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

2.3 Fixed parameters

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

2.3.1 Define problem

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

%matplotlib inline
```

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

dim_full = 4
lb = -2 * np.ones((dim_full,1))
ub = 2 * np.ones((dim_full,1))

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

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

2.3.2 Optimize

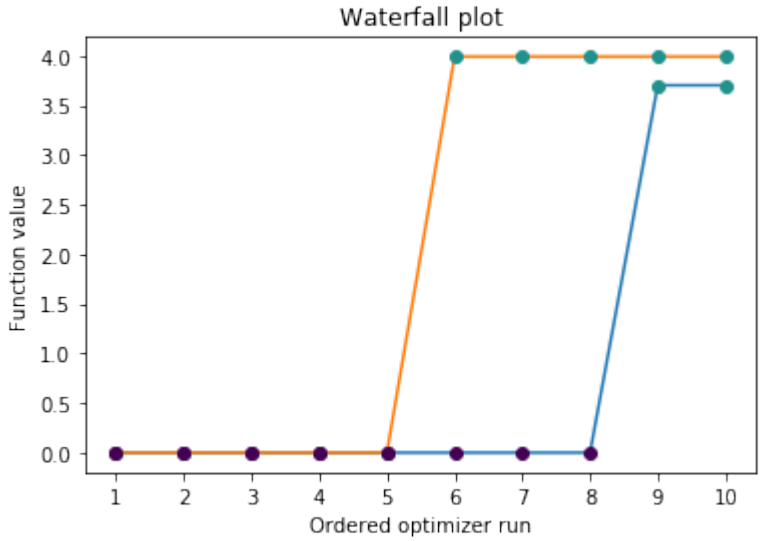
[3]:
```
optimizer = pypesto.ScipyOptimizer()
n_starts = 10

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

2.3.3 Visualize

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

[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ae7415eb8>
```



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

[5]:
```
           fval                                                grad  \
0  3.927779e-15  [1.380082008367687e-06, -8.083573203926321e-08...
1  7.810296e-14  [1.717501444180088e-06, -1.351913230799428e-06...
2  8.211686e-14  [1.2722200973602453e-06, 2.6490319813334526e-0...
3  6.675828e-13  [2.3108400838756338e-05, 5.154092104522114e-06...
4  7.745611e-13  [-7.231244216666451e-06, 1.2069320871269854e-0...
5  2.390803e-12  [-6.369931909390899e-06, 1.2558236812991624e-0...
6  6.429238e-12  [7.387638091332483e-05, -5.3039113081590246e-0...
7  3.157806e-10  [-0.0001412479254942318, -0.000317098067357588...
8  3.701429e+00  [-9.476014543263744e-06, 2.2448530735408312e-0...
9  3.701429e+00  [1.2004479082783348e-05, -0.000416246476298010...

                                                   x
0  [1.0000000049630953, 1.000000006500801, 1.0000...
1  [0.9999999523544961, 0.9999999001770131, 0.999...
2  [1.000000062824105, 1.0000001227817845, 1.0000...
3  [1.0000000997998029, 1.0000001423276184, 1.000...
4  [1.0000001767517774, 1.0000003724654523, 1.000...
5  [0.9999996997057465, 0.9999994138349461, 0.999...
6  [1.00000037493649, 1.0000005670569194, 1.00000...
7  [1.0000011490511884, 1.0000026569683544, 1.000...
8  [-0.7756612066524641, 0.6130963902374005, 0.38...
9  [-0.7756557693277876, 0.6130880696435137, 0.38...
```

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

[6]:
```
           fval                                                grad  \
0  1.034572e-18  [3.7571982860009024e-08, nan, 1.75952958872096...
1  4.515451e-17  [-1.6843336988283393e-07, nan, -2.346168865413...
2  1.788081e-16  [-5.347585997124525e-07, nan, -3.2423277395438...
3  8.127003e-15  [-2.942953443493472e-06, nan, 2.33788272377160...
4  7.530951e-14  [1.096951855484312e-05, nan, 7.631266543328272...
5  3.989975e+00  [8.507160074167075e-07, nan, -1.00001459300255...
6  3.989975e+00  [5.853680722367471e-07, nan, -4.80007196752967...
7  3.989975e+00  [5.872636898551775e-06, nan, -2.24092759348412...
8  3.989975e+00  [-9.41635691198428e-07, nan, -9.52827066229709...
```

(continues on next page)

(continued from previous page)

```
9  3.989975e+00  [3.688754127306737e-05, nan, -1.18522591456615...

                                                        x
0  [1.0000000000468479, 1.0, 1.0000000000175602, ...
1  [0.9999999997899833, 1.0, 0.9999999997658514, ...
2  [0.9999999993332187, 1.0, 0.9999999999676414, ...
3  [0.999999996330482, 1.0, 1.0000000023332163, 1.0]
4  [1.0000000136777036, 1.0, 1.0000000007616034, ...
5  [-0.9949747457536863, 1.0, 0.9999999990019814,...
6  [-0.9949747460895826, 1.0, 0.999999995209509, ...
7  [-0.9949747393965804, 1.0, 0.9999999977635453,...
8  [-0.9949747480225729, 1.0, 0.9999999904907477,...
9  [-0.9949747001356986, 1.0, 0.9999999881713979,...
```

2.4 AMICI Python example “Boehm”

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

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

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

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()])
```

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

```
libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <initialAssignment> <math> expression '207.6 * ratio' cannot be␣
↪fully checked. Unit consistency reported as either no errors or further unit errors␣
↪related to this object may not be accurate.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <initialAssignment> <math> expression '207.6 - 207.6 * ratio'␣
↪cannot be fully checked. Unit consistency reported as either no errors or further␣
↪unit errors related to this object may not be accurate.
```

(continues on next page)

(continued from previous page)

```
libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <assignmentRule> <math> expression '(100 * pApB + 200 * pApA *␣
↪specC17) / (pApB + STAT5A * specC17 + 2 * pApA * specC17)' cannot be fully checked.␣
↪Unit consistency reported as either no errors or further unit errors related to␣
↪this object may not be accurate.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <assignmentRule> <math> expression '1.25e-07 * exp(-1 *␣
↪Epo_degradation_BaF3 * t)' cannot be fully checked. Unit consistency reported as␣
↪either no errors or further unit errors related to this object may not be accurate.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <assignmentRule> <math> expression '-(100 * pApB - 200 * pBpB *␣
↪(specC17 - 1)) / (STAT5B * (specC17 - 1) - pApB + 2 * pBpB * (specC17 - 1))' cannot␣
↪be fully checked. Unit consistency reported as either no errors or further unit␣
↪errors related to this object may not be accurate.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <assignmentRule> <math> expression '(100 * pApB + 100 * STAT5A *␣
↪specC17 + 200 * pApA * specC17) / (2 * pApB + STAT5A * specC17 + 2 * pApA * specC17␣
↪- STAT5B * (specC17 - 1) - 2 * pBpB * (specC17 - 1))' cannot be fully checked. Unit␣
↪consistency reported as either no errors or further unit errors related to this␣
↪object may not be accurate.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <kineticLaw> <math> expression 'cyt * BaF3_Epo * pow(STAT5A, 2) *␣
↪k_phos' cannot be fully checked. Unit consistency reported as either no errors or␣
↪further unit errors related to this object may not be accurate.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <kineticLaw> <math> expression 'cyt * BaF3_Epo * STAT5A * STAT5B *␣
↪k_phos' cannot be fully checked. Unit consistency reported as either no errors or␣
↪further unit errors related to this object may not be accurate.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
```

(continues on next page)

(continued from previous page)

```
 The units of the <kineticLaw> <math> expression 'cyt * BaF3_Epo * pow(STAT5B, 2) *␣
↪k_phos' cannot be fully checked. Unit consistency reported as either no errors or␣
↪further unit errors related to this object may not be accurate.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <kineticLaw> <math> expression 'cyt * k_imp_homo * pApA' cannot be␣
↪fully checked. Unit consistency reported as either no errors or further unit errors␣
↪related to this object may not be accurate.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <kineticLaw> <math> expression 'cyt * k_imp_hetero * pApB' cannot␣
↪be fully checked. Unit consistency reported as either no errors or further unit␣
↪errors related to this object may not be accurate.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <kineticLaw> <math> expression 'cyt * k_imp_homo * pBpB' cannot be␣
↪fully checked. Unit consistency reported as either no errors or further unit errors␣
↪related to this object may not be accurate.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <kineticLaw> <math> expression 'nuc * k_exp_homo * nucpApA' cannot␣
↪be fully checked. Unit consistency reported as either no errors or further unit␣
↪errors related to this object may not be accurate.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <kineticLaw> <math> expression 'nuc * k_exp_hetero * nucpApB'␣
↪cannot be fully checked. Unit consistency reported as either no errors or further␣
↪unit errors related to this object may not be accurate.

libSBML Warning (SBML unit consistency): In situations where a mathematical␣
↪expression contains literal numbers or parameters whose units have not been␣
↪declared, it is not possible to verify accurately the consistency of the units in␣
↪the expression.
 The units of the <kineticLaw> <math> expression 'nuc * k_exp_homo * nucpBpB' cannot␣
↪be fully checked. Unit consistency reported as either no errors or further unit␣
↪errors related to this object may not be accurate.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'Epo_degradation_BaF3' does not have a 'units' attribute.
```

(continues on next page)

(continued from previous page)

```
libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'k_exp_hetero' does not have a 'units' attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'k_exp_homo' does not have a 'units' attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'k_imp_hetero' does not have a 'units' attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'k_imp_homo' does not have a 'units' attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'k_phos' does not have a 'units' attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'ratio' does not have a 'units' attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'sd_pSTAT5A_rel' does not have a 'units' attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'sd_pSTAT5B_rel' does not have a 'units' attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'sd_rSTAT5A_rel' does not have a 'units' attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
```

(continues on next page)

(continued from previous page)

```
 The <parameter> with the id 'specC17' does not have a 'units' attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'observable_pSTAT5A_rel' does not have a 'units'␣
↪attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'observable_pSTAT5B_rel' does not have a 'units'␣
↪attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'observable_rSTAT5A_rel' does not have a 'units'␣
↪attribute.

libSBML Warning (Modeling practice): As a principle of best modeling practice, the␣
↪units of a <parameter> should be declared rather than be left undefined. Doing so␣
↪improves the ability of software to check the consistency of units and helps make␣
↪it easier to detect potential errors in models.
 The <parameter> with the id 'BaF3_Epo' does not have a 'units' attribute.
```

In this example, we want to specify fixed parameters, observables and a 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 parameters (see below).

[5]:
```
# Retrieve model output names and formulae from AssignmentRules and remove the␣
↪respective rules
observables = amici.assignmentRules2observables(
```

(continues on next page)

(continued from previous page)

```
        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))'}}
```

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]:
```
sigmas = {'sd_pSTAT5A_rel', 'sd_pSTAT5B_rel', 'sd_rSTAT5A_rel'}
```

Generating the module

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

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

Importing the module and loading the model

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

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

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

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

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

```
Model parameters: ['Epo_degradation_BaF3', 'k_exp_hetero', 'k_exp_homo',␣
↪'k_imp_hetero', 'k_imp_homo', 'k_phos', 'sd_pSTAT5A_rel', 'sd_pSTAT5B_rel',␣
↪'sd_rSTAT5A_rel']
Model outputs:    ['observable_pSTAT5A_rel', 'observable_pSTAT5B_rel',␣
↪'observable_rSTAT5A_rel']
Model states:     ['STAT5A', 'STAT5B', 'pApB', 'pApA', 'pBpB', 'nucpApA', 'nucpApB',␣
↪'nucpBpB']
```

2.4.3 Running simulations and analyzing results

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

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

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

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

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

# Create solver instance
solver = model.getSolver()

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

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

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

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

(continues on next page)

(continued from previous page)

```
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:
47.976544579719786
```

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

optimizer.solver = 'bfgs'
```

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

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

2.4.5 Visualization

Create waterfall and parameter plot

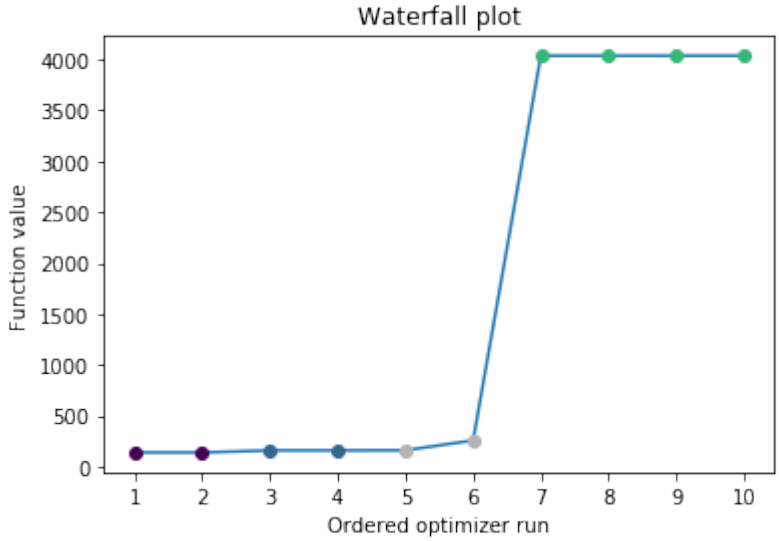
[18]:
```
# waterfall, parameter space,
import pypesto.visualize
```

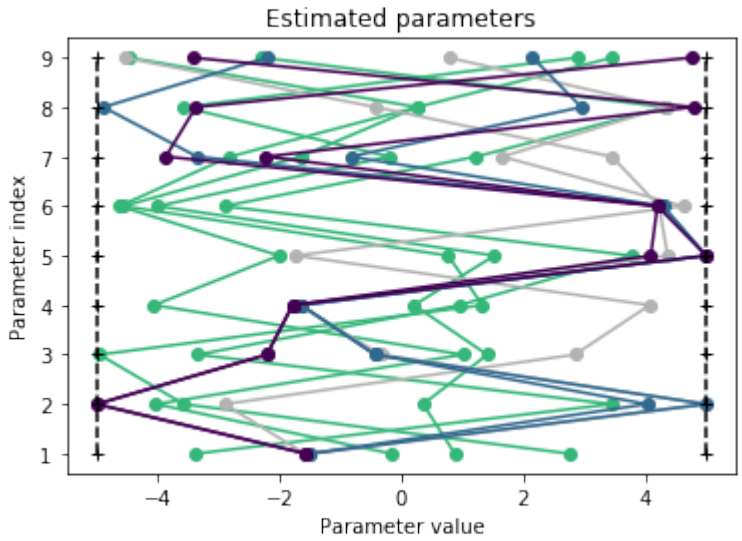
(continues on next page)

(continued from previous page)

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

[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd112e15208>
```





[ ]:

2.5 Download the examples as notebooks

- Rosenbrock
- Conversion reaction
- Fixed parameters
- Boehm model

Note: Some of the notebooks have extra dependencies.

CHAPTER 3

Contribute

3.1 Contribute documentation

3.2 Contribute tests

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

```
python3 -m pytest test
```

or alternatively:

```
python3 -m unittest test
```

You can also run specific tests.

Tests can be written with pytest or the unittest module.

3.2.1 PEP8

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

```
./run_flake8.sh
```

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

3.3 Contribute code

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

CHAPTER 4

Deploy

4.1 Versioning scheme

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

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

4.2 Deploy a new release

When you are done with the changes on your git branch, proceed as follows to deploy a new release.

4.2.1 Merge into master

First, you need to merge into the master:

1. check that all tests on travis pass
2. adapt the version number in the file pesto/version.py
3. update the release notes in doc/releasenotes.rst
4. merge into the origin master branch

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

4.2.2 Upload to PyPI

After a successful merge, you need to update also the package on PyPI:

5. create a so-called “wheel” via

```
python setup.py bdist_wheel
```

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

6. upload the archive to PyPI using twine via

```
twin upload dist/pypesto-x.y.z-py3-none-any.whl
```

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

The last step will only be possible if you have sufficient rights.

CHAPTER 5

Objective

class pypesto.objective.**Objective**(*fun=None*, *grad=None*, *hess=None*, *hessp=None*, *res=None*, *sres=None*, *fun_accept_sensi_orders=False*, *res_accept_sensi_orders=False*, *options=None*)

Bases: `object`

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

Parameters

- **fun** (*callable, optional*) – 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** (*callable, bool, optional*) – 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** (*callable, optional*) – 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** (*callable, optional*) –

  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** (*{callable, bool}, optional*) –

Method for computing residuals, i.e. `res(x) -> array_like, shape(m,).`

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

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

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

- **fun_accept_sensi_orders** (*bool, optional*) – Flag indicating whether fun takes sensi_orders as an argument. Default: False.

- **res_accept_sensi_orders** (*bool, optional*) – Flag indicating whether res takes sensi_orders as an argument. Default: False

- **options** (*pypesto.ObjectiveOptions, optional*) – Options as specified in pypesto.ObjectiveOptions.

history
pypesto.ObjectiveHistory – For storing the call history. Initialized by the optimizer in reset_history().

x_names
list of str – Parameter names. The base Objective class provides None. None if no names provided, otherwise a list of str, length dim_full. Can be read by the problem.

preprocess
callable – Preprocess input values to __call__.

postprocess
callable – Postprocess output values from __call__.

sensitivity_orders
tuple – Temporary variable to store requested sensitivity orders

Notes

preprocess, postprocess are configured in update_from_problem() and can be reset using the reset() method.

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

__init__ (*fun=None*, *grad=None*, *hess=None*, *hessp=None*, *res=None*, *sres=None*, *fun_accept_sensi_orders=False*, *res_accept_sensi_orders=False*, *options=None*)
Initialize self. See help(type(self)) for accurate signature.

static as_ndarrays (*result*)
Convert all array_like objects to numpy arrays. This has the advantage of a uniform output datatype which offers various methods to assess the data.

check_grad(*x*, *x_indices=None*, *eps=1e-05*, *verbosity=1*, *mode='mode_fun'*) → pandas.core.frame.DataFrame

Compare gradient evaluation: Firstly approximate via finite differences, and secondly use the objective gradient.

Parameters

- **x** (*array_like*) – The parameters for which to evaluate the gradient.
- **x_indices** (*array_like, optional*) – List of index values for which to compute gradients. Default: all.
- **eps** (*float, optional*) – Finite differences step size. Default: 1e-5.
- **verbosity** (*int*) –

  Level of verbosity for function output 0: no output 1: summary for all parameters 2: summary for individual parameters

  Default: 1.
- **mode** (*str*) – Residual (MODE_RES) or objective function value (MODE_FUN, default) computation mode.

Returns **result** – gradient, finite difference approximations and error estimates.

Return type pd.DataFrame

finalize_history()

Finalize the history object.

get_fval(*x*)

Get the function value at x.

get_grad(*x*)

Get the gradient at x.

get_hess(*x*)

Get the Hessian at x.

get_res(*x*)

Get the residuals at x.

get_sres(*x*)

Get the residual sensitivities at x.

has_fun

has_grad

has_hess

has_hessp

has_res

has_sres

static output_to_dict(*sensi_orders*, *mode*, *output_tuple*)

Convert output tuple to dict.

static output_to_tuple(*sensi_orders*, *mode*, ***kwargs*)

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

reset()

Completely reset the objective, i.e. undo the modifications in update_from_problem().

reset_history(*index=None*)
Reset the objective history and specify temporary saving options.

Parameters **index** (*As in ObjectiveHistory.index.*) –

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

class pypesto.objective.**ObjectiveOptions**(*trace_record=False*, *trace_record_grad=True*, *trace_record_hess=False*, *trace_record_res=False*, *trace_record_sres=False*, *trace_record_chi2=True*, *trace_record_schi2=True*, *trace_all=True*, *trace_file=None*, *trace_save_iter=10*)
Bases: dict

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

Parameters

- **trace_record** (*bool, optional*) – 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** (*bool, optional*) – Flag indicating whether to record the gradient in the trace. Default: True.
- **trace_record_hess** (*bool, optional*) – Flag indicating whether to record the Hessian in the trace. Default: False.
- **trace_record_res** (*bool, optional*) – Flag indicating whether to record the residual in the trace. Default: False.
- **trace_record_sres** (*bool, optional.*) – Flag indicating whether to record the residual sensitivities in the trace. Default: False.
- **trace_record_chi2** (*bool, optional*) – Flag indicating whether to record the chi2 in the trace. Default: True.
- **trace_record_schi2** (*bool, optional*) – Flag indicating whether to record the chi2 sensitivities in the trace. Default: True.

- **trace_all** (*bool, optional*) – Flag indicating whether to record all (True, default) or only better (False) values.
- **trace_file** (*str or True, optional*) – Either pass a string here denoting the file name for storing the trace, or True, in which case the default file name "tmp_trace_{index}.dat" is used. A contained substring {index} is converted to the multistart index. Default: None, i.e. no file is created.
- **index, optional** (*trace_save_iter.*) – Trace is saved every tr_save_iter iterations. Default: 10.

__init__(*trace_record=False*, *trace_record_grad=True*, *trace_record_hess=False*, *trace_record_res=False*, *trace_record_sres=False*, *trace_record_chi2=True*, *trace_record_schi2=True*, *trace_all=True*, *trace_file=None*, *trace_save_iter=10*)
Initialize self. See help(type(self)) for accurate signature.

static assert_instance(*maybe_options*)
Returns a valid options object.

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

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

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

which is the 'Linear' formulation in scipy.

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

class pypesto.objective.**AmiciObjective**(*amici_model*, *amici_solver*, *edata*, *max_sensi_order=None*, *preprocess_edata=True*, *options=None*)
Bases: pypesto.objective.objective.Objective

This is a convenience class to compute an objective function from an AMICI model.

Parameters

- **amici_model** (*amici.Model*) – The amici model.
- **amici_solver** (*amici.Solver*) – The solver to use for the numeric integration of the model.
- **edata** – The experimental data.
- **max_sensi_order** (*int*) – Maximum sensitivity order supported by the model.

__init__(*amici_model*, *amici_solver*, *edata*, *max_sensi_order=None*, *preprocess_edata=True*, *options=None*)
Initialize self. See help(type(self)) for accurate signature.

get_error_output(*sensi_orders*, *mode*)

postprocess_preequilibration(*data*, *original_value_dict*)

preprocess_edata(*edata_vector*)

preprocess_preequilibration(*data*)

CHAPTER 6

Problem

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

class pypesto.problem.**Problem**(*objective*, *lb*, *ub*, *dim_full=None*, *x_fixed_indices=None*, *x_fixed_vals=None*, *x_guesses=None*, *x_names=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** (*pypesto.Objective*) – The objective function for minimization. Note that a shallow copy is created.
- **ub** (*lb,*) – The lower and upper bounds. For unbounded directions set to inf.
- **dim_full** (*int, optional*) – The full dimension of the problem, including fixed parameters.
- **x_fixed_indices** (*array_like of int, optional*) – Vector containing the indices (zero-based) of parameter components that are not to be optimized.
- **x_fixed_vals** (*array_like, optional*) – Vector of the same length as x_fixed_indices, containing the values of the fixed parameters.
- **x_guesses** (*array_like, optional*) – Guesses for the parameter values, shape (g, dim), where g denotes the number of guesses. These are used as start points in the optimization.
- **x_names** (*array_like of str, optional*) – Parameter names that can be optionally used e.g. in visualizations. If objective.get_x_names() is not None, those values are used, else the values specified here are used if not None, otherwise the variable names are set to ['x0', ... 'x{dim_full}']. The list must always be of length dim_full.

dim

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

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

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*, *lb*, *ub*, *dim_full=None*, *x_fixed_indices=None*, *x_fixed_vals=None*, *x_guesses=None*, *x_names=None*)
Initialize self. See help(type(self)) for accurate signature.

get_full_matrix(*x*)
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*, *x_fixed_vals=None*)
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*)
Map matrix from dim_full to dim, i.e. delete fixed indices.

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

get_reduced_vector(*x_full*)
Map vector from dim_full to dim, i.e. delete fixed indices.

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

normalize_input()
Reduce all vectors to dimension dim and have the objective accept vectors of dimension dim.

CHAPTER 7

Optimize

pypesto.optimize.**minimize**(*problem*, *optimizer*, *n_starts*, *startpoint_method=None*, *result=None*, *options=None*) → pypesto.result.Result

This is the main function to call to do multistart optimization.

Parameters

- **problem** (*pypesto.Problem*) – The problem to be solved.
- **optimizer** (*pypesto.Optimizer*) – The optimizer to be used n_starts times.
- **n_starts** (*int*) – Number of starts of the optimizer.
- **startpoint_method** (*{callable, False}, optional*) – Method for how to choose start points. False means the optimizer does not require start points
- **result** (*pypesto.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.
- **options** (*pypesto.OptimizeOptions, optional*) – Various options applied to the multistart optimization.

class pypesto.optimize.**OptimizeOptions**(*startpoint_resample=False*, *allow_failed_starts=False*)

Bases: dict

Options for the multistart optimization.

Parameters

- **startpoint_resample** (*bool, optional*) – Flag indicating whether initial points are supposed to be resampled if function evaluation fails at the initial point
- **allow_failed_starts** (*bool, optional*) – Flag indicating whether we tolerate that exceptions are thrown during the minimization process.

__init__(*startpoint_resample=False*, *allow_failed_starts=False*)

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

static assert_instance(*maybe_options*)
Returns a valid options object.

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

class pypesto.optimize.**OptimizerResult**(*x=None*, *fval=None*, *grad=None*, *hess=None*, *n_fval=None*, *n_grad=None*, *n_hess=None*, *n_res=None*, *n_sres=None*, *x0=None*, *fval0=None*, *trace=None*, *exitflag=None*, *time=None*, *message=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.

x
ndarray – The best found parameters.

fval
float – The best found function value, fun(x).

grad, hess
ndarray – The gradient and Hessian at x.

n_fval
int – Number of function evaluations.

n_grad
int – Number of gradient evaluations.

n_hess
int – Number of Hessian evaluations.

exitflag
int – The exitflag of the optimizer.

message
str – Textual comment on the optimization result.

Notes

Any field not supported by the optimizer is filled with None. Some fields are filled by pypesto itself.

__init__(*x=None*, *fval=None*, *grad=None*, *hess=None*, *n_fval=None*, *n_grad=None*, *n_hess=None*, *n_res=None*, *n_sres=None*, *x0=None*, *fval0=None*, *trace=None*, *exitflag=None*, *time=None*, *message=None*)
Initialize self. See help(type(self)) for accurate signature.

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.

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

is_least_squares()

minimize(*problem*, *x0*, *index*)
" Perform optimization.

class pypesto.optimize.**ScipyOptimizer**(*method='L-BFGS-B'*, *tol=1e-09*, *options=None*)
Bases: pypesto.optimize.optimizer.Optimizer

Use the SciPy optimizers.

__init__(*method='L-BFGS-B'*, *tol=1e-09*, *options=None*)
Default constructor.

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

is_least_squares()

minimize(*problem*, *x0*, *index*)

class pypesto.optimize.**DlibOptimizer**(*method*, *options=None*)
Bases: pypesto.optimize.optimizer.Optimizer

Use the Dlib toolbox for optimization.

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

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

is_least_squares()

minimize(*problem*, *x0*, *index*)

CHAPTER 8

Profile

CHAPTER 9

Sample

CHAPTER 10

Result

The pypesto.Result object contains all results generated by the pypesto components. It contains sub-results for optimization, profiles, sampling.

class pypesto.result.**OptimizeResult**

Bases: object

Result of the minimize() function.

__init__()

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

append(*optimizer_result*)

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*) → list

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.

__init__()

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

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
pypesto.Problem – The problem underlying the results.

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.

CHAPTER 11

Visualize

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

pypesto.visualize.**waterfall**(*result*, *ax=None*)
Plot waterfall plot.

Parameters

- **result** (*pypesto.Result*) – Optimization result obtained by 'optimize.py'
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**waterfall_lowlevel**(*fvals*, *ax=None*)
Plot waterfall plot using list of function values.

Parameters

- **fvals** (*numeric list or array*) – Including values need to be plotted.
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.

Returns **ax** – The plot axes.

Return type matplotlib.Axes

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.
- **ind_clust** (*numeric list*) – Indices to reconstruct 'clust' from a list with 1:number of clusters.

pypesto.visualize.**assign_clustered_colors**(*vals*)
Cluster and assign colors.

Parameters **vals** (*numeric list or array*) – List to be clustered and assigned colors.

Returns **Col** – One for each element in 'vals'.

Return type list of RGB

pypesto.visualize.**parameters**(*result*, *ax=None*)
Plot parameter values.

Parameters

- **result** (*pypesto.Result*) – Optimization result obtained by 'optimize.py'.
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.

Returns **ax** – The plot axes.

Return type matplotlib.Axes

pypesto.visualize.**parameters_lowlevel**(*xs*, *fvals*, *lb=None*, *ub=None*, *x_labels=None*, *ax=None*)
Plot parameters plot using list of parameters.

Parameters

- **xs** (*nested list or array*) – Including optimized parameters for each startpoint. Shape: (n_starts, dim).
- **fvals** (*numeric list or array*) – Function values. Needed to assign cluster colors.
- **ub** (*lb,*) – The lower and upper bounds.
- **x_labels** (*array_like of str, optional*) – Labels to be used for the parameters.
- **ax** (*matplotlib.Axes, optional*) – Axes object to use.

Returns **ax** – The plot axes.

Return type matplotlib.Axes

CHAPTER 12

Startpoint

Method for selecting points that can be used as start points for multistart optimization. All methods have the form

    method(**kwargs) -> startpoints

where the kwargs can/should include the following parameters, which are passed by pypesto:

n_starts: int Number of points to generate.

lb, ub: ndarray Lower and upper bound, may for most methods not contain nan or inf values.

x_guesses: ndarray, shape=(g, dim), optional Parameter guesses by the user, where g denotes the number of guesses. Note that these are only possibly taken as reference points to generate new start points (e.g. to maximize some distance) depending on the method, but regardless of g, there are always n_starts points generated and returned.

objective: pypesto.Objective, optional The objective can be used to evaluate the goodness of start points.

max_n_fval: int, optional The maximum number of evaluations of the objective function allowed.

pypesto.startpoint.**uniform**(***kwargs*)
Generate uniform points.

pypesto.startpoint.**latin_hypercube**(***kwargs*)
Generate latin hypercube points.

pypesto.startpoint.**assign_startpoints**(*n_starts*, *startpoint_method*, *problem*, *options*)
Assign startpoints.

CHAPTER 13

Release notes

13.1 0.0 series

13.1.1 0.0.2 (2018-10-18)

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

13.1.2 0.0.1 (2018-07-25)

- Basic framework and implementation of the optimization

CHAPTER 14

Authors

This package was mainly developed by:

- Jan Hasenauer
- Yannik Schälte
- Fabian Fröhlich
- Daniel Weindl
- Paul Stapor
- Leonard Schmiester
- Dantong Wang
- Leonard Schmiester
- Caro Loos

CHAPTER 15

Contact

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

- Yannik Schälte: yannik.schaelte@gmail.com

CHAPTER 16

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 17

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

C

D

E

F

G

H