abcpmc Documentation

Release 0.1.2

Joel Akeret

April 21, 2016

Contents

A Python Approximate Bayesian Computing (ABC) Population Monte Carlo (PMC) implementation based on Sequential Monte Carlo (SMC) with Particle Filtering techniques.

The **abcpmc** package has been developed at ETH Zurich in the Software Lab of the Cosmology Research Group of the ETH Institute of Astronomy.

The development is coordinated on GitHub and contributions are welcome. The documentation of **abcpmc** is available at readthedocs.org and the package is distributed over PyPI.

CHAPTER 1

Features

- Entirely implemented in Python and easy to extend
- Follows Beaumont et al. 2009 PMC algorithm
- Parallelized with muliprocessing or message passing interface (MPI)
- Extendable with k-nearest neighbour (KNN) or optimal local covariance matrix (OLCM) pertubation kernels (Fillipi et al. 2012)
- Detailed examples in IPython notebooks
  - A 2D gauss case study
  - A Multi distance case study
  - A toy model including a comparison to theoretical predictions

1.1 Contents:

1.1.1 Installation

At the command line via pip:

```
$ pip install abcpmc
```

This will install the package and all of the required dependencies.

Note: If you wish to use *abcpmc* on a cluster with MPI you need to manually install mpi4py.

1.1.2 Usage

IPython notebook examples

Detailed examples are available in the project and online: A 2D gauss example and a toy model including a comparison to theoretical predictions.

General usage of abcpmc

In generall the use of abcpmc in simple:

```
import abcpmc
import numpy as np

#create "observed" data set
size = 5000
sigma = np.eye(4) * 0.25
means = np.array([1.1, 1.5, 1.1, 1.5])
data = np.random.multivariate_normal(means, sigma, size)
#-------

#distance function: sum of abs mean differences
def dist(x, y):
    return np.sum(np.abs(np.mean(x, axis=0) - np.mean(y, axis=0)))

#our "model", a gaussian with varying means
def postfn(theta):
    return np.random.multivariate_normal(theta, sigma, size)

eps = abcpmc.LinearEps(20, 5, 0.075)
prior = abcpmc.GaussianPrior(means*1.1, sigma*2) #our best guess

sampler = abcpmc.Sampler(N=10, Y=data, postfn=postfn, dist=dist)

for pool in sampler.sample(prior, eps):
    print("T: {0}, eps: {1:>.4f}, ratio: {2:>.4f}".format(pool.t, pool.eps, pool.ratio))
    for i, (mean, std) in enumerate(zip(np.mean(pool.thetas, axis=0), np.std(pool.thetas, axis=0))):
        print(u"    theta[{0}]: {1:>.4f} \u00B1 {2:>.4f}".format(i, mean,std))
```

the resulting output would look something like this:

```
T: 0, eps: 2.0000, ratio: 0.2439
    theta[0]: 0.9903 ± 0.5435
    theta[1]: 1.6050 ± 0.4912
    theta[2]: 1.0567 ± 0.4548
    theta[3]: 1.2859 ± 0.5213
T: 1, eps: 1.8987, ratio: 0.3226
    theta[0]: 1.1666 ± 0.4129
    theta[1]: 1.6597 ± 0.5227
    theta[2]: 1.1263 ± 0.3366
    theta[3]: 1.4711 ± 0.2150
T: 2, eps: 1.7974, ratio: 0.3030
    theta[0]: 1.1263 ± 0.2505
    theta[1]: 1.4832 ± 0.5057
    theta[2]: 1.0585 ± 0.3387
    theta[3]: 1.4782 ± 0.2808
T: 3, eps: 1.6961, ratio: 0.4167
    theta[0]: 1.1265 ± 0.1845
    theta[1]: 1.2032 ± 0.4470
    theta[2]: 1.0248 ± 0.2074
    theta[3]: 1.4689 ± 0.4250

...

T: 19, eps: 0.0750, ratio: 0.0441
    theta[0]: 1.1108 ± 0.0172
```

```
    theta[1]: 1.4832 ± 0.0166
    theta[2]: 1.0895 ± 0.0202
    theta[3]: 1.5016 ± 0.0097
```

Parallelisation on cluster with MPI

abcpmc has an built-in support for massively parallelized sampling on a cluster using MPI.

To make use of this parallelization the abcpmc Sampler need to be initialized with an instance of the *MpiPool*:

```
import abcpmc
from abcpmc import mpi_util

...

mpi_pool = mpi_util.MpiPool()
sampler = abcpmc.Sampler(N, Y, postfn, dist, pool=mpi_pool) #pass the mpi_pool

if mpi_pool.isMaster(): print("Start sampling")

for pool in sampler.sample(prior, eps):

...
```

If the threshold is dynamically adapted the user has to make sure that the state is synchonized among all MPI task with a broadcast:

```
eps.eps = mpi_util.mpiBCast(new_threshold)
```

Finally, the job has to be launched as follows to run on *N* tasks in parallel (might depend on your system):

```
$ mpirun -np N python <your-abc-script.py>
```

1.1.3 abcpmc Package

sampler Module

class abcpmc.sampler.**Sampler**(*N*, *Y*, *postfn*, *dist*, *threads=1*, *pool=None*)

ABC population monte carlo sampler

Parameters

- **N** – number of particles
- **Y** – observed data set
- **postfn** – model function (a callable), which creates a new dataset x for a given theta
- **dist** – distance function rho(X, Y) (a callable)
- **threads** – (optional) number of threads. If >1 and no pool is given <threads> multiprocesses will be started
- **pool** – (optional) a pool instance which has a <map> function

close()

Tries to close the pool (avoid hanging threads)

particle_proposal_cls
alias of ParticleProposal

sample(*prior*, *eps_proposal*, *pool=None*)
Launches the sampling process. Yields the intermediate results per iteration.

Parameters

- **prior** – instance of a prior definition (or an other callable) see sampler.GaussianPrior
- **eps_proposal** – an instance of a threshold proposal (or an other callable) see sampler.ConstEps
- **pool** – (optional) a PoolSpec instance,if not None the initial rejection sampling

will be skipped and the pool is used for the further sampling

Yields pool yields a namedtuple representing the values of one iteration

class abcpmc.sampler.**GaussianPrior**(*mu*, *sigma*)
Normal gaussian prior

Parameters

- **mu** – scalar or vector of means
- **sigma** – scalar variance or covariance matrix

class abcpmc.sampler.**TophatPrior**(*min*, *max*)
Tophat prior

Parameters

- **min** – scalar or array of min values
- **max** – scalar or array of max values

class abcpmc.sampler.**ParticleProposal**(*sampler*, *eps*, *pool*, *kwargs*)
Creates new particles using twice the weighted covariance matrix (Beaumont et al. 2009)

class abcpmc.sampler.**OLCMParticleProposal**(*sampler*, *eps*, *pool*, *kwargs*)
Bases: *abcpmc.sampler.ParticleProposal*

Creates new particles using an optimal loacl covariance matrix (Fillipi et al. 2012)

class abcpmc.sampler.**KNNParticleProposal**(*sampler*, *eps*, *pool*, *kwargs*)
Bases: *abcpmc.sampler.ParticleProposal*

Creates new particles using a covariance matrix from the K-nearest neighbours (Fillipi et al. 2012) Set *k* as key-word arguement in *abcpmc.Sampler.particle_proposal_kwargs*

abcpmc.sampler.**weighted_cov**(*values*, *weights*)
Computes a weighted covariance matrix

Parameters

- **values** – the array of values
- **weights** – array of weights for each entry of the values

Returns sigma the weighted covariance matrix

abcpmc.sampler.**weighted_avg_and_std**(*values*, *weights*, *axis=None*)
Return the weighted avg and standard deviation.

Parameters

- **values** – Array with the values
- **weights** – Array with the same shape as values containing the weights
- **axis** – (optional) the axis to be used for the computation

Returns avg, sigma weighted average and standard deviation

threshold Module

Various different threshold implementations Created on Jan 19, 2015

author: jakeret

class abcpmc.threshold.**ConstEps**(*T*, *eps*)
Bases: *abcpmc.threshold.EpsProposal*

Constant threshold. Can be used to apply alpha-percentile threshold decrease :param eps: epsilon value

class abcpmc.threshold.**EpsProposal**(*T*)
Bases: object

next()

reset()

class abcpmc.threshold.**ExponentialConstEps**(*max*, *min*, *T1*, *T2*)
Bases: *abcpmc.threshold.EpsProposal*

class abcpmc.threshold.**ExponentialEps**(*T*, *max*, *min*)
Bases: *abcpmc.threshold.EpsProposal*

Exponentially decreasing threshold

Parameters

- **max** – epsilon at t=0
- **min** – epsilon at t=T
- **T** – number of iterations

class abcpmc.threshold.**LinearConstEps**(*max*, *min*, *T1*, *T2*)
Bases: *abcpmc.threshold.EpsProposal*

Linearly decreasing threshold until T1, then constant until T2

Parameters

- **max** – epsilon at t=0
- **min** – epsilon at t=T
- **T1** – number of iterations for decrease
- **T2** – number of iterations for constant behavior

class abcpmc.threshold.**LinearEps**(*T*, *max*, *min*)
Bases: *abcpmc.threshold.EpsProposal*

Linearly decreasing threshold

Parameters

- **max** – epsilon at t=0
- **min** – epsilon at t=T

- **T** – number of iterations

class abcpmc.threshold.**ListEps**(*T*, *eps_vals*)
Bases: *abcpmc.threshold.EpsProposal*

mpi_util Module

A helper util for message passing interface (MPI) handlings

class abcpmc.mpi_util.**MpiPool**(*mapFunction=<built-in function map>*)
Bases: object

isMaster()
Returns true if the rank is 0

map(*function*, *sequence*)

abcpmc.mpi_util.**mpiBCast**(*value*)
Mpi bcasts the value and returns the value from the master (rank = 0).

1.1.4 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Implement Features

Write Documentation

abcpmc could always use more documentation, whether as part of the official abcpmc docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.

- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test test/test_abcpmc.py
```

1.1.5 Credits

Development Lead

- Joel Akeret <jakeret@phys.ethz.ch>

Contributors

None yet. Why not be the first?

Citations

As you use **abcpmc** for your exciting discoveries, please cite the paper that describes the package:

Akeret, J., Refregier, A., Amara, A, Seehars, S., and Hasner, C., Journal of Cosmology and Astroparticle Physics (2015)

1.1.6 History

0.x.x (2016-0x-xx)

- Fixed map issue under Py3
- Replaced numpy.random with a RandomState
- Restart sampling feature

0.1.2 (2016-01-27)

- Added support for sampling with multiple distance simultaneously
- Clean setup.py
- Simplifying the code
- Improved documentation

0.1.1 (2015-05-03)

- Python 3 support
- Minor fixes
- Improved documentation

0.1.0 (2015-04-28)

- First release

1.2 Feedback

If you have any suggestions or questions about **abcpmc** feel free to email me at jakeret@phys.ethz.ch.

If you encounter any errors or problems with **abcpmc**, please let me know!

a