py-SMC^2

User manual

Contents

1 Introduction

This note intends to show how to use the `py-smc2` package.

1.1 What's in there

This package implements various methods to perform Bayesian inference in state space models. It focuses on SMC^2, a method presented in *SMC^2: A sequential Monte Carlo algorithm with particle Markov chain Monte Carlo updates*, by Nicolas Chopin, Pierre E. Jacob, Omiros Papaspiliopoulos. The article is available on arXiv at this address:

> http://arxiv.org/abs/1101.1528

For comparison, other methods are implemented:

- *Particle Markov Chain Monte Carlo* by Andrieu, Doucet and Holenstein (2010, Journal of the Royal Statistical Society, Series B, 72:269-342):

  > CiteSeerx link

  This MCMC method allows to sample from the posterior distribution of a state space model, but not sequentially. The package implements the Particle Marginal Metropolis–Hastings (PMMH) algorithm, not the Particle Gibbs. We also added the adaptive proposal (as in adaptive Metropolis–Hastings, where a specific acceptance rate is targeted), following various recent articles.

- *Combining Parameter and State Estimation in Simulation-Based Filtering*, by Liu and West. In A. Doucet, N. de Freitas, and N. J. Gordon, editors, Sequential Monte Carlo Methods in Practice, pages 197-224. Springer, New York. This method allows to sample from the posterior distributions sequentially, like SMC^2 does. This method is referred to in the package as "BSMC".

- *A self-organizing state-space model* by G Kitagawa (Journal of the American Statistical Association, 1998). This is like a simpler version of BSMC, where the parameters are reweighted but not moved.

1.2 What's a state space model

Sometimes also referred to as Hidden Markov Chain model, but here the latent process can be defined on continuous spaces.

The observations are denoted by $y_1, \ldots y_T$. Throughout this note $y_{k:l} := y_k, \ldots, y_l$. In a state space models the observations are supposed to be dependent, through a hidden process (x_t). We first set an initial distribution for x_0:

$$x_0 \sim \mu_\theta(\cdot)$$

Then the process evolves through its "dynamics":

$$x_t \sim f_\theta(\cdot|x_{t-1})$$

And the observations are supposed to be drawn from a “measure” distribution:

$$y_t \sim g_\theta(\cdot|x_t)$$

Note that these functions are indexed by θ, referred to as the parameter. The parameter lives in $\Theta \subset \mathbb{R}^d$. For convenience, we assume that we can transform the parameters so that $\Theta = \mathbb{R}^d$ (that’s a requirement of the package but not of the method in its full generality).

We are interested in the cases where:

- we can sample from μ_θ
- we can sample from $f_\theta(\cdot|x_{t-1})$
- we can compute $g_\theta(y_t|x_t)$ up to a multiplicative constant.

And we are interested in estimating the parameters θ, as well as finding the hidden process (x_t). We are also interested in byproducts of this estimation problem, for instance in model comparison, prediction, any functional of the hidden states, smoothing, etc.

2 Quick start-up guide

2.1 Installation

The package is coded in python and there are no binaries, hence requires python…

Also it requires the following:

- `numpy`, `scipy` (usually packaged as `python-numpy`, `python-scipy`),
- The package makes extensive use of `scipy.weave`, in particular of the function inline. It is installed along with `scipy`. However for it to work you also need a C compiler like `gcc`. Hence make sure you have the latest version of `gcc`. Mac OS user should consider installing XCode.
- `Rpy2` does not hurt but is not required (mainly if you have it installed, something at some point will be performed quicker),
- `cProfile` it is a profiler for python, usually installed along with python, and it is only required if the profiling option is enabled.
- `R` is required for the graphs, but you can disable the graph part (see the `PLOT` option, in the next section). The `ggplot2` package is also required, so once you’ve installed `R`, install this package by launching:

```
install.packages("ggplot2")
```

Once you have all this, simply extract the package (you can find the zip archive on the Google Code webpage):

http://code.google.com/p/py-smc2/downloads/list

or fetch it using `hg` like this in a terminal:

```
hg clone https://code.google.com/p/py-smc2/
```

To try if the package works, launch the program from the root folder in a terminal:

```
python launch.py
```

If the package does not work (ie you see errors in the terminal after having launched the previous command), it is probably because of `scipy.weave`. So make sure you can use `scipy.weave` before investigating further. I would recommend going there: http://www.scipy.org/PerformancePython and checking whether you can make the `scipy.weave.inline` snippet work on your system.

2.2 The user file

The file `userfile.py` is the first file you should look at. It is not really code, but merely algorithmic parameters. The important ones are the following:

- `MODEL`: it should be a string indicating a state-space model. Each model has two files in the `models` subfolder. For instance `locallevel` is described by `locallevelx.py` and `localleveltheta.py`. The list of models currently available is:

  - `hiddenAR` : a simple linear gaussian model with one parameter: the coefficient of the auto-regressive hidden process. The variance of its noise and the variance of the observational noise are fixed and considered known.
  - `locallevel` : a simple local level model with two parameters: the variance of the hidden random walk, and the variance of the measurement function.
  - `periodic`: a non-linear gaussian model as in Gordon, Salmond and Smith 1993.
  - `athletics`: the model used in the paper for modelling athletics records (with a GEV measurement function and a second order random walk for the hidden process)
  - `thetalogistic`: an ecological model for predicting population sizes; it is highly non-linear and gaussian.
  - `SVonefactor`: the Stochastic volatility model used in the paper.
  - `SVfixedrho`, `SVwithFixedParameters`, `SVmultifactor`: variations of the above.

  (forget about the models finishing with `CUDA` for now)

- `T`: the number of observations to take into account. For testing, set a small value like 10 (if it does not work for 10, it won't work for more!).

- `DATASET`: a string indicating the data set to use. If you put `synthetic` then a new data set will be simulated from the model. You can browse in the `data` subfolder to see the different data sets available; their filenames finish by `.R`. Let's put `synthetic` there for starters.

- METHOD: a string indicating the method. Currently there are:
  - SMC2
  - SOPF
  - BSMC
  - adPMCMC

  For now let us suppose that you want to try SMC^2, so put SMC2 there for starters.

That's enough to get the main idea. Then of course you might want to tweak the algorithmic settings, which are different for each method. For SMC^2, the most important ones are:

- NTHETA: the number of θ-particles. A good default is to put the maximum between 1000 and the number of observations.
- NX: the number of x-particles attached to each θ-particle.
- DYNAMICNX: a boolean (True or False in python, beware of the upper-case first letter) indicating whether the number of x-particles should increase along the iterations or not. If it is set to True, then when the acceptance rate of the PMMH moves goes below DYNAMICNXTHRESHOLD, the number of x-particles is multiplied by two.

Finally, there are various other parameters you might be interested in:

- SAVINGTIMES: should be a list of integers, at which the results (ie the weighted particles) should be stored. By default it is empty, but suppose that you want to do sequential analysis, that's where you should specify it. Note that some things are stored at each iteration anyway, like the ESS and the evidence.
- PROFILING: a boolean (True or False) indicating whether the computing time should be reported in a separate file. It slightly slows down the program but it is most often interesting.
- PLOT: a boolean (True or False) indicating whether results should be presented in graphs. By default the program creates a R file (GENERATERFILE is True), and if PLOT is enabled, then the R file is launched at the end. So you can directly look at a pdf file in the results subfolder, and see the various posteriors, the ESS along the iterations, etc. If you have not installed R, then you should disable this, otherwise it will result in an error (but your results will still be saved).

The other parameters in userfile.py are pretty self-explanatory. Some are used only if some other method is used (SOPF, BSMC, adPMCMC). Some would deserve explanations in a future version of this manual (like SMOOTHING, or PREDICTION).

Some user files than the default one are provided in the userfiles/ subfolder.

2.3 Launching the program

Once the user file is set-up, how to launch it? First make sure you saved the user file (its name should be `userfile.py`) if you made any change to it.

Then launch a terminal, go in the root folder of the package and type

```
python launch.py
```

The file `launch.py` should not be modified: only `userfile.py` should be changed. The file `launch.py` basically analyses `userfile.py`, and then calls the appropriate functions in the right order. These core functions are in the `src` subfolder, but shouldn't be modified or called directly in most cases. The user file is made precisely so that the user does not have to handle the core functions.

Note that you can name your user file the way you want and put in the folder you want. Suppose that you saved your userfile as `myuserfile.py` in the folder `/mydocuments/`, then you can launch the program on your file with the path to the user file as an argument of the program:

```
python launch.py /mydocuments/myuserfile.py
```

This makes it easy to store user files without having to move and rename them each time you want to launch them.

During the program you are able to follow the algorithm's progression. If you set up the parameters so that it should be fast (small `T`, small `NTHETA`, small `NX`), then it should take a few seconds. For bigger parameters values, the program can take hours (for any of the proposed methods), so be sure that you set up the parameters right before launching. The computing time can also be very different from one model to the other (the stochastic volatility models being the slowest of the proposed models).

2.4 Understanding more: file structure

The `py-smc2` package is organized as follows:

- the root folder contains mainly `userfile.py` and `launch.py`. The user file is the one file that should be modified by most users. The launch file is there to be executed but not modified.

- `data` subfolder: where the data sets should be. There are the files finishing by `.R`. These are not actually `R` code files but simple text files with one value per line. Open `anisus.R` or `nutria.R` for simplest examples. If you want to add a new data set, add a file in this subfolder, with the `.R` file extension. You can then specify it in the `DATASET` entry of the user file (without the extension), as we saw earlier. For example, if you add `foo.R` in the `data` subfolder, put `DATASET = "foo"` in the user file. Just make sure it is in the same format as the proposed data sets, that is: one value per line, no column names, if the observation is multidimensional, the different components are separated by a space (see `athletics-best-two.R` for an example).

- `manual` subfolder: where this manual is.
- `models` subfolder: where the model files are. There are two files for each model: for a model `foo`, there would be `foox.py` and `footheta.py`. The file finishing in `x.py` mainly described the dynamics of the hidden process and the measurement function. The file finishing in `theta.py` describes the transformations made the parameters so that $\Theta = \mathbb{R}^d$ and the prior distribution on the parameters.
- `results` subfolder: when the program is launched, it creates subfolders in this folder. In the subfolder, various files describe the results. Some files just store the results (stored by default in `.RData` format). If enabled, the profiling is stored in a separate file (finishing by `-profile.txt`). The user file is copied in this subfolder too, so that one knows which parameters produced the given results (file finishing by `-userfile.py`). An `R` file is created (file finishing by `-plots.R`). This file, when launched, produces graphs. If it was launched by the program (option `PLOT` set to `True`), then a pdf file is already in the subfolder. If not, then launching the `R` file creates the pdf file. The `R` file is also informative if you want to know precisely how the results are stored. Executing the `R` file line by line should be enlightening to most advanced users.
- `rgraphs` subfolder: this messy folder stores the code to create more complicated graphs, involving model comparison or method comparison. They are not supposed to work directly, but only after the program has been launched a few times with specific settings and result files are available. Most users should avoid this folder.
- `snippets` subfolder: this groups python functions needed by the program but totally unrelated to statistics.
- `src` subfolder: this is the main code folder, with the various methods, the model handling, the generation of R files, etc. This is all in python, with bits in C using `weave.inline`.
- `test` subfolder: this is a folder to test bits of code before incorporating them in the main files. Not interesting for most users.
- `userfiles` subfolder: contains a bunch of userfile examples, for launching various methods on various models.

3 Using the package for your own purpose

3.1 Making your own models

Let us look at a model file line by line.

First one needs to specify the state space model, that is:

- μ_θ, the distribution of x_0, from which one needs to sample from,
- f_θ, the transition function, from which one needs to sample from,
- g_θ, the measure function, from which one needs to evaluate the density.

Additionally you might want to specify a function to sample from g_θ, which can be useful to generate synthetic data sets. Otherwise you could still use the model on real data set, which is the most important.

Let us see how to specify a model by taking the example of the hidden auto regressive model.

- μ_θ is $\mathcal{N}(0,1)$
- $f_\theta(x_t|x_{t-1})$ is $\mathcal{N}(\rho x_{t-1}, \sigma^2)$
- $g_\theta(y_t|x_t)$ is $\mathcal{N}(\rho x_t, \tau^2)$

For simplicity we are going to suppose that σ and τ are known, hence the only parameter left to estimate is ρ.

3.2 Adding your data files