Pi4U tutorial

What is Pi4U?

Pi4U is an extensible framework for non-intrusive Bayesian Uncertainty Quantification and Propagation of complex and computationally demanding physical models, that can exploit massively parallel computer architectures

What is implemented in Pi4U?

- TMCMC (for exact Bayesian inference)
- CMA-ES (for optimization)
- Subset Simulation (for rare event sampling)
- ABC-SubSim (for approximate Bayesian inference)
- A-PNDL (for parallel numerical differentiation)

Next steps in Pi4U?

- APIs for python, MATLAB, R
- Manifold TMCMC
- Hierarchical Bayesian (HB)
- Surrogate modelling
- EM algorithm for HB

TODO:

- Fix the composite prior and document it properly
- Document the Gaussian prior
- What priors do we support? Best option: a keyword for the prior for each parameter, then give prior parameters
- GIve the users the possibility to make their own prior
- Add a check for the dimensionality vs number of parameters set by the user
- Provide an example with a fitfun in the script form
- Delete sn-TMCMC and everything else which is obsolete
- Delete with/without exp acceptance (leave only without)
- CMA-stype proposal: make clear that it's 0 or 1
- Document interactive dumping more clearly
- Add CMA boundaries to the initials file
- DIscuss if we should have MAP (as an option in the parameter file)

INSTALLATION on Mac:

Pi4U is based on TORC, a task-parallel library created by P. Hadjidoukas [1]. The steps to install TORC on a Mac are:

1. `brew install mpich gsl R` *# skip if you already have those package*
2. `R` # opens R. Inside R type the following:
   a. `install.packages("sp")` # for plotting data
   b. `quit()` *# quits R*
3. `cd <path to your new pi4u installation>`
4. `git clone <nethz username>@scratch-petros.ethz.ch :/export/home/gitroot/pi4u.git`
5. `cd pi4u/torc_lite`
6. `./configure CC=mpicc --prefix=<install folder>` *# default prefix: /usr/local*
7. `make && make install`

INSTALLATION on a cluster

Instead of 1. load the corresponding modules

INSTALLATION on Linux (Ubuntu)

Follow the steps for Mac, just instead of step 1. Run:

```
sudo apt-get install libmpich-dev libgsl-dev r-base
```

For the step `"install.packages("sp")"` in `R` and the step `"make install"` (in the case the default `--prefix` value was used in the `./configure` step), you may need to use `sudo`.

EXAMPLE to run:

Sample negative Rosenbrock function (a = 1, b = 100) using TMCMC [2]:

1. `cd pi4u.git/engines`
2. `make`
3. `mpirun -np 1 ./engine_tmcmc`
4. `cd postprocessing_tools`
5. set `fname` in `kdeplotter.R` to "`../curgen_db_008.txt`" which contains samples of the last generation
6. Plot results: `R --no-save < kdeplotter.R`
7. `cd ..`
8. open `curgen_db_008.png` *# you should see Fig. 1*

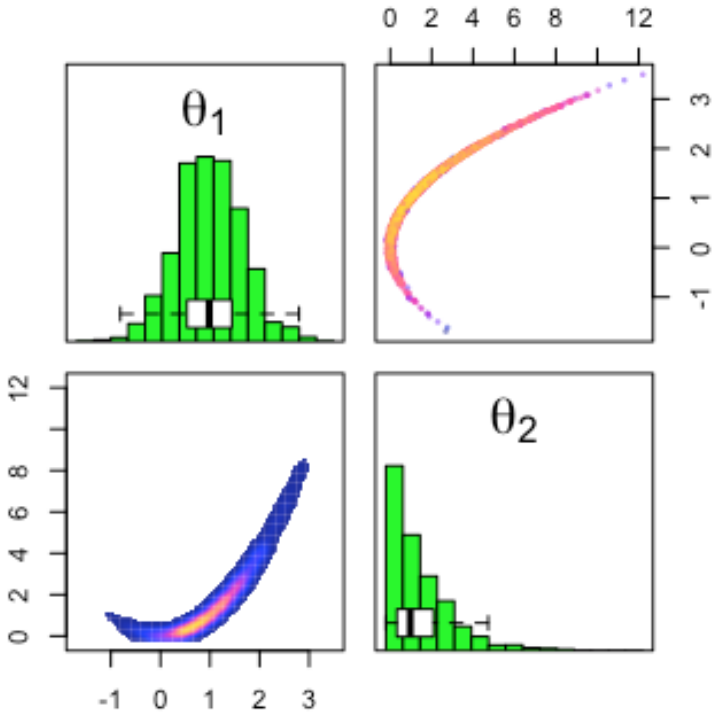

Fig. 1. Posterior PDF for the Rosenbrock function. Diagonals: marginal distributions of the corresponding parameters. Above diagonal: samples colored by the log-likelihood value. Below diagonal: smoothed histogram of the samples.

PARAMETER file: `tmcmc.par`

The parameters in this file control the behavior of TMCMC. Be careful to change the prior parameters when changing dimension N of the problem, e.g. for gaussian prior the covariance matrix should be of dimension NxN (in matlab format) and the mean vector of dimension N.

SETTING UP your example

1. Write your fitness function:
    a. for an easy log-likelihood: modify `engines/fitfun.c` in such a way that it takes a vector of parameters and returns the log-likelihood value (see engines/fitfun.c for examples). The `fitfun` should have the following signature: double `fitfun(double *x, int N, void *output, int *info)`, where `x` is the current sample, `N` is the dimensionality of the problem (denoted `Nth` in the `tmcmc.par`), `output` is used if the function should return something more than just a log-likelihood value, `info` contains information about the position of the current sample in the TMCMC task-graph: `info[0]` = stage, `info[1]` = chain, `info[2]` = task, `info[3]` can be used for subtask index.
    b. for a complicated log-likelihood: write a script which will compute your log-likelihood value and call it from the `engines/fitfun.c`
2. Modify the `tmcmc.par` file accordingly: change the dimensionality of the problem (`Nth`), the population size (`PopSize`), the type of prior

Running other engines

CMA-ES:

CMA-ES is a black-box minimization algorithm [3].

1. `mpirun -np 1 ./engine_cmaes` *# It uses the same `fitfun.c` and is compiled together with `engine_tmcmc`*
2. Have in mind that CMA-ES does minimization, so the log-likelihood has a minus sign inside `engine_cmaes.c`, because we look for the parameters which maximize the log-likelihood
3. Check the `outcmaesxrecentbest.dat` for the output
4. Now you can play with the parameters: edit `cmaes_initials.par` (see the comments inside)

Troubleshooting

- **MPI:** Make sure your localhost name is the same in your terminal and your `/etc/hosts` file. If not, you will get an error when trying to run mpich, “gethostbyname failed”. To fix that: do `cat /etc/hosts` and it should print out sth like: `// 127.0.0.1 my_hostname`, where `my_hostname` should match the one in your terminal. If not, change one of them: change `/etc/hosts`: `sudo vim /etc/hosts`. Change terminal hostname: `sudo scutil --set HostName my_new_host_name`

References

[1] P.E. Hadjidoukas, E. Lappas, and V.V. Dimakopoulos. “A runtime library for platform-independent task parallelism”, *In 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 229–236. IEEE Computer Society, Munich, Germany, 2012.
[2] J. Ching and Y. Chen. “Transitional Markov chain Monte Carlo method for Bayesian model updating, model class selection, and model averaging.” *J. Eng. Mech.*, 133(7):816–832, 2007.
[3] Hansen, Nikolaus, Sibylle D. Müller, and Petros Koumoutsakos. “Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES).” *Evolutionary computation* 11.1 (2003): 1-18.