Building of `mltp.jc()` and Simulations Results

February 17, 2008

1 Overview

The program `mltp.jc()` is based on a slightly modified version of the original function `jean.cocteau()`, which tests the performance of a 4-kernel PMC scheme for a two gaussians mixture problem. The program computes the modes positions and the total weight associated to each mode by calculating their basins of influence –that is the maximum contour lines encircling each mode – and counting the number of particles therein. The algorithm loops since the convergence is attained. Then the programs checks whether the maximum weight is on the highest mode. Further analysis is performed on cases in which the mode that has the maximum weight is not the highest nor symmetrical to the highest.

`mltp.jc()` allows for the possibility of running multiple cycles of `jean.cocteau()` with a same set of parameters, in order to gather statistics on the algorithm. The seeds for both the generation of the data and particles dynamics are stored, so that each simulation can be replicated and several particles dynamics analyzed on the same log-posterior surface. The default output consists of:

- two files containing respectively the coordinates of the log-posterior surface"special" points (i.e. peaks, pits and saddle points) and those of the basins. In this way, these parameters can be simply uploaded in case of several cycles with the same set of data;
- a text file resuming all the parameter values, weights dynamics during the computation and main results;
- a postscript file containing all plots, that can at any rate be compressed.

2 Inputs and Outputs

The complete list of the *new input variables* is:

- `sigma2` → the value of the second variance, which can be set indipendently from `sigma1` (default: `sigma2=sigma1`);
- `gridsiz` → the old variable which can now be modified on input (default: `gridsiz=150`);
- `tau` → the convergence threshold (default: `tau=1e-2`);
- `convcri` → it allows the choice among the three convergence criteria (see Sect. 4). It can take four values `(0,1,2,3)` which imply respectively: no criteria called for, the proposal weight, the Shannon's entropy or the Effective Sample Size (ESS) criteria (default: `convcri=0`);
- `beta` → if `convcri == 1`, it sets the value of the convergence parameter in the Euler equation (2), (default: `beta=1e-5`);
- `howmany` → if `convcri == 1`, it set the number of proposal weight on which the convergence is based. It can take integer values from 1 (weak convergence) to 4 (strong convergence), (default: `howmany=1`);
- `niter` → maximum number of iterations: when the number of iterations reaches this value the program stops even if the convergence has not been attained (default: `niter=200`);
- `ncycles` → number of successive groups of iterations with the same set of parameters but different couples of seeds `(myseedD, myseedP)`, (default: `ncycles=1`);
- `againD` → if `TRUE`, the program looks up for data seeds in the file `nfseedD` or takes the value of `myseedD` (default: `againD=FALSE`);
- `againP` → the same as `againD` but for particles seeds (default: `againP=FALSE`);
- `myseedD` → the chosen seed value, which is an integer of 1 to 9 digits (default: `myseedD=NULL`);

- `myseedP` → the same as `myseedD` but for particles seed (default: `myseedP=NULL`);
- `nfseedD` → the name of the file containing the data seeds (default: `nfseedD=""`);
- `nfseedP` → the same as `nfseedD` but for particles seed (default: `nfseedP=""`);
- `info` → if `TRUE` then the program writes a summary of the inputs and the main outputs in the file `iosum_n(`*data*`)c(`*cycle*`)s(`*seed*`)`, otherwise they appear on the screen during the simulation (default: `info=FALSE`);
- `doplot` → when equal to `0`, the simulation produces no plot, otherwise plots may be saved in a postscript file `sim_n(`*data*`)c(`*cycle*`)s(`*seed*`)` (`doplot==1`) or outputed on the screen (`doplot==2`, default);
- `ndir` → name of the directory containing the output files. The subdirectories '`noRB/`' and '`RB/`' are created by the program according to the value of the logical input variable `noRB` (default: `ndir="./simres"`);
- `comp` → if `TRUE` all the files produced by the execution of one cycle of `mltp.jc()` are automatically `tar`-ed and `gzip`-ed in a single file `allres_n(`*data*`)c(`*cycle*`)s(`*seed*`).tar.gz` (default: `comp=FALSE`);
- `njump` (old parameter) → an auxiliary variable for setting the step in the computation of the basins of influence of each mode.

Besides the files containing the input/output summary and the series of plots at each cycle, the following quantities may be stored in the corresponding `(`*quantity*`)_n(`*data*`)c(`*cycle*`)s(`*seed*`)` files by uncommenting the appropriate lines of the code when needed:

- `f1` and `f2` → the "linear" densities;
- `nu1` and `nu2` → the points of the grid (these are stored in the correspondent linear density file);
- `z` → the log-posterior surface.
- `sppts` → the number (`spptd$npeaks`, `sppts$npits`, `sppts$nspts`) and coordinates (`sppts$coordpeaks[1:3, 1:sppts$npeaks]`, `sppts$coordpits[1:3,1:sppts$npits]`, `sppts$coordspts[1:3,1:sppts$nspts]`) of the peaks (modes), pits and saddle points of `z`. The modes are sorted in decreasing levels and pits and saddle points are only calculated for a faster computation of the basins;
- `basin` → the contour lines associated to each mode: `basin[1:sppts$npeaks,1]` contains the levels, `basin[1:sppts$npeaks,2]` contains the series of abscissae and `basin[1:sppts$npeaks,3]` the series of ordinates;
- `mapust` → the coordinates of the particles issue of the last iteration (not resampled);
- `wmode` → the total weight associated to each mode;

By default, only `iosum_*`, `sim_*.ps`, `sppts_*` and `basin_*` are stored.

3 New Functions

3.1 `findspecialpoints(nu1,nu2,z,gridsiz)`

This function computes the positions of the log-posterior surface peaks, which locate its modes, and those of pits and saddle points, whose levels help in finding the basins. It uses the function `turnpoints()` of the package `pastecs` to locate the turning points of all the `gridsiz` z-lines in vertical direction, then it selects the ordinates of all the turn points found and locate the turn points of the corresponding horizontal z-lines. This gives the coordinates of the extrema. It returns the list `sppts`, which contains the numbers `$npeaks`(`$npits`)(`$nspts`) and coordinates `$coordpeaks`(`$coordpits`)(`$coordspts`) of the modes/pits/saddle points ordered by decreasing levels.

In some cases, an odd number of modes is found even in symmetrical cases, because one of the minor modes is "absorbed" by a nearest higher mode, while its symmetrical is not. This is probably due to rounding errors in the function `turnpoints()` of library `pastecs` and it has no consequence on the final result. In fact the absorbed mode generally attracts no or very little particles.

3.2 `findcontours(nu1,nu2,z,modes,njump)`

This function computes the largest possible contour line encircling a single mode. I have been trying various methods. The first method I tried was: starting from the highest mode and a given initial level (the level of the second highest peak), contour lines are computed, a check is made on the presence of only one mode inside each contour line and then the level is lowered. This is done iteratively until more modes are contained in a single curve. The scanning of the surface can be refined by increasing the value of `njump`. The scanning resolution is inversely proportional to

`jump=njump*abs(max(z)-min(z))/(10**round(log(abs(max(z)-min(z)),10)))`.

This method was dropped because the choice of `njump` and the way of computing `jump` were rather tricky: the scanning resolution had to be low enough for a faster computation but not too low otherwise some problems might arise, such as concentric contours. Moreover I found that a more efficient method than the rough scanning was that of setting a unique value for the level (`livello=(min(sppts$coordpeaks[3,])-abs(min(z)))/90`), computing the contour lines and checking which mode was in each of the curves. If there was more than one mode in a single contour, this method attributed the contour to the highest and no basin to the others.

In order to be less *ad hoc* and more precise, the actual method uses the levels of pits and saddle points in decreasing order to compute contour lines. It then makes the usual check on the modes (see above), by means of the function `inout()` of the package `splancs`, which tests if a given point is contained in a polygon.

3.3 `countpart(nsimu,modes,basin,mapust,w)`

This function calculates the number of particles contained in the basin of influence of each given mode and consequently associates a total weight to it. It uses the function `inpip()` of the library `splancs` which selects points inside a polygon.

4 The convergence criteria

While in the original version, the parameter `niter` set a fixed value for the number of iterations, in this version the evolution stops when a certain kind of convergence is achieved. Of course, if this is not the case the program stops when a maximum number of iterations (`niter=100` in a new role) is attained, since such a high number of iterations would be uninteresting from the point of view of the results and the efficiency of the algorithm.

The convergence may be defined on the basis of three different quantities:

Proposals weights. It is based on the absolute value of 1 to 4 (depending on the value of `howmany`) random walk proposal weights variation rate $\left|\frac{\Delta p_k}{p_k}\right|_{\mathtt{i}}$, $k = 1, \ldots, 4$ at iteration `i`.

A very frequent feature of the proposals weights evolutions in the non Rao-Blackwellized case is that two of them attain an "absorbant" state (the zero value) and stuck to it, while the other two reach a limit cycle, that is they keep on fluctuating in a symmetric way. This may be caused by the effective presence of a limit cycle or may be an effect of discretization. In this case a continuous system with the same law of evolution could attain a fixed point. To bypass this inconvenient and check if a limit value (fixed point) can be attained, the nonlinear difference equation which controls the evolutions of the weights:

$$\mathbf{p}_{i+1} = f(\mathbf{p}_i) \tag{1}$$

can be changed into:

$$\mathbf{p}_{i+1} = (1-\beta)\mathbf{p}_i + \beta f(\mathbf{p}_i)\,. \tag{2}$$

The two equations are the same for $\beta = 1$, while for $\beta << 1$ they attain the same fixed point (if any), but equation (2) converges much faster.

Entropy. Based on the absolute value of the Shannon's entropy variation rate $\left|\frac{\Delta s}{s}\right|_{\mathtt{i}}$ at iteration `i`, where:

$$s(w) = -\sum_{j=1}^{\mathtt{nsimu}} w_j \ln(w_j)\,,$$

and w_j, $i = j, \ldots,$ `nsimu`, are the renormalized importance weights.

ESS. Based on the absolute value of the Effective Sample Size (ESS) variation rate $\left|\frac{\Delta\ \mathrm{ESS}}{\mathrm{ESS}}\right|_{\mathtt{i}}$ at iteration `i`, where:

$$\mathrm{ESS}(w) = \frac{1}{1+\frac{\sigma_w^2}{\mu_w^2}} \approx \frac{1}{1+\frac{s_w^2}{\bar{w}^2}}$$

In all cases, the convergence depends on an overall parameter `tau` which set the threshold value for the variation rates, that is, the convergence is attained if

$$\left| \frac{\Delta \mathrm{CI}_i}{\mathrm{CI}_i} \right| \leq \texttt{tau}\,, \tag{3}$$

where CI stands for one of the convergence indicators p_k, s or ESS. One can strengthen the convergence condition by imposing 3 on several consecutive iterations.

5 Log-posterior Surface Asymmetry

The D-kernel PMC algorithm "fails" when the mode that ends up with the maximum weight is not the highest nor symmetrical to the highest. There must be of course many ways of defining the asymmetry of the log-posterior surface. A simple and intuitive index can be the ratio of the level difference between the highest (h) and the "heaviest" (H) mode to the log-posterior surface range, namely:

$$\alpha = \frac{|z_h - z_H|}{\Delta z}\ . \tag{4}$$

Since the log-posterior surface is always shifted to have the highest mode at the null level, this index is basically the level ratio of the heaviest mode to Δz. Notice that, for some sets of parameters, the z-surface diverges at some points of the (ν_1, ν_2)-domain. To by-pass the inconvenient, I artificially set the level at those points equal to $z_{\min}$.