Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Mateusz Łącki

Nr albumu: 39885

Properties of the Parallel Tempering algorithm.

Praca magisterska
na kierunku MATEMATYKA

Czerwiec 2013

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data

Podpis kierującego pracą

Oświadczenia autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data

Podpis autora pracy

Abstract

Parallel Tempering (PT) is an extension to standard Metropolis-Hastings (MH) algorithm for simulating samples from a given distribution. The standard MH algorithm with high probability remains a local algorithm, in the sense that samples are drawn from some local probability mass cluster. This is highly problematic in case of multimodial distributions that frequently appear in applications, i.e. in bayesian hierarchical models. The PT algorithm is one of most celebrated possible solutions to that problem.

Here we present a newly developed template for running simulations written in the R statistical programming language. Being highly modular it provides a general tool for performing simulations with many user provided state-spaces.

Słowa kluczowe

Parallel Tempering, Metropolis-Hastings, Stochastic Simulations, R

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.2 Statystyka

Klasyfikacja tematyczna

ACM I.6.7.

Tytuł pracy w języku angielskim

Properties of the Parallel Tempering algorithm.

Contents

List of Figures

Introduction

The Parallel Tempering algorithm is an extension to standard Metropolis-Hastings algorithm whose aim is to draw samples from a given target distribution. The Parallel Tempering algorithm is also known as the Replica Monte Carlo Simulation. The principle aim of the algorithm is to approximate the integrals

$$\mathcal{E}g(X) = \int_\Omega g(x) \mathrm{d}\,\Pi(x),$$

where $X \sim \pi$, π is a measure, and g is some $\mathbb{L}^1(\Pi)$ function. In practical applications π has either a density function, or a probability function. The Metropolis-Hastings algorithm proceeds by creating a sequence of sample points, $\{X^{[i]}\}_{i=1}^\mathrm{N}$, from the limiting distribution of a certain operator, $\mathcal{Q}$, chosen so that its invariant distribution exists, is unique, and coincides with π. The approximation is then taken to be simply

$$\mathcal{E}g(X) \approx \frac{1}{\mathrm{N}} \sum_{i=1}^\mathrm{N} g(X^{[i]}).$$

The problem with the usual Metropolis-Hastings algorithm is that it has poor mixing abilities, so that the above approximations may be biased when π is multimodial. The Parallel Tempering provides a potential solution to this problem by enlarging the state space. Several copies of progressively modified operators are run on this space, the differences between them depending on one parameter only[1]. This parameter is traditionally called *the temperature*, because of the physical interpretion it had in the seminal work of Swendsen and Wang (1986).

The sample points created by individual operators are sometimes refered to as chains. The idea of simultaneous simulations is the first of two consituents of the Parallel Tempering, the other being that of interlacing results from different chains. The new distributions have their probability mass spread in regions around the clusters of probability of the target-measure in a less and less concentrated way. Consequently, simulations from these distributions explore more of the state-space at the cost of higher variance. The subsequent random interchange between samples generated from different chains enables the base-level chain, corresponding to the target distribution, to explore most of its structure resulting in simulations better estimates.

The applicability of the Parallel Tempering stems from the more and more widespread use of the Bayesian modeling. In fact, it is the need of calculating some the a posteriori distributions that in many cases has driven scientists to develop efficient sampling techniques in the first place. Nowadays the use of Bayesian models spreads from meteorology, to chemistry, from OCRs, to biogenetics. And in all of these fields it is very natural for the multimodial distributions at some point to appear.

---

[1]There are many similarities with the homotopy theory in this regard.

Chapter 1

Motivation

In this Chapter we motivate the use of the PARALLEL TEMPERING making use of an example widely-cited in the literature,see Baragatti *et al.* (2013) and Liang and Wong (2001).

As pointed out in the Introduction, the reason for using the PARALLEL TEMPERING is the multimodiality of the distribution of interest, π. In our simulations we assume to know the density function of π to be the following mixture of normal densities:

$$f(x) = \sum_{i=1}^{20} \frac{\omega_i}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(x - \mu_i)^{\text{t}}(x - \mu_i)}{2\sigma_i^2}\right), \tag{1.1}$$

where $\sigma_1 = \cdots = \sigma_{20} = 0.1$, $\omega_1 = \cdots = \omega_{20} = 0.05$, and where the means μ_i are given by

μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7	μ_8	μ_9	μ_{10}
2.18	8.67	4.24	8.41	3.93	3.25	1.70	4.59	6.91	6.87
5.76	9.59	8.48	1.68	8.82	3.47	0.50	5.60	5.81	5.40

and

μ_{11}	μ_{12}	μ_{13}	μ_{14}	μ_{15}	μ_{16}	μ_{17}	μ_{18}	μ_{19}	μ_{20}
5.41	2.70	4.98	1.14	8.33	4.93	1.83	2.26	5.54	1.69
2.65	7.88	3.70	2.39	9.50	1.50	0.09	0.31	6.86	8.11

Figure 1.1 inspects visually the density. One notices easily that different peaks get intermingled, as only 15 peaks are visibleout of 20 peaks are clearly visible. The peaks are usually seperated by at least an area of unit diameter of lowered probability mass.

The METROPOLIS-HASTINGS is known to be a local algorithm and the reasons for it will be exposed in Chapter 2. To motivate the use of the PARALLEL TEMPERING it is enough to say for now that sample points generated by the usual METROPOLIS-HASTINGS algorithm[1] will be more often drawn from the vicinity of modes. This in turn might give the impression that most of the probability is concentrated somewhere next to the starting points[2]. We shall refer to the percentages of iterations in the vicinity of a particular mode its sojourn time, postponing a more thorough definition till Chapter 4.

By the results that stem from the Ergodic Theory, the sojourn times for each component of the mixture given by Eq. 1.1 should converge to the probabilities of the modes' vicinity areas, ω_i, and so be roughly equal in the above-mentioned example. However, in practical terms, this convergence

---

[1] For introduction to the theory behind the Metropolis-Hastings algorithm consider Geyer (2012).
[2] We choose the starting points at random
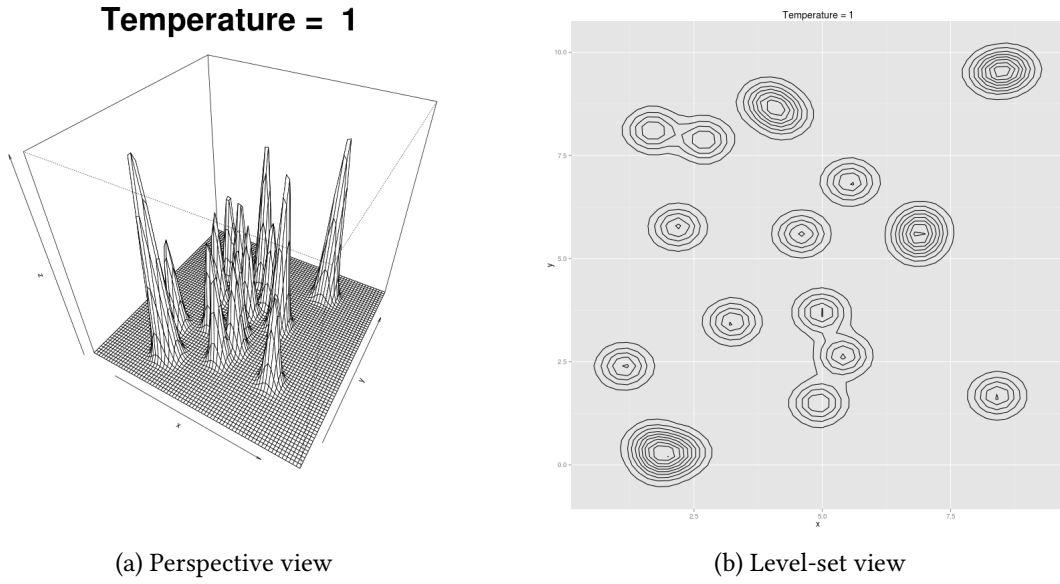
(a) Perspective view

(b) Level-set view

Figure 1.1: Toy-example density.

might be prohibitively long, way longer than what a typical user would consider fortunate to wait for. This is clearly a big problem, since in practical applications, as mentioned in the Introduction, one considers the approximation of the needed integral to be simply the mean evaluation of a function on the generated sample point and these approximations cannot possibly be good if we cut the procedures too quickly.

To investigate the METROPOLIS-HASTINGS algorithm's localness, we have run the METROPOLIS routine. Figures 1.2 and 1.3 show that the localness of the algorithm is persistent even with longer runs of the METROPOLIS-HASTINGS, the reasons for which will be mentioned in Chapter 2. Of course any type of inference based on results based on sample points as in Figure 1.3 would be totally not true.

Everything changes when one considers the PARALLEL TEMPERING instead, as seen in Figures 1.4 and 1.5. In Figure 1.4a one can visually ascertain himself that most of the STATE SPACE got explored. In Figure 1.4b one notices that only one mode was never explored. The simulation however was not a very long one. If one sets the number of iterations to be higher there are no modes that are unexplored in the studied example, as shown in Figure 1.5. In this picture one notices clearly that points accumulate in the regions with higher probability levels.

It is therefore important to verify in practice what can be achieved when using different types of the PARALLEL TEMPERING. In Chapter 2 we shall introduce several potential Swap Strategies that will result in different algorithm performance, which itself will be described in Chapter 4. The Strategies developed in Chapter 2 do not in any means exhaust all the potential ideas that a researcher might be interested in. In fact it may be true that a physicist could easily derive some criterion for swapping proposals based on the models natural interpretation. To this end a modularised template should be derived that would permit the user to check his own ideas without the need to reimplement all the details of the algorithm - such software has been prototyped and is presented in Chapter 3.

(a) Thousand steps...   (b) ...that lead nowhere.
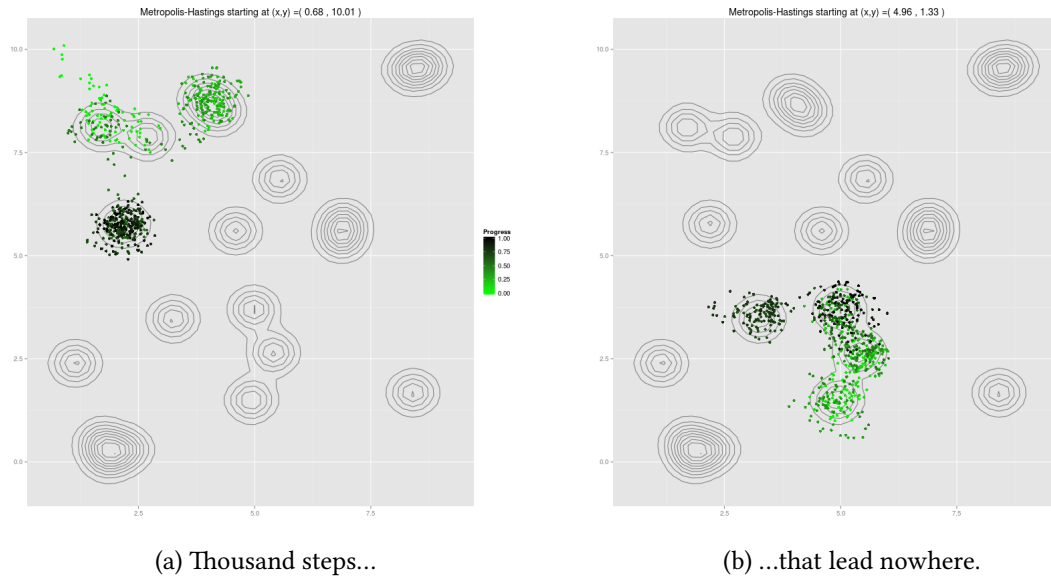
Figure 1.2: Thousand Iterations of Metropolis-Hastings algorithm - about 250 different points generated. Colour intensity increases as points get generated in more advanced stages of the algorithm. One notices easily that short runs of the algorithm do not explore the whole State Space.
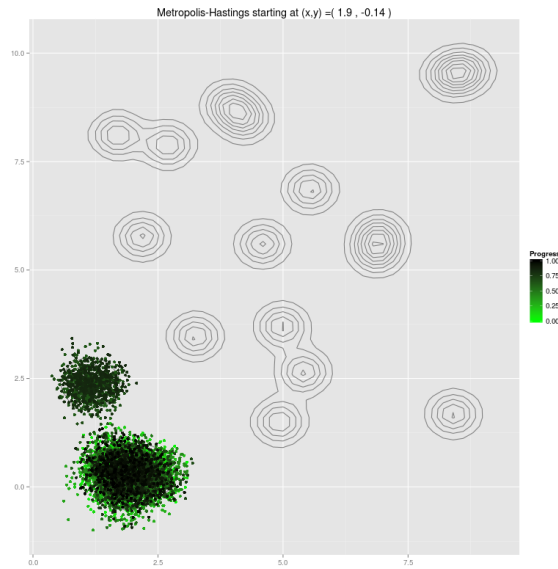
Figure 1.3: Ten Thousand Iterations of Metropolis-Hastings algorithm - about a quarter of which resulted in different points. Again the Metropolis-Hastings failed at exploring the whole State Space.
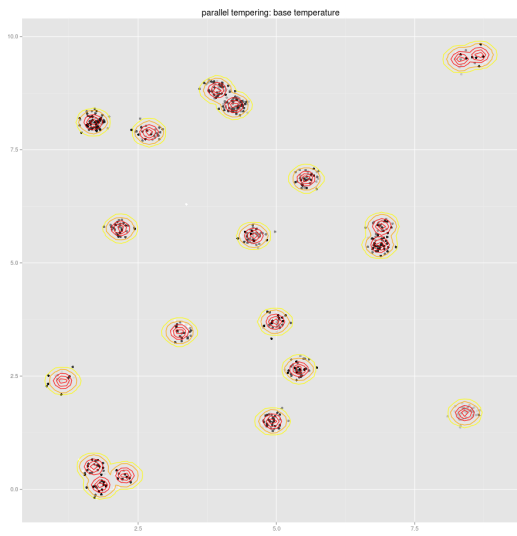
(a) Two thousand steps under Strategy 1....
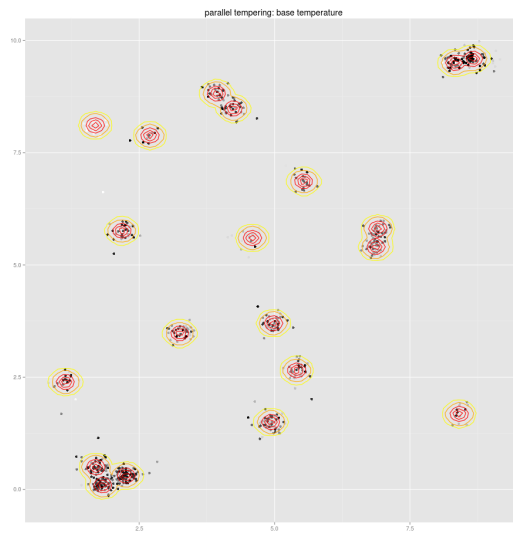
(b) Two thousand steps under Strategy 2.

Figure 1.4: Results of the Parallel Tempering when and Strategy 2 from Chapter 2 are applied. One notices immediately the qualitative difference between these examples and the results of the Metropolis-Hastings algorithm, as the State Space is explored more thouroughly. The results are obtainable under a minute on a modern computer equiped with one core processor unit, 1.5 GHz.
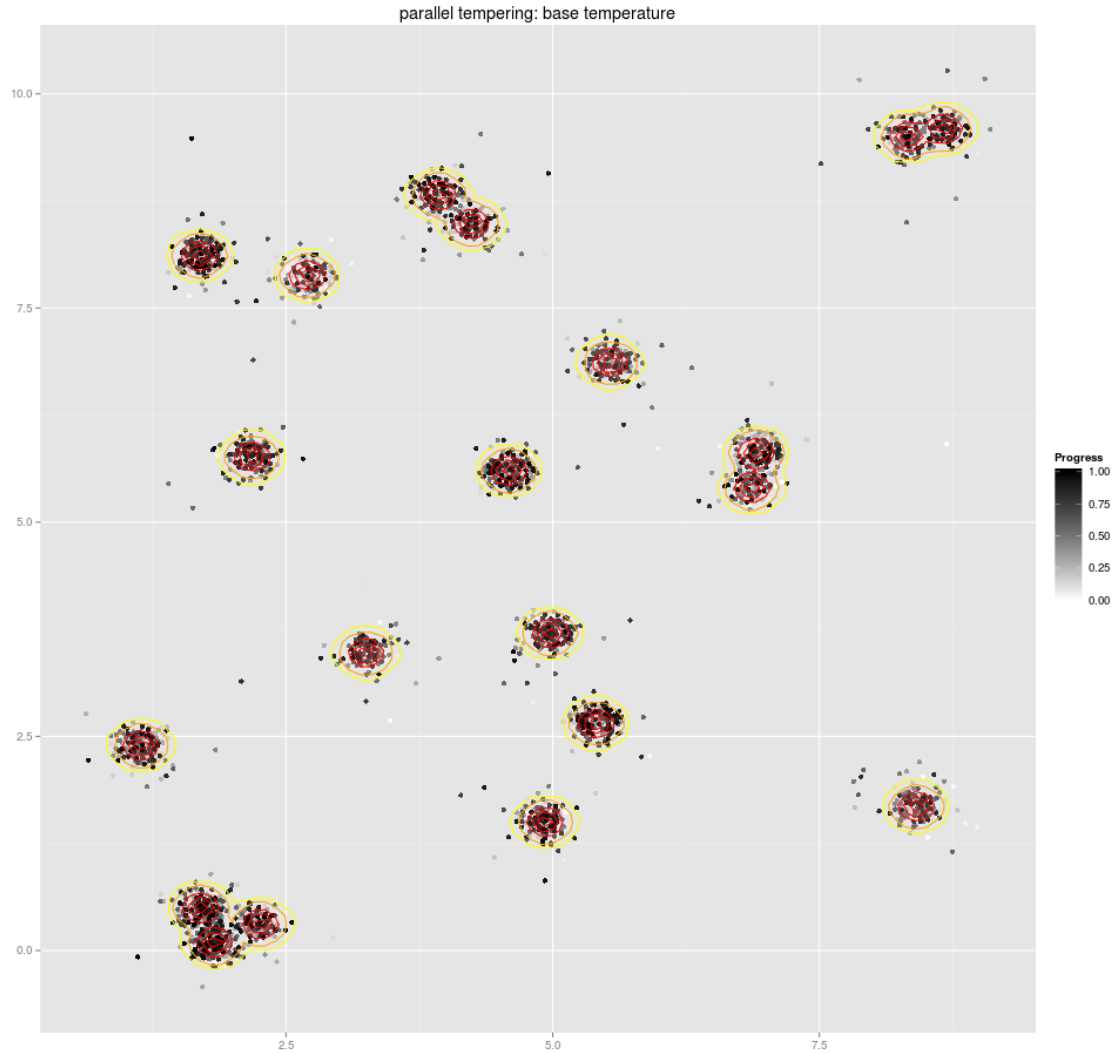
Figure 1.5: Ten Thousand Iterations of PARALLEL TEMPERING with 2500 iterations of burn-in, under STRATEGY III. About a quarter of random-walk steps were accepted. The simulation lasted about 140 seconds on a 8 years-old computer equiped with a one core computer. The isoquants encircle regions that contain one percent of overall probability mass (in yellow), five percent (orange), and twenty five, fifty and seventy five (all in red). The shading of sample points varies from white to black, getting darker as the simulation proceeds.

Chapter 2

Theory

In this chapter we shall expose the theoretical details of the Parallel Tempering. The following solutions are implemented in the prepared software, as described in Chapter 3.

Problem statement

Let $(\Omega, \mathfrak{F})$ be a measurable space. Ω — the State Space, a subset of a Polish space, and $\mathfrak{F}$ - countably-generated Borel subsets. Finally, denote the unit interval by $\mathcal{I} = [0, 1]$.

Problem Given a distribution $\Pi : \mathfrak{F} \mapsto \mathcal{I}$

      I absolutely continuous with respect to some measure on Ω, with a density π that we know to evaluate

      II density π known up to its proportionality factor

      III density π being multimodal

draw a sample $\{X_1, \dots, X_N\}$ from Π so that for any function $g \in \mathbb{L}^1(\Pi)$ one can approximate well the integrals with empirical means

$$\int_\Omega g \, \mathrm{d}\,\Pi \approx \frac{1}{N} \sum_{i=1}^{N} g(X^{[i]}).$$

The rather abstract setting might seem to be an overshot, because, as the reader suspects, the most obvious and natural choice would be that of $\Omega = \mathbb{R}^N$ and $\mathfrak{F}$ the corresponding Borel sets. That, however, is not always the case — for very often one encounters some discrete spaces as well. Moreover, the abstract setting underlines the modularity of the implementation of the algorithm itself, being described in CHAPTER.

Had it been that π was not multimodal then by using the usual Metropolis-Hasting algorithm one could have easily solved the Problem. Adding condition III, however, makes the use of that algorithm problematic because of its localness — a phenomenon well visualised in CHAPTER.

The Parallel Tempering

The Parallel Tempering algorithm is based on the idea of extending our attention to the product space $(\Omega^L, \mathfrak{F}^{\otimes L}, \pi_\beta)$ equiped with a specific Markov chain[1]. The main idea is that on each copy of Ω

---

[1]Here $\mathfrak{F}^{\otimes L} \equiv \underbrace{\mathfrak{F} \otimes \cdots \otimes \mathfrak{F}}_{\text{L times}}$ is the usual product sigma algebra.

the corresponding chain will encounter a slightly modified version of the density of interest π. In the usual PARALLEL TEMPERING one settles for

ASSUMPTION I $\pi_\beta \propto \pi^{\beta_1} \times \cdots \times \pi^{\beta_L}$,

where $\beta = (\beta_1, \ldots, \beta_L)$ are such that $1 = \beta_1 > \cdots > \beta_L > 0$. Their inversions, $T_i = \frac{1}{\beta_i}$, called temperatures, do form therefore a growing sequence of numbers. This naming convention finds its origin in the use of the PARALLEL TEMPERING in Statistical Physics where the T_i parameters had direct interpretation.

We do stress that the normalising constants for each π^{β_l} are not assumed to be known, as we might not even know it for π itself.

A Markov Chain[2] $X \equiv \{X^{[k]}\}_{k \geq 0}$ is then constructed on Ω^L. Its construction makes explicit use of two kernels that make direct reference to π_l^β — first one, being a standard RANDOM WALK, is responsible for the exploration of the STATE SPACE. The other, called the SWAP, takes care of communication between different subchains — for X can also be thought of as being a collection of coordinate subchains

$$X^{[k]} = (X_1^{[k]}, \ldots, X_L^{[k]}).$$

Observe that the first subchain corresponds to the solution of the PROBLEM.

The main idea behind the construction of chain X is that the subchains operating on tempered versions of π influence the subchains linked with lower temperatures by offering them their own accepted proposals. This should be highly beneficial, as more tempered chains will encounter less problems in accepting proposals from regions where the first chain would hardly ever stray.

To make this approach work at all we are still bound to make some other assumptions.

ASSUMPTION II The starting points $X^{[0]}$ are selected so that $\pi(X_l^{[0]}) > 0$ for $l \in \{1, \ldots, L\}$.

Obviously, what follows is that $\pi(X_l^{[0]})_k^\beta > 0$ as well. It is important, for otherwise, what will be demonstrated in the following sections, the algorithm would be dividing by 0 and break. Moreover, what follows from ASSUMPTION II is that the algorithm will never find itself in a point on which the density evaluates to zero.

The choice of modifying π by exponentiation, as in ASSUMPTION I, besides its simplicity[3], has considerable advantages in the analysis of the algorithm. For in the RANDOM WALK phase of the algorithm's step suppose a proposal y is generated by the lth chain from a region with less Π probability on it in comparison to the position x occupied in the last step, so that $\pi(y) < \pi(x)$. What follows,

$$\alpha_{\beta_l}(x, y) = 1 \wedge \left(\frac{\pi(y)}{\pi(x)}\right)^{\beta_l} = \left(\frac{\pi(y)}{\pi(x)}\right)^{\beta_l} > \frac{\pi(y)}{\pi(x)} = 1 \wedge \frac{\pi(y)}{\pi(x)} = \alpha_{\beta_1}(x, y),$$

which means that the probability of accepting such a proposal is larger on the more tempered chains. On the other hand, if $\pi(y) > \pi(x)$, then this probability remains the same, $\alpha_{\beta_1}(x, y) = \alpha_{\beta_k}(x, y)$. This assures that more tempered chains can make longer excursions into regions not so much Π-probable.

One cannot neglect one possibility though: if the support of π is not connected and the connected components are far away then it is highly improbable that the algorithm will give unbiased results because of poor mixing that cannot be fixed by exponentiation and some other technique should be

---

[2]For the algorithm step numbering we shall henceforth use the square brackets notation.
[3]For in general one could settle for some other kind of homotopic deformation.

used. We stress this point because in computer simulations one might tumble upon this problem because of the finite arithmetic − the rounding of numbers to zero.

As pointed out before, the update mechanism of the PARALLEL TEMPERING consists of two stages. Assume that the algorithm has reached n^{th} step, $X^{[n-1]}$. Following notation used in Miasojedow, Moulines and Vihola (2013a), we introduce two kernels that act subsequently on $X^{[n-1]}$

$$X^{[n-1]} \overset{M_{\Sigma,\beta}}{\rightarrow} \widetilde{X}^{[n-1]} \overset{\mathcal{S}_\beta}{\rightarrow} X^{[n]}.$$

We shall refer to $\mathcal{S}_\beta$ and $M_{\Sigma,\beta}$ as to the swap kernel and the random-walk kernel respectively[4]. These will be now presented in greater detail.

The RANDOM WALK Kernel

Take a point in the STATE SPACE, $x = (x_1, \ldots, x_L) \in \Omega^L$. To assign probabilities to the proposal of the next step, given x, we consider events $A_i \in \mathfrak{F}$. We shall assume that

ASSUMPTION III RANDOM WALK proposal generation occurs independently on all subchains

$$M_{\Sigma,\beta}(x, A_1 \times \cdots \times A_L) = \prod_{l=1}^{L} M_{\Sigma_l,\beta_l}(x_l, A_l).$$

The $M_{\Sigma_l,\beta_l}(x_l, A_l)$ corresponds to standard Metropolis kernel, as in Geyer (2012). To be more precise

$$M_{\Sigma_l,\beta_l}(x_l, A_l) \equiv \int_A \alpha_{\beta_l}(x_l, y_l) q_{\Sigma_l}(y_l - x_l) \mathrm{d}y_l + \mathbb{I}(x_l, A) r(x_l) \tag{2.1}$$

where

$$r(x_l) = \int [1 - \alpha_{\beta_l}(x_l, y_l)] q_{\Sigma_l}(y_l - x_l) \mathrm{d}y_l$$

is the probability of not accepting the proposal and the subprobability density of accepting the update [5] is

$$\alpha_{\beta_l}(x_l, y_l) \equiv 1 \wedge \frac{\pi^{\beta_l}(y_l)}{\pi^{\beta_l}(x_l)}. \tag{2.2}$$

To understand why M_{Σ_l,β_l} is given by Eq. 2.1 we note that the role of the kernel is to assign probability to the next step of the algorithm given that the previous one assumed value x[6]. The METROPOLIS-HASTINGS algorithm goes as follows[7]:

Algorithm 2.0.1 (METROPOLIS-HASTINGS).
Require: *$X^{[0]}$ such that $\pi(X^{[0]}) \neq 0$*
  procedure METRO(*$X^{[0]}$, N*)
    for *$n \in \{0, \ldots, N-1\}$* ***do***
      Draw proposal $Y \sim q(X^{[n]}, y)\mathrm{d}y$
      Draw $U \sim \mathcal{U}(0, 1)$
      if *$U \leq \alpha(X^{[n]}, Y)$* ***then***

---

[4] M as Metropolis.
[5] Also known as the Hastings ratio.
[6] So we abstract here from how the actual value x was obtained, being mysteriously revailed to us by Nature, under the guise of random number generator. In fact kernels and their duals play principal role in the Bayesian statistics.
[7] We omit the inverse temperatures for time being.

$$X^{[n+1]} \leftarrow Y$$
else
$$X^{[n+1]} \leftarrow X^{[n]}$$
end if
end for
end procedure

Therefore, one should assign probability to event $\{X^{[n+1]} \in A\}$ for all $A \in \mathfrak{F}$. But this is done, as

$$\int_A \alpha(x,y)q(x,y)\mathrm{d}\,y = \int_A \mathcal{P}\Big(U \le \alpha(x,y)\Big)q(x,y)\mathrm{d}\,y =$$

$$= \int_A \mathcal{P}\Big(U \le \alpha(x,Y)|Y=y\Big)q(x,y)\mathrm{d}\,y = \mathcal{P}\Big(U \le \alpha(x,Y), Y \in A\Big) =$$

$$= \mathcal{P}\Big(X^{[n+1]} \neq X^{[n]}, X^{[n+1]} \in A\Big)$$

where we assume Y to be drawn out of the proposal distribution and U to be independent of it. Setting $A = \Omega$ one notices that[8]

$$\mathcal{P}\Big(X^{[n+1]} = X^{[n]}\Big) = \int [1 - \alpha(x,y)]q(x,y)\mathrm{d}\,y = r(x)$$

− precisely as claimed.

In the above equations the proposal distribution being used, following notation from Geyer (2012), is $q(x_l, y_l) = q_{\Sigma_l}(y_l - x_l)$, where q_{Σ_l} is the density of $\mathcal{N}(0, \Sigma_l)$.

The simple form of Eq. 2.2 stems from proposal distribution symmetry,

$$q(x_l, y_l) = q(y_l, x_l).$$

The choice of α is motivated by assuring the self-adjointness of the $M_{\Sigma,\beta}$ operator, which in turn implies that distribution Π is preserved, so that

$$\Pi M_{\Sigma,\beta} = \Pi. \tag{2.3}$$

As indicated by Miasojedow *et al.* (2013b), the implementation of $M_{\Sigma,\beta}$ consists of separate simulation of M_{Σ_l,β_l} for each of the coordinate chains, $X_l^{[n-1]}$.

The swap kernels

Now we address the question of how to interlace the independent chains generated with kernel $M_{\Sigma,\beta}$. We want to swap the generated states so that chains with higher temperature influence the regions of lower temperatures. Being more precise, the swapping gives some possibility to the chains with lower temperatures to find themselves in states that would otherwise be very unlikely to happen, the path leading to them being very unlikely to occur - these paths are rendered more probable for the chains with higher temperature. The swaps are made at random so that the probability of swaps takes into account some properties of π evaluated on the last points from all the chains.

Before passing to detailed description of different ways the swaps might be made, which we call STRATEGIES, let us first describe the properties of the swap kernel $\mathcal{S}_\beta$ common to all STRATEGIES. In all cases, we rely on the following assumption

---

[8]Under the sensible assumption that the proposal distribution does not have any atom in x − this is usually automatically assumed by considering a distribution that is absolutely continous w.r. to Lebesgue measure.

ASSUMPTION IV At one step of the algorithm, there will be only one possible swap between a chosen at random pair of coordinates.

The restriction here is not so much in the number of swaps per step. What seems to be more relevant is rather the constraint to a particular subset of the group of all permutations — for one could certainly envisage that any random permutation could take place. However, it simplifies the interpretation of STRATEGIES that are to be described.

Let us denote the swap operator by S_{ij}, so that

$$S_{ij}x = (x_1, \ldots, x_{i-1}, x_j, x_{i+1}, \ldots, x_{j-1}, x_i, x_{j+1}, \ldots, x_L).$$

We require $\mathcal{S}_\beta(x, \circ)$ to be a measure concentrated on the set of all possible pairs of swaps between coordinates of $x = (x_1, \ldots, x_L)$, i.e. on the set $\mathfrak{S}_x \equiv \{S_{ij}x : i < j\}$. We note that the proposal measure for the swap kernel $\mathcal{Q}$ is of the following form

$$\mathcal{Q}(x, A) \equiv \sum_{i<j} p_{ij}(x)\mathbb{I}_A(S_{ij}x)$$

Then, the swap kernel would be of the following form, for any $x \in \Omega^L$ and A

$$\mathcal{S}_\beta(x, A) = \int_A \alpha_{\mathrm{swap}}(x, y)\mathcal{Q}(x, \mathrm{d}\,y) + r(x)\mathcal{I}(x, A)$$

where $r(x) = 1 - \int_{\Omega^L} \alpha_{\mathrm{swap}}(x, y)\mathcal{Q}(x, \mathrm{d}\,y)$. Plugging $\mathcal{Q}$ permits us to write

$$\int_A \alpha_{\mathrm{swap}}(x, y)\mathcal{Q}(x, \mathrm{d}\,y) = \sum_{i<j} p_{ij}(x) \int \alpha_{\mathrm{swap}}(x, y)\mathbb{I}_A(y)\delta_{S_{ij}x}(\mathrm{d}\,y)$$

$$= \sum_{i<j} p_{ij}(x)\alpha_{\mathrm{swap}}(x, S_{ij}x)\mathbb{I}_A(S_{ij}x),$$

so that finally

$$\mathcal{S}_\beta(x, A) = \sum_{i<j} p_{ij}(x)\alpha_{\mathrm{swap}}(x, S_{ij}x)\mathbb{I}_A(S_{ij}x) + \left(1 - \sum_{i<j} p_{ij}(x)\alpha_{\mathrm{swap}}(x, S_{ij}x)\right)\mathcal{I}(x, A),$$

where

$$\alpha_{\mathrm{swap}}(x, y) = \frac{\pi_\beta(\mathrm{d}\,y)\mathcal{Q}(y, \mathrm{d}\,x)}{\pi_\beta(\mathrm{d}\,x)\mathcal{Q}(x, \mathrm{d}\,y)} \wedge 1$$

is the Radon-Nikodym derivative of two measures. It's calculated with respect to measure $\mu(\mathrm{d}\,x, \mathrm{d}\,y) \equiv \pi(\mathrm{d}\,x)\mathcal{Q}(x, \mathrm{d}\,y)$. The measure that is being differentiated is μ composed with the exchange coordinates operation, $R(x, y) = (y, x)$[9]. So finally $\alpha_{\mathrm{swap}} \equiv \frac{\mathrm{d}\,\mu \circ R^{-1}}{\mathrm{d}\,\mu}$.

Observe that thanks to the proposal's support finiteness we actually get

$$\alpha_{\mathrm{swap}}(x, y) = \frac{\pi_\beta(y)\mathcal{Q}(y, x)}{\pi_\beta(x)\mathcal{Q}(x, y)} \wedge 1.$$

Now, since $y \in \mathfrak{S}_x$, the tagetted π_β is a tensor of measures, $Q(x, S_{ij}x) = p_{ij}(x)$, and $Q(S_{ij}x, x) = p_{ij}(S_{ij}x)$ [10], we get finally

---

[9]Which is obviously measurable in the appropriate sense.

[10]Here we pass from measure notation to probability function notation, $\mathcal{Q}(x, \{S_{ij}x\}) = \mathcal{Q}(x, S_{ij}x)$. We can do it because $\mathcal{Q}$ is purely atomic given x.

$$\alpha_{\text{swap}}(x, S_{ij}x) = \left[\left(\frac{\pi(x_j)}{\pi(x_i)}\right)^{\beta_i - \beta_j} \frac{p_{ij}(S_{ij}x)}{p_{ij}(x)}\right] \wedge 1$$

In our computer simulations several swapping strategies have been tested. For instance,

STRATEGY I $p_{ij}(x) \propto \frac{\pi(x_j)}{\pi(x_i)} \wedge \frac{\pi(x_i)}{\pi(x_j)} = \exp\left(-\left|\log(\pi(x_j)) - \log(\pi(x_i))\right|\right).$

In this particular the proportionality is in fact equality, because the above expression is transposition symmetric and this considerably simplifies the statistical sum analysis. In general, we want $p_{ij}(x) \propto h(x_i, x_j)$ for some function h. Thus,

$$\frac{p_{kl}(S_{kl}x)}{p_{kl}(x)} = \frac{\frac{h(x_l, x_k)}{A(\tilde{x}_{kl}) + f(x_k) + g(x_l) + h(x_l, x_k)}}{\frac{h(x_k, x_l)}{A(\tilde{x}_{kl}) + f(x_l) + g(x_k) + h(x_k, x_l)}} = \underbrace{\frac{h(x_l, x_k)}{h(x_k, x_l)}}_{\text{'direct effect'}} \times \underbrace{\frac{A(\tilde{x}_{kl}) + f(x_l) + g(x_k) + h(x_k, x_l)}{A(\tilde{x}_{kl}) + f(x_k) + g(x_l) + h(x_l, x_k)}}_{\text{'distribution effect'}},$$

where $\tilde{x}_{kl}$ is x without its kth and l^{th} coordinates, $f(x_l) = \sum_{i<l, i \neq k} h(x_i, x_l)$, and $g(x_k) = \sum_{j>k, i \neq l} h(x_k, x_j)$. As we see it is fairly straightforward to compare the above expression to 1 if the 'distribution effect' equals 1, and difficult had it been the other case. To assure this happens, one can settle for a function h that is transposition invariant, $h(x, y) = h(y, x)$. This is certainly the case of STRATEGY I. In this strategy therefore we are promoting exactly swaps between coordinates whose proposals have relatively the same π density values, i.e. $\pi(x_j) \approx \pi(x_i)$.

STRATEGY II $p_{ij}(x) \propto \frac{\pi(x_j)}{\pi(x_i)} \wedge 1 = \exp\left(-\left[\log(\pi(x_j)) - \log(\pi(x_j))\right]\right) \wedge 1.$

This strategy breaks the symmetry of the previous one rendering the evaluation of the effect on the α_{swap} challenging, if not impossible. The main idea behind this strategy was to

The subsequent strategies are again functions of the fist strategy.

STRATEGY III $p_{ij}(x) \propto \left(\frac{\pi(x_j)}{\pi(x_i)} \wedge \frac{\pi(x_i)}{\pi(x_j)}\right)^{|\beta_i - \beta_j|} = \exp\left(-|\beta_i - \beta_j| \times \left|\log(\pi(x_j)) - \log(\pi(x_i))\right|\right).$

This strategy permits us to soften a bit the requirement that $\pi(x_j) \approx \pi(x_i)$. This effect is strengthened for coordinates that are similarly tempered, i.e. where $\beta_i - \beta_j \approx 0$. Swaps between adjacent chains will be therefore more probable.

One can also use a strategy that incorporates some distance measure between points, ρ being any quasi-metric. STRATEGY IV is a good example of such procedure - the more distant the points, the less probable it is for them to get swapped.

STRATEGY IV $p_{ij}(x) \propto \left(\frac{\pi(x_j)}{\pi(x_i)} \wedge \frac{\pi(x_i)}{\pi(x_j)}\right)^{\frac{|\beta_i - \beta_j|}{1+\rho(x_i, x_j)}} = \exp\left(-\frac{|\beta_i - \beta_j| \times |\log(\pi(x_j)) - \log(\pi(x_i))|}{1+\rho(x_i, x_j)}\right).$

Finally, for purposes of comparisons with Baragatti *et al.* (2013), we include also strategies that are STATE SPACE independent, i.e. the probability of swaps does not depend on the evaluations of the density π in the sample points. Such naïve strategies include

STRATEGY V $p_{ij} = \frac{2}{L(L-1)}$

and

STRATEGY VI $p_{ij} = \frac{1}{L-1}\mathbb{I}_{\{|i-j|=1\}}$

amounting to choosing uniformly among all possible swaps and all neighbouring-in-temperature swap respectively.

Chapter 3

Implementation

In this chapter we will present the implementation of the PARALLEL TEMPERING algorithm, as described in Chapter 2.

The analysis of the algorithm in all its complexity called for the development of a more structurised template for carrying out numerical computations. The PARALLEL TEMPERING algorithm, being an extension to the METROPOLIS-HASTINGS algorithm, does naturally share many similarities its structure. Both algorithms can have very abstract formulations and can be carried out theoretically on any countably- generated measurable space. Practical considerations would never go that far. However, one cannot, in principle, discard the use of Stochastic Simulations in solving many real-life modelling problems such as, for instance, the Bayesian model selection, or Ising-type modelling of crystallic structures. The mulitude of potential applications calls for a reasonably flexible implemention of the PARALLEL TEMPERING algorithm. To overcome these problems, the object-oriented paradigm was used in this work, so that the STATE SPACE and the ALGORITHM were two separate entities.

Having realised the need for modularity, it seemed natural to take the whole idea one step further and develop a general template for Metropolis-Hastings-like simulations, going way beyond the idea of a task-specific computer programme. For there are other potential developments of the METROPOLIS-HASTINGS algorithm that are being used, all baring strong similarities in stuctures of these algorithms. Our template's goal was therefore to provide basic building blocks used for simulations, leaving the practitioners concentrate on the analysis. The template is codenamed STOCHASTICSIMULATIONS and that shall be the **R** package name when shipped.

R offers several implemented meta-structures that enable basic object oriented programming techniques. Among these there are the S3 classes, S4 classes, and the most recent Reference Classes. The implementation the template, was carried out using Reference Classes. This choice was dictated by several reasons. First, only S4 classes and Reference Classes offer the possibility of basic type verification[1]. This assured that no serious errors were introduced in the implementation phase and that users cannot provide absurd input for the algorithm. Moreover, Reference Classes are claimed to be the only implementation of objects in **R** that use passing arguments by reference, and a priori could saved some time in the calculations on unnecessary object copying[2]. Finally, Reference Classes are considered to be highly compatible with **C++** precompiled programmes being called from **R**. This is a clear advantage, as the future shipments of STOCHASTICSIMULATIONS are planned to be implemented in **C++**, so that users can simulate faster that the whole programme was more memory efficient.

---

[1]Still far from the **C++** standards though.

[2]No serious differences in execution time were actually spoted when comparing both the functionally programmed prototype with the object-oriented final version, however.
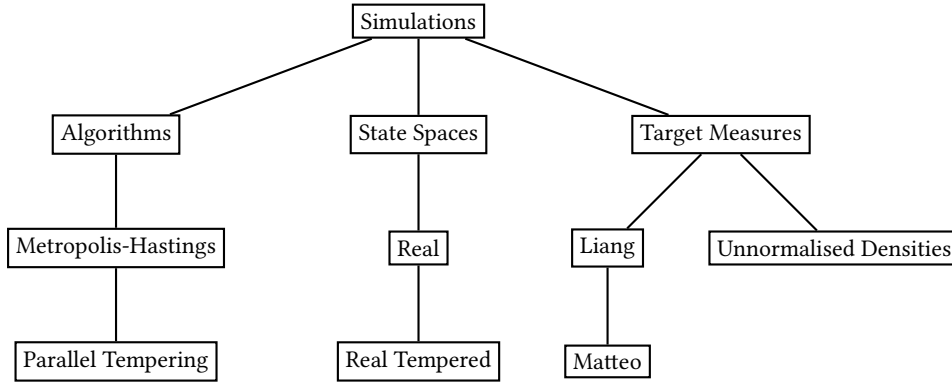
Figure 3.1: Current operational entity-relations diagram.

3.1. Division into objects

To assure real modularity of our template, more divisions were proposed than only the one mentioned at the beginning of this chapter. The overall structure is represented in Figure 3.1.

We already mentioned the separation of the ALGORITHM from the STATE SPACE. It is useful to think of the ALGORITHM as of a decision-maker. Its role is restricted to making decisions on acceptance or rejections of the proposals generated by the STATE SPACE and then on giving orders to the STATE SPACE to perform subsequent actions. The information on which the ALGORITHM bases its decisions depends on the points from the STATE SPACE only indirectly, via the evaluations of densities[3] at the proposal points. For in phase of the Random Walk the decision is based on the evaluation of

$$\alpha_\beta(x_l, y_l) = 1 \wedge \frac{\pi^{\beta_l}(y_l)}{\pi^{\beta_l}(x_l)} = R\Big(\pi(y_l), \pi(x_l)\Big),$$

and in the Random Swap phase — upon evaluation of one of the strategies, and additionally, in case of Strategy IV, of the quasi metric

$$p_{ij}(x) = S\Big(\pi(x_i), \pi(x_j), \underbrace{\rho(x_i, x_j)}_{\text{Random Swap only}}\Big)$$

A third candidate for an entity thus appears - namely an object whose methods would serve to measure some characteristic of the STATE SPACE sample points for the use in the decision-making of the ALGORITHM. We called this entity the TARGET MEASURE.

The very idea behind the TARGET MEASURE entity is that it should serve as a container for user-defined probability function or a density together with every additional data-structure required for its evaluation. Moreover, in case the user was interested in testing the PARALLEL TEMPERING algorithm's potential against some analytically tractable toy-example, or in case this example was easy to simulate using a simpler or potentially more efficient technique than MCMC, the TARGET MEASURE entity would be the place to store any additional method of carrying out calculations regarding this particular distribution. For example, in our simulations (confront Chapter simulationsAndResults) we could additionally provide efficient methods for quantile simulations and evaluates of the real distribuant. Implementations of these functions were collected as methods of the appropriately named instatiation of the TARGET MEASURE structure.

---

[3]In the discrete case – probability functions

18

The reason for separating the Target Measure from the State Space is again dictated by the modularity requirement. Totally different models can be modelled on the same State Space, the only difference being the way the probability is assigned. It is therefore natural to define an entity whose task will be to serve as a warehouse for sample points, together with efficient methods or reading them, and in the end – presenting the results to the user. These things, to some extent, can be done for any probability measure defined by the user.

The Object Oriented paradigm gives the programmer also the tool of Inheritance. Its role is to assure no code getting copied when several programmes share the same structure and differ only at some minor implementation details. In our implementation we have noticed that the Parallel Tempering algorithm, being an extension to the Metropolis-Hastings, differs only in the appearance of the swap stage and its methods, the Random Walk stage being almost the same[4]. The Parallel Tempering algorithm inherits from the Metropolis-Hastings methods that are used for generating Random Walk proposals and updating the stored density evaluations. The Metropolis-Hastings in its turn inherits fields from the Algorithm entity — a virtual class, whose role is to store variables that can be used in any algorithm whatsoever, e.g. containing information on the user-supplied maximal number of iterations, or binary information on whether the calculations are done in the burn-in period, this being important for efficient storing of simulated points, or whether the simulation has been already carried out which is important for proper visualisation of the results.

Some of the entities must contain pointers to other entities in order to be able to call methods of other entities when needed[5]. Because of certain Reference Classes limitations[6], the Algorithm has a pointer on the State Space, which in its turn has a pointer on the Target Measure. Under different circumstances, different realisations of objects will appear as the Algorithm, the State Space, and the Target Measure. For instantance, if the user wanted to carry out standard Metropolis-Hastings calculations on the three-dimensional euclidean space and check how it copes with the density function he provided, then the programme would have to first initialise the Unnormalised Density being the instantiation of the Target Measure entity. Then, initialise the Real space, as an instatiation of the State Space, and create a pointer on the Unnormalised Density. Finally, it would have to initialise the Metropolis-Hastings being the operational Algorithm, and set the pointer on the previously initialised Real space. To automate this chain of initialisations, a controller entity was established under the name of Simulation. The Simulation greets the user, checks whether he is not interested in one of the preprogrammed examples of how different algorithms behave (compare to Chapter simulationsAndResults), and then instantiates all different entities using the user-provided informations.

For reasons of user-friendliness, additional functions were written, whose names follow a simple convention. The name of the algorithm can be composed of up to two substings: the first substring being the name of the algorithm used, and the second - the name of some standard State Space. The functions serve as wrappers for more general Reference Class constructor of the Simulation object. Continuing on the above-mentioned example, the user would simply had to call[7]

```
1    Example <- metropolisReal(
2      n = 1000,
3      spaceDim = 2,
4      targetDensity = userDefinedDensity
5    )
```

---

[4]The difference being that it is called for several chains instead of one.

[5]Remember the Algorithm being compared to a decision maker.

[6]As pointed out before, Reference Classes are relatively new, and, as such, poorly documented. With trial and error it has been checked, that this particular choice of entity-nesting simply works.

[7]The convention is known under the name of camelCase.

to initialise the simulation of the METROPOLIS-HASTINGS algorithm on the REAL state space, with USERDEFINEDDENSITY being an R-implemented unnormalised density function[8]. If he was to use the PARALLEL TEMPERING algorithm approach, the initialisation code would be

```
Example <- parallelemperingRealtempered(
  n = 1000,
  spaceDim = 2,
  targetDensity = userDefinedDensity
)
```

The presented object structure is far from final. Hopefully it will evolve in time, trying to match the needs of various users. We plan to the separate the swap distributions from the Parallel Tempering entity, giving the user the possibility to check swapping strategies tailored to his needs. Moreover, the existance of the REAL TEMPERED entity stems from purely technical reasons of proper visualisation. The optimal solution would be to create a special controller class for visualisation that would take into account differences in the information presentation of results created through the use of different algorithms and provide some basic solutions.

To guarantee user-friendliness, the most common STATE SPACE— the multidimensional Euclidean space — has been provided, so that the user can carry out computations specifying only the required minimum - the density function of the measure of interest. With future releases of the software, more standard State Spaces are scheduled for implemention as well, giving the practitioners experience the potentials and drawbacks of Metropolis-Hastings-like stochastic simulations.

3.2. Functions and Methods

A detailed enumeration of headers of the implemented methods together with brief description can be found in Appendix A.

3.3. Two-dimensional Колмогоров-Смирнов distance

In order to check how different strategies approximate the toy-example distribution described in the next chapter, one had to settle for some kind of criterion. Recall that the general PROBLEM is to approximate an integral

$$\int_\Omega g \, \mathrm{d}\Pi \approx \frac{1}{\mathrm{N}} \sum_{i=1}^{\mathrm{N}} g(X^{[i]}).$$

One of the principal results in the Ergodic Theory is that one should observe weak convergence of empirical measures to the TARGET MEASURE Π on almost all trajectories. The sample measures are simply taken to be

$$\Pi_{\mathrm{N}}(A) \equiv \frac{1}{\mathrm{N}} \sum_{i=1}^{\mathrm{N}} \mathbb{I}_A(X^{[i]}),$$

for $A \in \mathfrak{F}$. The weak convergence means that for any function $f : \Omega \mapsto \mathbb{R}$ bounded and continous and a fixed $\omega \in \Omega$ one observes

---

[8]In future versions C++ calls will be usable.

$$\Pi_{\mathrm{N}} f(\omega) \equiv \frac{1}{\mathrm{N}} \sum_{i=1}^{\mathrm{N}} \mathbb{I}_A\left(X^{[i]}(\omega)\right) f\left(X^{[i]}(\omega)\right) \xrightarrow{\mathrm{N}\to\infty} \int_{\Omega} f(x)\Pi(\mathrm{d}\,x),$$

and we denote it by $\Pi_{\mathrm{N}} \rightharpoonup \Pi$. The above-mentioned result is simply

$$\mathcal{P}\left(\Pi_{\mathrm{N}} \rightharpoonup \Pi\right) = 1.$$

If so, it is obvious that if the theorem holds, then on almost all trajectories one would also observe

$$\mathcal{P}\left(\sup_{x\in\mathbb{R}^{\mathrm{K}}}\left|F(x) - \hat{F}_{\mathrm{N}}(x)\right| \to 0\right) = 1,$$

where $\hat{F}_{\mathrm{N}}$ is the Empirical Cumulative Distribution Function — **ECDF**. If so, then if the values of the trajectory-based **ECDF** were far away from the true distribuant, the approximation would be of poor quality. Therefore, if one is to compare different algorithms in their task to solve the Problem, as we do when applying different Swap Strategies, then a natural measure for such comparison is exactly the evaluation of the Колмогоров-Смирнов distance

$$d_{KS}\left(F, \hat{F}_{\mathrm{N}}(\omega)\right) \equiv \sup_{x\in\mathbb{R}^{\mathrm{K}}}\left|F(x) - \hat{F}_{\mathrm{N}}(x)(\omega)\right|.$$

But since it is trajectory-dependent, then a better way would be to carry out several hundreds of simulations and check for the differences in the mean values of the Колмогоров-Смирнов distance and its variance and use these quantities as proxies for our test. This is of course a heuristics, for the reasoning to be precise one would have to consider if the process has the huge bulk of its probability centered around the mean function.

The principal problem with the KS test is that it is computationally expensive and somewhat unpopular to carry out in the multidimensional set-up. Because of the lack of appropriate **R**-package, the whole procedure had to be reconsidered and implemented.

Suppose we are given a simulation with $\tilde{\mathrm{N}}$ iterations after the burn-in period. It often happens that the algorithm stays for more than one iteration in certain sample points[9]. We count the occurences of that points and then normalise them dividing them by $\tilde{\mathrm{N}}$ and call the result *a charge*. Thus, we are given a matrix with sample coordinates and their charges. If the target measure π is absolutely continous with respect to the Lebesgue measure, then the points generated by accepted proposals should all differ on both of their coordinates, the measure of any submanifold of $\mathbb{R}^{\mathrm{K}}$ being zero. If that is not the case, then we simply face the problem of computer's finite arithmetics and could solve it by a longer binary representation of the real numbers. In practice that is an extremely unlikely event and we will neglect it. Therefore we will from now on focus on points with distinct coordinates and refer to the number of thereof by N. The world *sample* will also be applied only to the uppermentioned points.

The computations of the **KS** distance might be computationally expensive, requiring, in pesimistic case, evaluation of the true Cumulative Distribution Function F in approximately $\frac{\mathrm{N}^{\mathrm{K}}}{\mathrm{K}!}$ points[10]. To see that, let us focus in all of this section on the bivariate case[11].

---

[9]In fact we can easily control the point sejour-time due to the random walk by manipulating the proposal step's distribution parameters. In case of the kernel generated by the conditional normal distribution, it suffices to diminish the entries of the covariance matrix.

[10]The exact number being equal to number of points on the intersection of a K + 1 dimensional simplex with the coordinates summing to N, and the $\mathbb{Z}^{\mathrm{K}+1}$ lattice.

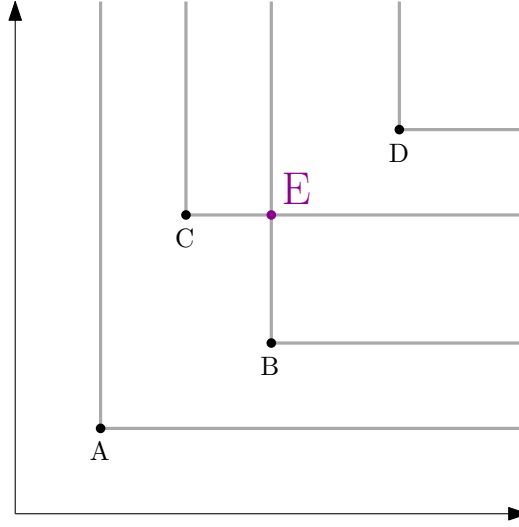[11]...the generalisation to higher dimension being straighforward.

Figure 3.2: Sample points and level sets of an examplary **ECDF**.

Observe, that the level sets of any **ECDF** are uniquely determined by the sample points (as A, B, C and D in Figure 3.2) and points generated from them by taking the vectorised maximum

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \vee \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1 \vee b_1 \\ a_2 \vee b_2 \end{bmatrix},$$

such as E in Figure 3.2.

One of the easiest way of establishing the values on the level sets of $\hat{F}_N$ is by considering a square, $(N + 2) \times (N + 2)$ matrix B, as in Figure 3.3. The entries of B correspond to values of $\hat{F}_N$ on certain rectangles. Take the set of all x-coordinates of sample-points, Φ_x, and the set of all y-coordinates of sample-points, Φ_y. Add to them the coordinates of two dummy points: (x_0, y_0) and (x_{N+1}, y_{N+1}) We enumerate points of these sets in the ascending order,

$$\Phi_x = \{x_0 < x_1 < \cdots < x_N < x_{N+1}\}$$

and

$$\Phi_y = \{y_0 < y_1 < \cdots < y_N < y_{N+1}\}.$$

Then, the $\hat{F}_N$ function is constant on rectangles

$$R_{ij} = [x_i, x_{i+1}) \times [y_j, y_{j+1}),$$

where $i, j \in \{0, 1, \ldots, N + 1\}$. It is easy to realise now, that the number of level sets depends on the number of sample points and points generated by them using the vectorised maximum, $\hat{N} = \#\{z : \exists x, y \in \text{Sample} x, y \neq z \text{ and } z = x \vee y\}$. This number is easily seen to be the greatest if all sample points could be arranged so that their x-coordinates strictly increase and y-coordinates strictly decrease.

To actually derive the **KS** distance we must assume that we can evaluate not only the true **CDF** F, but also its two marginals[12]. Similarly to its univariate counterpart, the **KS** distance can be calculated on only a finite number of points. Consider any rectangle. $\hat{F}_N$ is constant on it. So the **KS** distance depends solely on the evaluations of F on that set. But F is a distribuant - a function monotone in all its arguments, and so the maximum distance from $\hat{F}_N$ can be attained only on the

---

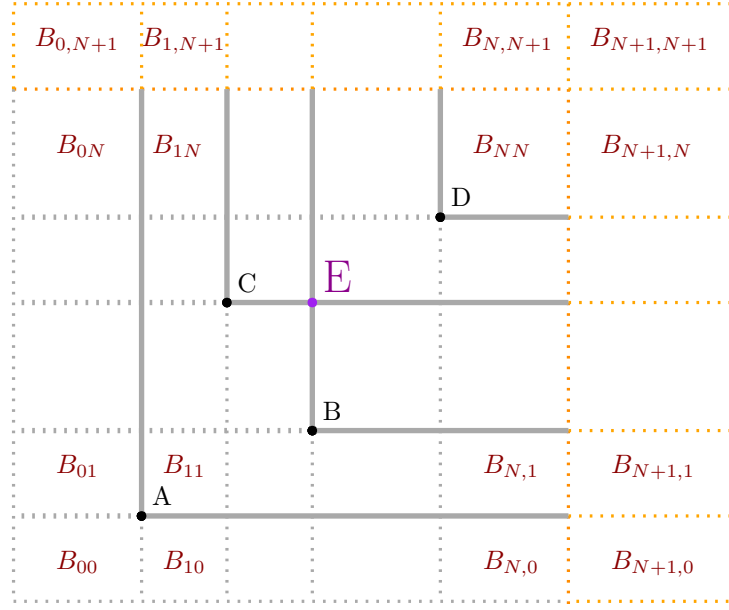[12]Limits of all the possible subsets of coordinates in infinity.

Figure 3.3: Relation between B matrix and **ECDF** level sets.

vertices of the rectangle. On the border rectangles ($R_{i,N+1}$ and $R_{N+1,j}$) the evaluation simplifies to vertices adjacent to R_{ij} rectangles and it is there, where we have to evaluate the marginals instead of F. $\hat{F}_N$ is equal to zero on rectangles on the southern and western extremities. Thus we put $B_{i0} = B_{0j} = 0$.

The rest of the algorithm is based on the dynamic programming approach introduced by Ni and Vingron (2012) for comparing lists of changes in gene expression under different scoring regimes. The overall algorithm proceeds as follows:

Algorithm 3.3.1 (2D-ECDF).

1: $KS := 0$
2: $\text{J} = \text{N} + 1$
3: *Read* α
4: *Prepare* Φ_x *and* Φ_y.
5: **while** $i \in \{1, \dots, \text{N} + 1\}$ **do**
6:     **while** $j \in \{1, \dots, \text{J}\}$ **do**
7:         $B_{ij} \leftarrow \begin{cases} B_{i-1,j-1} + charge, \ R_{ij}\text{'s upper-left vertex contains a sample-point} \\ B_{i,j-1} + B_{i-1,j} - B_{i-1,j-1}, \ otherwise. \end{cases}$
8:         **if** $i, j < \text{N} + 1$ *and* $B_{ij} > B_{i,j-1} \vee B_{i-1,j}$ **then**
9:             $F_{ij} \leftarrow F(x_i, y_j)$
10:         **else if** $j = \text{N} + 1$ **then**
11:             $F_{ij} \leftarrow F(x_i, \infty)$
12:         **else if** $i = \text{N} + 1$ **then**
13:             $F_{ij} \leftarrow F(\infty, y_j)$
14:         **end if**
15:         **for** $a \in \{i - 1, i\}$ *and* $b \in \{j - 1, j\}$ **do**
16:             $A \leftarrow |F_{ij} - B_{ab}|$
17:             **if** $A > KS$ **then**

```
18:              KS ← A
19:          end if
20:          if F_{ij} ∧ B_{ij} > 1 − KS − α then
21:              J ← j − 1
22:          end if
23:      end for
24:      j ← j + 1
25:  end while
26:  i ← i + 1
27: end while
```

Important changes with respect to Ni and Vingron (2012) can be seen in line 16 and the introduction of two extra variables J and α in lines 6, 21, and 20. The introduction of J stems from the observation, that if both F and $\hat{F}_N$ are already in the interval $(1 − KS, 1]$, then any further evaluations of KS to the East and North from the grid point under study could not result in a bigger difference then the one already observed. Parameter α enlarges that band further to $(1 − KS − \alpha, 1]$, so that the true **KS** distance will not be larger than by α from the value already established. We have also noted that it is not important to evaluate the real distribuant at every point of the matrix B. Rather than that, we note that because F is a distribuant, evaluations are required on South-Eastern edges of the level sets, such as points A, B, C, D, and E on Figure 3.3. That is the reason behind code in line 9. Other changes seem to be of minor importance and result from discretisation of the *a priori* continous problem.

Our impelementation of the **KS** distance calculator might not the optimal one. As pointed out by Bentley (1980) the problem of calculating values of an **ECDF** only in the sample points could be solved in $\mathcal{O}(N \log(N))$ iterations. The existence of an algorithm requiring only $\mathcal{O}(\hat{N} \log(\hat{N}))$ operations, interesting in its own sake and probably rather straightforward, will be object of further research.

Chapter 4

Simulations and Results

Recall that Liang-Wang example of a multimodial function:

$$f(x) = \sum_{i=1}^{20} \frac{\omega_i}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(x - \mu_i)^{\mathrm{t}}(x - \mu_i)}{2\sigma_i^2} \right), \tag{4.1}$$

where $\sigma_1 = \cdots = \sigma_{20} = 0.1$, $\omega_1 = \cdots = \omega_{20} = 0.05$ and the means μ_i are enlisted in Chapter 1.

Since we are dealing with stochastic simulations, do focus on the first two empirical moments analysis of different statistics in order to get some understanding of which SWAP STRATEGIES is better. Because of theoretical reasons mentioned in the Chapter 2, one might be tempted to compare different SWAP STRATEGIES by considering the behaviour of their Колмогоров-Смирнов distance with the true underlying distribution, given by Eq. 4.1. To this end the implemented KS statistics calculator was used, see Chapter 3.3. The overall results are gathered in Figure 4.1.

The simulations involving the calculation of the KS statistics were carried out using the same parameters. The chains proposals have had their covariances matrix chosen so that about one in four proposal got accepted, which is the working rule-of-thumb among the practitioners of the stochastic simulation after the seminal work of Roberts and Rosenthal (2001). The covariances were chosen to be a slightly modified versions of those proposed by Baragatti *et al.* (2012). Baragatti chosen the matrices to be of the form $T_i^2 \mathbb{I}$, where $\mathbb{I}$ is simply a 2 by 2 identity matrix and T_i the ith temperature. The modification consisted in premultiplying the frist three matrices by 0.05 and the last ones by 0.01. The temperatures were also taken from baragatti and were equal to $T = (\,1,\ 2.8,\ 7.7,\ 21.6,\ 60\,)$. The computation involved 10000 iterations, a fourth of which was neglected being the burn-in period. The initial states were chosen from a uniform distribution on a square $[0, 10]^2$ that contains all the means of the distributions that add up to form the considered mixture.

Figure 4.1 summarizes results of two different experiments consisting of different number of simulations — the first of 40, the latter - of 120. There was no particular reason for choosing this amount of simulations — the time-cost of the simulations that do calculate the KS statistics is however quite long and so we did not manage to collect information on the same number of simulations. The second group has larger standard deviation — this can be attributed to differences in number of experiments. However what can be compared is the mean of these distributions. Also the standard deviation can be compared within the two groups. One can see that the differences in mean values of the KS statistics are small, possibly neglectable. However it remains a fact, that first three strategies give results that are on average smaller, so that they are closer to the original distibution in the KS sense. This is to say that generally the state-dependent strategies give better estimates than the state independent one, with the notable exception of the STRATEGY IV. One can also notice that the among the first three state-dependent strategies it is STRATEGY III that has the lowest variance. Among the state independent stragies it also seems plausible not to draw swaps from a

space restricted to only neighbouring ones, leaving it possible to exchange accepted proposals in the RANDOM WALK phase between all the chains.
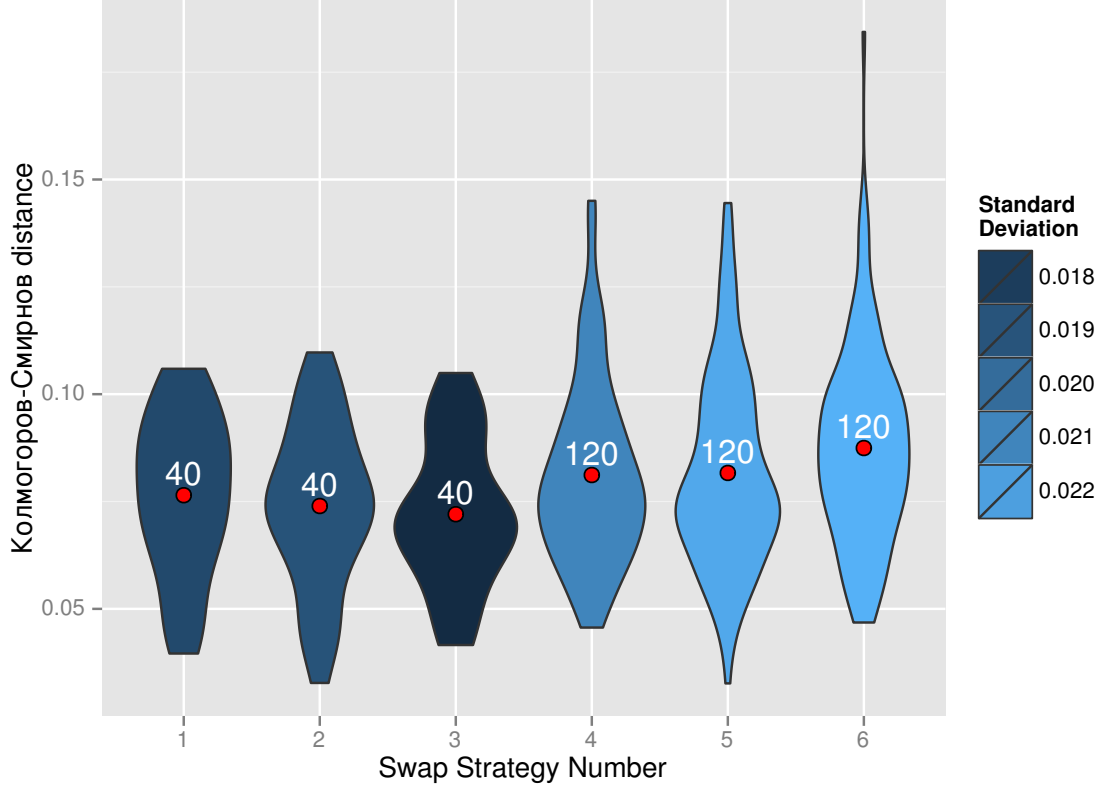

Figure 4.1: Results of the KS-statistics simulations. The number of iterations a particular strategy was tested is annotated with the white numbers just above the red dots. The violin plots are simply smoothened empirical distributions. Red dots are their empirical means. The colour of the violin plots corresponds to the standard deviation, which is more instructive to study when considering subgroups with the same number of simulations.

4.1. PTEEM as motivation for STRATEGY I

One of the aims of the simualation was to compare ourselves with the results of Baragatti *et al.* (2012). In this article the author proposed an algorithm that supposingly combines the good sides of both the PARALLEL TEMPERING and the Equi-Energy Sampler, orignally conceived by Kou *et al.* (2006). The resulting algorithm, called PTEEM, again consists of two steps. In the original article it is assumed that the probability one faces is uniquely defined by a hamiltonian that describes the energy levels of a system, so that the density with respect to some measure λ is

$$\pi(x) \propto \exp\Big(-h(x)\Big),$$

however, as we shall see, it is not stringent a condition. The first phase is a simple random walk done independently for all the coordinate chains. The other step is more complicated. In PTEEM the STATE SPACE is divided into regions called energy rings, being regions with the same energy levels, or simply $D_j = h^{-1}[H_j, H_{j+1})$, where H_j are chosen *plus ou moins* heuristically. In the second

phase of the algorithm two draws are perforem: (1) an energy ring is chosen at random among all those that contain at least two chains[1] and (2) two chains that are in the same energy ring are chosen at random and such proposal gets either accepted or rejected.

The very idea of this approach is therefore much similar to what is being done in Strategy I and comparisons with the results obtained there.

In order to do so additional simulations were carried out without the evaluation of the KS statistics. The temperatures and proposal covariances were the same as those described above. We have reduced the overall number of iterations to 7500 with the burn-in period set to 2500 iterations as before. The initial states were drawn uniformly from a square $[0, 1]^2$ for the reason of making it more difficult for the algorithm to reach all the modes.

In Figure 4.2 one can notice that the choice of the starting point might have truly influenced the overall properties of the Parallel Tempering. It plots the number of undiscovered modes for different strategies. What can be noticed is that all strategies sometimes fail to discover the modes that are far away from the starting points. The number of undiscovered modes is a measure of the algorithms mixing properties, as it says much about the algorithms inability to find itself in a certain place in the State Space.

To assign different sample points $\{X^{[k]}\}_{k=0}^{K}$ generated by the algorithm to particular modes a classifier had to be constructed. In our case, a randomised χ^2-classifier was used. The reason behind it is that if a random vector is normally distributed and centered at point x, then the distance from mode x is χ^2-distributed. So, one can assign a given point to its mode at random in the following way: for one sample point (1) calculate the probabilities of observing radius bigger than the one observed for all the modes and (2) chose at random the mode with the probability proportionate to the quantities calculated in point (1). Such a procedure assures that points that are closer to a particular mode are much more likely to be assigned to it. Also, if there are sevaral points in the proximity of two modes, then the randomness allows to redistribute the points in a way that does take into account that some of the probability mass comes from one mode, and some from the other.

Studying Figure 4.2 one notices easily, that all the state-dependent strategies have a clear advantage over the state-independent strategies in their ability to explore most of the State Space. Above all, Strategy VI, where only neighbouring chains are permitted to exchange accepted proposals from the Random Walk phase, fails miserably and in 5 promiles of cases does not even discover the closest modes to the place of initial drawing. It is Strategy III that managed to discover most of modes most frequently and always discovers all the nearby modes to tha place of initial drawing. We think that this is also the reason why Strategy III scores better in the KS tests, however it remains obscure why Strategy IV acts similarly to state-independent strategies when it is quite good at exploring most of the State Space.

Another way of comparing different strategies is to see how they manage to approximate the weights of different modes in the entire distribution. A possible criterion for measuring the quality of approximation is to look upon the average absolute error. It averages the absolute distance of a simulation approximation of weight from the true one, equal to 0.05. Figure 4.3 summarises the results of such procedure.

What can be seen in Figure 4.3 is that again the state-independent proceduters are notably worse in prescribing the correct weights to different modes. The average absolute errors reach 0.025 and even 0.03 which amounts to respectively 50% and 60% of the true value, which is much. It's worth stressing however that even the winning Strategy I, that manages to score best in 18 out of 20 modes, sometimes reaches a 50% relative[2] error. It's best result is to score a mearly 30% relative error.

---

[1] One sets the number of energy rings and temperatures so that it is always the case.
[2] Again with respect to 0.05

One can also wonder whether the quality of estimates improves whith longer runs of the algorithm, that is, whether the procedure is stable and convergent. Figure 4.4 gives an answer to such question. All the rhombi that correspond to longer runs of the algorithm are found inside the dots that represent the shorter ones — significantly longer runs (in the above example three times longer) do lead to overall improvement in the precision of estimates. This improvement amount to about a 50% reduction in error, which stems from visual inspection of the length of rhombi's diagonal wiht respect to the dots' diameter. One notices also slight reshufflements in ranking among the state-dependent strategies, as in different plots pairs dot-rhombus with different colour schemes appear: STRATEGY III seems to overtake STRATEGY II in ranking. One can also obeserve that among the state-independent strategies there no difference in rankings, STRATEGY VI restricted to swaps between neighbouring chains loosing with STRATEGY V. One can also notice that STRATEGY IV has better results than STRATEGY V in one additional mode.

One can also try to compare different strategies in their task of actually appoximating different integrals. We have restricted ourselves to evaluate how the algorithms handle the task of actually calculating the real moment of Liang's distribution. These are readily computable analytically.

In Figure 4.5 we compare the outcomes of different Swapping Strategies in this particular task. Observe that all the strategies on average perform admiringly in this task, since the average empirical estimate of moments is very near the true value — the rhombi are usually near the centre of the dot that marks the true value. One can see that all the algorithms better estimate the first coordinate of the mean than the second. Also the estimates of the covariance matrix show, that the variance of the y coordinate have bigger variance. Within the state-independent group of strategies it is also apparent that the possibility to accept swaps from not-neighbouring chains results in a concentration of the empirical distribution. However results of state-dependent strategies seem to be fairly comparable.
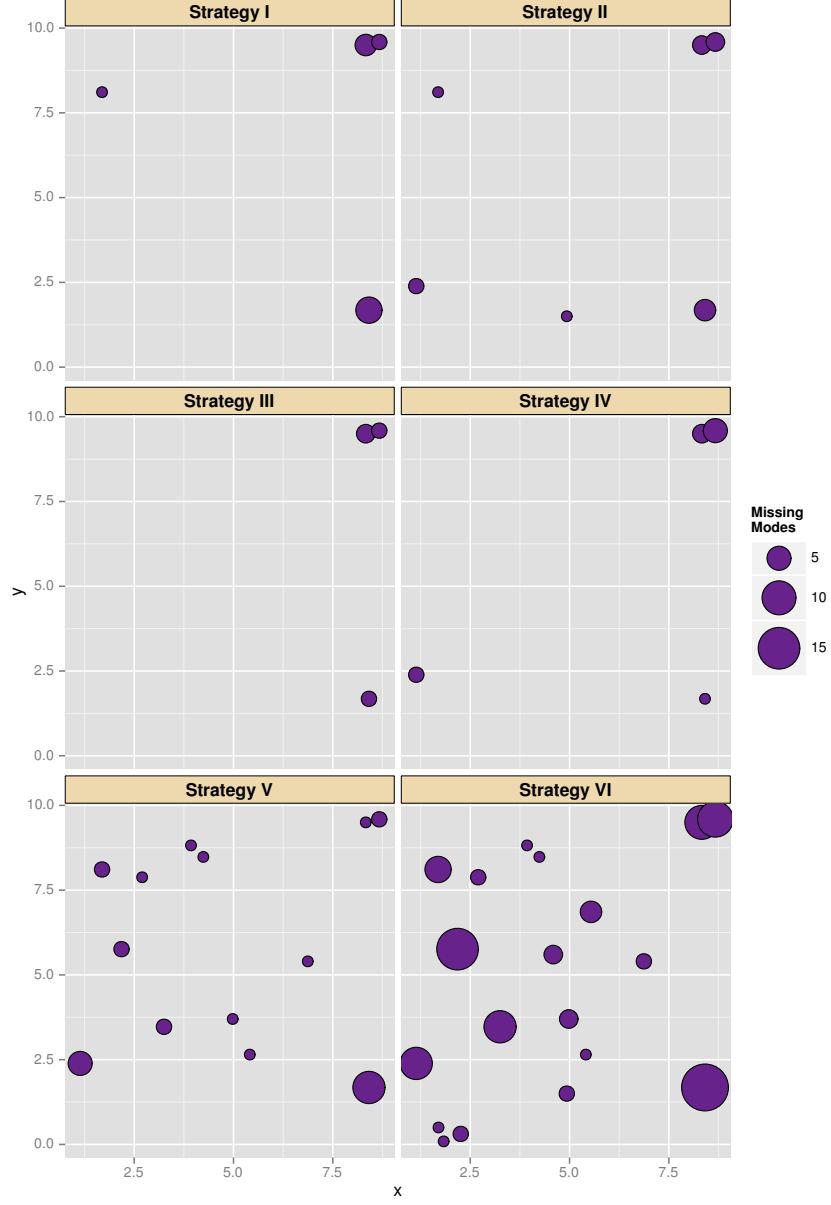
Figure 4.2: The baloon plot depicts the failure of the PARALLEL TEMPERING under different swapping regimes to discover during one simulation all the components of the Liang's disribution, defined by Eq. 4.1. The sample points from the simulated chains were assigned to modes using the randomised χ^2-classifier. The size of the baloon corresponds to the number of simulations, out of 1000, that resulted in not assigning any points to the particular mode. The less dots appear on the plot, the more more modes were discovered,
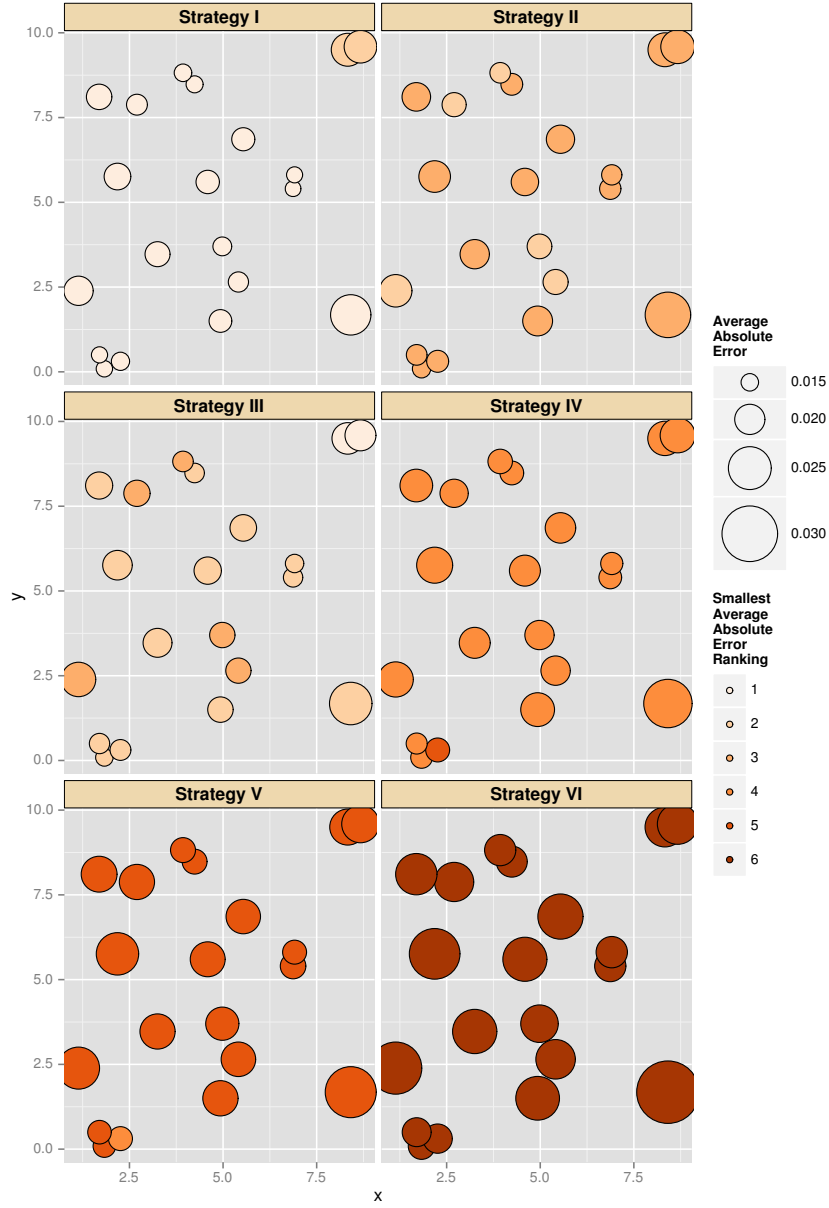
Figure 4.3: The baloon plots depicts the errors of the PARALLEL TEMPERING in evaluating the weights of different components of the Liang's mixture, defined by Eq. 4.1, under different Swap Strategies. The sample points from the simulated chains were assigned to modes using the randomised χ^2-classifier. The error is calculated simply as the l^1 distance of results divided by the number of observations, being equal to 1000. It is being calculated seperately for different strategies and different distributions that compose the mixture; all of them appear with weight being equal to 0.05. The dots are coloured according to their position in the overall ranking of errors, done separately for every mode of Liang's distribution, so that the darker they are, the bigger was the error for a particular mode. The sample points from the simulated chains were assigned to modes using the randomised χ^2-classifier.
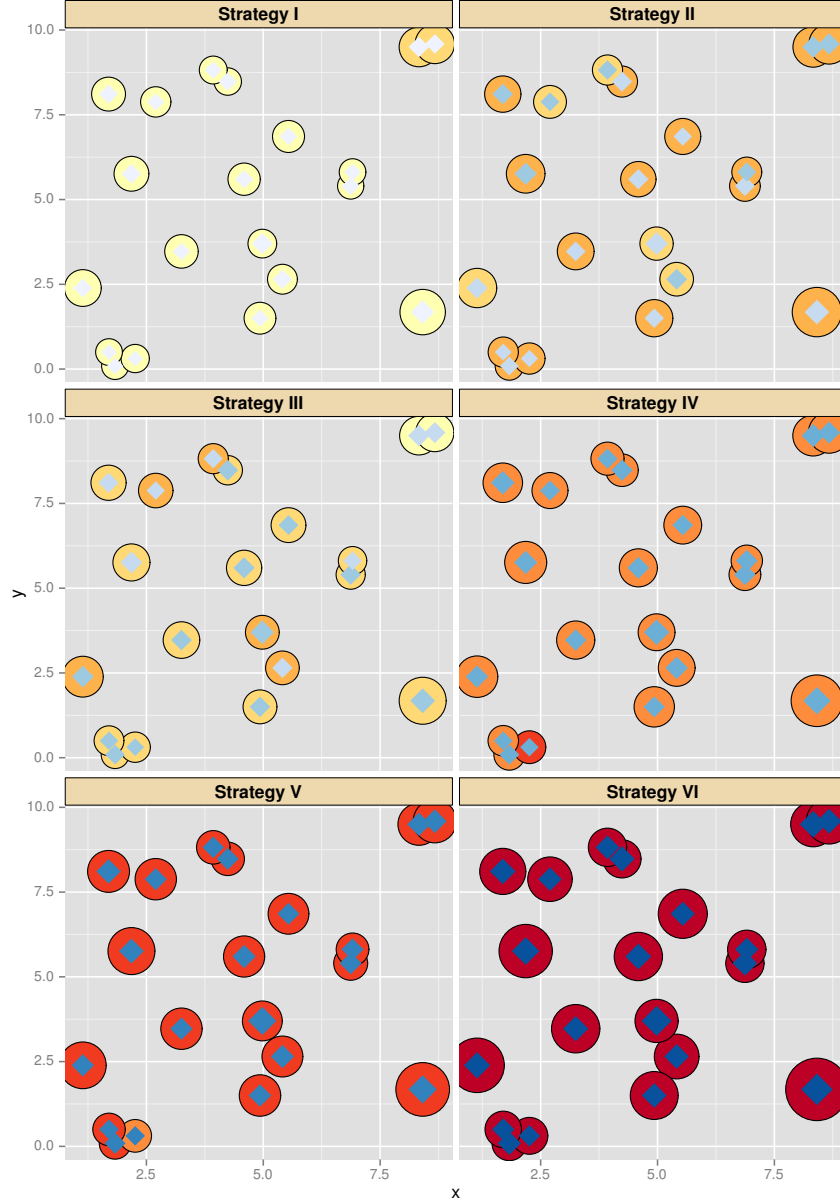
Figure 4.4: This figure explores the differences in the quality of the estimations of modes between shorter and longer runs of the PARALLEL TEMPERING for different swapping strategies. Both dots and rhombi represent the Average Absolute Errors: dots correspond to shorter simulations — 2500 iterations, and rhombi to longer simulation — 7500 iterations, *ceteris paribus*. If the vertices of a given rhombus were to touch the border of a dot then errors would be the same; this, however, never occurs and the estimates are indeed much sharper. Again the colours intensity helps distinguish among the ranking of approximation: lighter colours correspond to smaller values of errors.
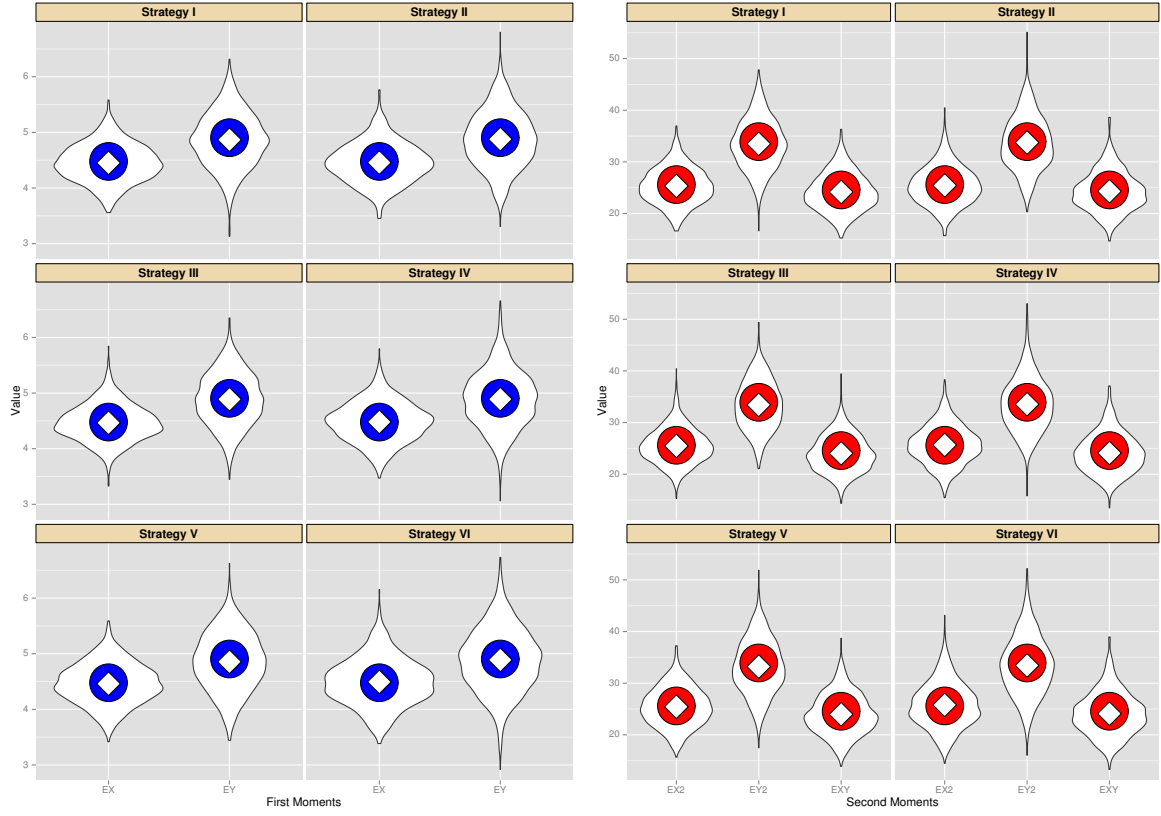
(a) First Moments

(b) Second Moments

Figure 4.5: The violin plots depict smoothened distributions of first and second moments of the Liang mixture, described by Eq. 4.1, for different Swap Strategies. The empirical means are ploted as white rhombi on the blue background. The blue dots are centered at Liang Distribution's exactly calculated moments, so that one can asses the differences of theory and experiment simply by comparing the relative posititioning of the centres of rhombi and dots.

Summary and Future Research Directions

As seen in the Chapter 1 the standard Metropolis-Hastings algorithm has intrinsic limitations when it comes to solving the PROBLEM of drawing sample points from multimodal distributions. The PARALLEL TEMPERING serves as a potential solution to that PROBLEM, being at the same time quite a flexible approach.

In this work we have envisaged several Swapping Strategies. These Strategies are nothing else but laws according to which the PARALLEL TEMPERING travels through the STATE SPACE. STRATEGY I served together with STRATEGY V and STRATEGY VI as reference points for the remaining three strategies. STRATEGY I is based on the idea of PTEEM devised by Baragatti *et al.* (2013) of swaps occuring between chains that end up having the most similar temperature levels. STRATEGY V and STRATEGY VI on the other hand are state-independent strategies, as exposed in Miasojedow *et al.* (2013b). Two different measures have showed a slight improvement in approximating the target distribution as measured by the two-dimensional KS statistics and the count of undiscovered modes when using STRATEGY III. On the other hand the criterion based on measuring approximated Average Absolute Error[3] has given better results when using STRATEGY I. However all the measures yield fairly similar results in that the state-dependent strategies are better than state-independent strategies and so their use seems more than plausible when solving real life problems.

All the necessary calculations were done using a state-of-the-are programme written in **R** statistical computation language. The implementation envisages a meticulously applied object oriented programming paradigm dictated by the modularity of the problem, as exposed in Chapter 3. The modularity of the solution to the PROBLEM stems from instrinsic features of the Markov Chain Monte Carlo that can be better understood by the study of its more abstract mathematical formulation that underlines the possibility of using virtually any STATE SPACE. From the statement of this fact the observation, that both the METROPOLIS-HASTINGS and the PARALLEL TEMPERING can be understood as algorithms that need as input only evaluations of different probabilities. The probability measure[4] is therefore the only link between the implementation of the STATE SPACE and the implementation of the ALGORITHM.

In search for some good measure of comparing different strategies, the KS statistics was given appropriate attention. Because of the lack of a R-implemented software, a new implementation was derived, based on approach derived by Ni and Vingron (2012), with minor improvements that prevent from unnecessary calculations.

All of the implemented software is to be shiped soon as an independent **R** package.

As for the future plans an implementation of an adaptive version of the PARALLEL TEMPERING is planned. The need for such algorithm stems from a potential drawback in the PARALLEL TEMPERING, which is the lack of a unified approach to setting up the temperature levels. In fact the internalisation of such procedures into the algorithmic model could enhance mightly the efficiency parameters of the algorithm, e.g. by reduction in the number of unnecessary chains.

---

[3]Approximated by the arbitrary but reasonable choice of the sample points classifier.
[4]With minor exception for the Quasi-Metric applied in STRATEGY IV.

We also think that implementation of the KS calculator can be significanlty improved by meticulous application of the multi-dimensional *divide et impera* approach, originally developed by Bentley (1980).

Moreover, for technical reasons the derived template should be re-implemented using a faster and more orderly computer language, such as **C++**.

Appendices

Appendix A

Description of implemented methods within objects

ALGORITHM

R CODE	DESCRIPTION
| ```r
initialize =
  function (
    iterationsNo = NULL ,
    burnIn = 2000L ,
    ...
  )
``` | This method initialises the ALGORITHM object. The default setting of the iterations number to NULL is technical and bypasses an error in the Reference Classes implementation: the Reference Classes objects have their constructors that are again Reference Classes objects. Calling the constructor automatically calls generation of the underlying object. This cannot be constructed properly without user-provided inputs. An easy solution is to include in that objects an if statement that checks whether what happens is not the above-mentioned case and construct an empty structure if that is so.<br>A typical burn-in is set to 2000 steps. |
| ```r
show = function (...)
anteSimulationShow =
    function (...)
postSimulationShow =
    function (...)
``` | These bulk methods divide are responsible for the exploration of the inputs and outputs of the algorithm in the terminal. |

R CODE	DESCRIPTION
`getDataForVisualisation = function(...)`	Orders the state space to manipulate the data so that plotting procedures of the **ggplot2** package could handle them
`makeStepOfTheAlgorithm = function(` `iteration,` `...` `)`	Calls procedures responsible for execution of a single algorithm step: in case of the Metropolis-Hastings algorithm it executes the Random Walk. In case of the Parallel Tempering algorithmit undertakes both the Random Walk and the Random Swap.
`turnOnBurnIn = function(...)` `turnOffBurnIn = function(...)`	These methods pass the information to the State Space on whether it should avoid remembering sample points because of the burn-in period, or not.
`simulate = function(...)`	Starts the simulation process.

Metropolis-Hastings

R CODE	DESCRIPTION
`insertChainNames = function(...)`	Stores the names of different chains.

R CODE	DESCRIPTION
`prepareSimulation = function ( . . . )`	Initialises values needed before the simulation.
`acceptanceRejection = function ( . . . )`	Tabularises different chains statistics on rejection and acceptance in the Random Walk phase.
`randomWalk = function ( . . . )`	Performs the random walk step: it asks the state-space to generate the logs of unnormalised probabilities evaluated in the proposed points and then performs the usual rejection part. All this could be done parally if it was needed - this feature will be shipped with version 2.0.
`randomWalkRejection = function ( . . . )`	Here the Hastings quotients get compared with randomly generated values from the unit interval. All values are taken in logs for numerical stability.
`getLogAlpha = function ( . . . )`	Substracts logarithms of evaluations of the unnormalised densities in the proposal points from their last step counterparts.
`triggerUpdate− −AfterRandomWalk = function ( . . . )`	This procedure updates the STATE SPACE after an operation consisting of accepting any new proposal in the random-walk phase of the algorithm. Updates are also needed in the probabilities of the last states stored in a field in the parallel-tempering object.
`updateAfterRandomWalk = function ( . . . )`	This procedure updates the information gathered by the ALGORITHM: it updates the correct logarithms of the unnormalised densities and notes which chains did move in the Random Walk.

ParallelTemperings

R code	Description

``
insertStrategyNo =
  function(
    strategyNo
)
``
	Checks whether user inserted correct swapping strategy number.

``` getNeighbours = function(...) ```	Generates the indices of the neighbouring chains in the lexicographic ordering.
``` insertTranspositions = function(...) ```	Prepares the dictionary between lexicographic ordering and pair ordering of possible swaps; evaluates the maximal number of potential swaps and inititiates a matrix that stores information on the transitions that occured throughout the simulation.
``` plotHistory = function(...) ```	Prepares a plot of the swaps distribution that occured during the simulation.

``
writeSwaps =
  function(
    directoryToWrite,
    ...
)
``
	Prepares a **.csv** file containing the swap history.

``` swapHistory = function(...) ```	Tabularizes the information on swap history. First row enumerates how many swaps of a given type were proposed. The second one enumerates how many of these were actually accepted.

R CODE	DESCRIPTION
| ```
swap =
  function (
    iteration
  )
``` | Performs the swap phase of a given step of the algorithm. |
| ```
swapProposalGeneration =
  function (...)
``` | Generates the proposal for the SWAP. The function differentiates between state-dependent and state-independent swaps. It stores the information about the swap for later analysis. It also marks which chains did exchange their last sample points. |
| ```
swapRejectionAndUpdate =
  function (
    iteration ,
    ...
  )
``` | Performs the rejection in the swap step and the resulting update. Basing on the information of which interchanges did occur, it reavaluates the values of the unnormalised probabilities of swaps. It calculate Hastings quotients for the SWAP phase and performs rejection. After the rejection it updates the swap history and the values of the above-mentioned probabilities. Finally, it orders the STATE SPACE to reshuffle the sample points and store them. |
| ```
updateSwapUProbs =
  function (
    transpositions−
    −ForUpdate ,
    ...
  )
``` | Updates values of unnormalised probabilities of swap. |
| ```
findTranspositions−
−ForUpdate =
  function (...)
``` | Finds numbers of transpositions in the lexical ordering whose probabilities must be updated after the random walk phase. |

R CODE	DESCRIPTION
| ```
updateSwapUProbs =
  function (
    transpositions−
    −ForUpdate
  )
``` | Updates values of unnormalised probabilities of swap. |
| ```
generateTranspositions =
  function (
    chainNumbers
  )
``` | Creates a matrix that enlists all possible transpositions of supplied indices of the temperature vector. |
| ```
translateLexicTo−
−Transpositions =
  function (
    lexics
  )
``` | Translates transpositions ennumbered lexicographically into pairs of corresponding numbers. |
| ```
translateTranspositions−
−ToLexic =
  function (
    transpositions
  )
``` | Lexicographically orderes given pairs of transpositions. |
| ```
swapStrategy =
  function (
    transposition
  )
``` | Calculates the unnormalised probabilities of swaps, based on the preinserted strategy number. |

StateSpaces

R code	Description
| ```
insertInitialStates =
  function(
    initialStates = matrix(
      ncol=0, nrow=0),
    spaceDim = 0L,
    chainsNo = 0L,
    ...
)
``` | Checks whether the user inserted correct initial states. If their dimension is different then the dimension supplied by the user, or they were not enough of them, i.e. less than the number of chains, or if initial states were not supplied, it generates new ones uniformly from a hypersquare $[0, 10]^{\text{Problem Dimenstion}}$. |
| ```
turnOnBurnIn =
  function (...)

turnOffBurnIn
  = function (...)
``` | These methods pass the information to the STATE SPACE on whether it should avoid remembering sample points because of the burn-in period, or not. |
| ```
proposeLogsOfUMeasures
  = function (...)
``` | Calls the TARGET MEASURE to evaluate the logarithms of unnormalalised density or probability function. |
| ```
randomWalkProposal
  = function (...)
``` | Generates the proposal in the Random Walk phase and returns the results to the ALGORITHM. |
| ```
updateStatesAfter-
-RandomWalk
  = function (...)

updateStatesAfterSwap
  = function (...)
``` | Realises the call from the ALGORITHM to update the required parameters. |

R CODE	DESCRIPTION
`calculateBetweenSteps` `= function ( ... )`	General wrapper for updates to be executed between the Random Walk phase and the Random Swap phase.

REAL (STATE SPACE)

R CODE	DESCRIPTION

`insertInitialStates =`
`  function (`
`    initialStates = matrix (`
`        ncol=0, nrow=0),`
`    spaceDim = 0L,`
`    chainsNo = 0L,`
`    ...`
`)`
	Checks whether the user inserted correct initial states. If their dimension is different then the dimension supplied by the user, or they were not enough of them, i.e. less than the number of chains, or if initial states were not supplied, it generates new ones uniformly from a hypersquare $[0, 10]^{\text{Problem Dimenstion}}$.

`createDataStorage =` `  function ( ... )`	Creates an appropriately big matrix for the storage of sample points.

`insertProposal−`
`−Covariances = function (`
`  proposalCovariances`
`    = matrix (`
`        ncol=0,`
`        nrow=0`
`    ),`
`    ...`
`)`
	Checks whether the user provided correct proposal covariances. The user can provide one matrix if he wants the covariances to be the same for all chains. If that is not the case, the user is obliged to provide a list of matrices that contains as many matrices as there are temperature levels.
	If the user provides wrong data, unit variances are to be chosen. If the user provides an object whose type is neither matrix, nor list, then the algorithm stops.

R code	Description

```
checkCovariance = function
    (
    covarianceMatrix ,
    ...
    )
```
Returns true of the given object is a matrix and its dimensions match the dimension of the STATE SPACE provided by the user beforehand.

Real Tempered (State Space)

R code	Description

```
insertTemperatures =
  function(
    temperatures ,
    ...
    )
```
Checks whether the user inserted correct temperature values and stores them.

```
getIteration =
  function(
    iteration    = 1L ,
    type    =
      'initial states ',
    ...
    )
```
For a given iteration extracts results of a given step type, to choose among 'initial states', 'random walk', and 'swap'.

```
prepareDataForPlot =
  function ( ... )
```
Reshuffles the entire history of states so that the entire result conforms to the data frame templates of **ggplot2**

```
plotAllTemperatures =
  function ( ... )
```
Performs a plot of all simulated chains with an overlayed map of the real density from the Liang example.

R CODE	DESCRIPTION
| ```
plotBasics =
  function (
    algorithmName ,
    . . .
  )
``` | Performs a plot of the base level temperature chain of main interest with an overlayed map of the real density from the Liang example. |
| | |
| ```
measureQuasiDistance =
  function (
    iState ,
    jState ,
    . . .
  )
``` | Measures the quasi distance between states needed for. |

TARGET MEASURE

R CODE	DESCRIPTION
| ```
measure =
  function ( . . . )
``` | Evaluates the density or the probability function at a given point. |
| | |
| ```
establishTrueValues =
  function ( . . . )
``` | Evaluates the values of the provided density on a grid $[-2, 12]^2$ with the mesh set at $0.1$. These are to be plotted so that the user may compare the results of the simulation with how it really looks. |

Target Unnormalised Densities

R code	Description
| ```r
initialize =
  function(
    targetDensity =
      function () {},
  ...
)
``` | Stores the unnormalised density function provided by the user. |

Target Liang Densities and Matteo Densities

R code	Description
| ```r
initialize =
  function(
    iterationsNo = NULL,
    quantile-
      -SimulationsNo =,
    mixturesNo =,
    mixturesWeight = ,
    mixturesMeans = ,
    sigma = ,
    weightConstant = ,
    algorithmName =
      'Liang',
  ...
)
``` | Initialises the reference example of the Liang density described in detail in chapter .... |
| ```r
plotDistribuant =
  function (...)
``` | Plots the distibuant of Liang density based on the grid points from $[-2, 12]^2$ with the mesh set at $0.1$. |
| ```r
distribuant =
  function(
    x,
    ...
  )
``` | Calculates the true value of Liang's measure distribuant at point $x$. It serves as a reference value when evaluating the Колмогоров-Смирнов distance. |

R code	Description

``
marginalDistribuant =
  function (
    proposedState ,
    coordinateNo ,
    . . .
)
``
	Calculates the true value of Liang's measure marginal distribuants at point x. One can insert the coodinate number, i.e. 1 or 2, which does not assume the infinite value. It serves as a reference value when evaluating the Колмогоров-Смирнов distance.

``
getSquareGrid =
  function (
    minimum ,
    maximum ,
    mesh ,
    . . .
)
``
	Prepares the matrix with values from $[-2, 12]^2$ with the mesh set at 0.1.

``
getQuantiles =
  function (
    simulationsNo ,
    . . .
)
``
	Simulates the values of Liang Measure's quantiles using Monte Carlo simulation scheme.

``
simulateQuantiles =
  function (
    simulationsNo ,
    . . .
)
``
	Writes down the results of the quantile simulation for further calculations. This function is handy when trying to evaluate visually the correctness of different simulation strategies.

``
measureDistance−
−FromMeans =
  function (
    point ,
    . . .
)
``
	Calculates the distance of a given point from all of the means in the mixture of the 20 gaussian densities.

R CODE	DESCRIPTION
| ```r
classify = function(
  point,
  ...
)
``` | Start the classification procedures. Their aim is to prescribe the sample points to particular densities that are components of the Liang mixture of measures. The aim of this procedure is to check whether the presence of the sample points in the approximity of modes approximates well their probability input to the whole distribution, which is set to $1/20^{\text{th}}$. |
| ```r
classifyByLength =
  function(
    point,
    ...
  )
``` | Performs the classification of sample points to the modes using the distance-from-mean criterion. |
| ```r
classifyByChiSquare =
  function(
    point,
    ...
  )
``` | Performs the classification of sample points to the modes using the minimal-chi-square criterion. |
| ```r
getFirstAndSecondMoments =
  function(...)
``` | Calculates all first and second true moments of the Liang distribution. |

Simulations

R code	Description

``
initialize =
  function (
    space = ,
    algo = ,
    target = ,
    example = ,
    iterationsNo = ,
    burnIn = ,
    chainsNo = ,
    spaceDim = ,
    initialStates = ,
    targetDensity = ,
    temperatures = ,
    strategyNo = ,
    quasiMetric = ,
    covariances = ,
    detailedOutput = ,
    save = ,
    trialNo = ,
    evaluateKS = ,
    integratedFunction =,
    rememberStates = ,
    evaluateSojourn = ,
    . . .
)
``
	Initialises the simulation - prepares the appropriate TARGET MEASURE, links it to appropriate STATE SPACE, chooses the right ALGORITHM.

``
checkTemperatures =
  function (
    temperatures ,
    . . .
)
``
	Checks whether the user provided correct temperatures.

``
setIterationsNo =
  function (
    iterationsNo ,
    . . .
)
``
	Checks whether the user provided correct maximal number of iterations that the algorithm is to execute and stores it.

R CODE	DESCRIPTION
`write =` `function ( ... )`	Creates the file where the results of the simulations will get stored, in the `./data` catalogue, under a naming including the number of iterations, number of simulations, and the number of used strategy, in case the user has chosen the PARALLEL TEMPERING algorithm. Also, it calls the particular saving procedure of different objects.
`furnishResults =` `function ( ... )`	Prepares a vector containing the statistics on a particular run of the simulation. Among them: the strategy number, the rejections of the Random Walk, the rejections of the Random Swap, the Колмогоров-Смирнов statistic, the sojourn estimates, and the integral estimates.
`simulate =` `function ( ... )`	Performs the simulation.

Bibliography

Baragatti, M., Grimaud, A. and Pommeret, D. (2012). Likelihood free parallel tempering. *Arxive*.

—, — and — (2013). Parallel tempering with equi-energy moves. *Statistics and Computing*, **23** (3), 323–339.

Bentley, J. L. (1980). Multidimensional divide-and-conquer. *Communications of the ACM*, **23** (4), 214–229.

Geyer, C. J. (2012). *Markov Chain Monte Carlo Lecture Notes*. Unpublished.

Kou, S., Zhou, Q. and Wong, W. (2006). Equi-energy sampler with application in statistical inference and statistical mechanics. *The Annals of Statistics*, **34** (4), 1581–1619.

Liang, F. and Wong, W. H. (2001). *Evolutionary Monte Carlo for protein folding simulations*, vol. 115. Journal of Chemical Physics.

Miasojedow, B., Moulines, E. and Vihola, M. (2013a). Adaptive parallel tempering algorithm.

—, — and — (2013b). An adaptive parallel tempering algorithm. *Journal of Computational and Graphical Statistics*, **I** (january), CHECK IT SOMEWHERE.

Ni, S. and Vingron, M. (2012). R2ks: A novel measure for comparing gene expression based on ranked gene lists. *Journal of Computational Biology.*, **19** (6), 766–775.

Roberts, G. O. and Rosenthal, J. S. (2001). Optimal scaling for various metropolis-hastings algorithms. *Statistical Science*, **16** (4), 351–367.

Swendsen, R. and Wang, J.-S. (1986). *Replica Monte Carlo Simulation of Spin-Glasses, Parallel Tempering*.