Accelerating Bayesian Inference for Evolutionary Biology Models

X. Meyer, B. Chopard, N. Salamin

# Supporting Materials

## Contents

## 1 Methods

### 1.1 Pseudo-code of the main algorithm

This pseudo-code presents the main steps of an iteration of the algorithm presented in the main article. During an iteration, $n_p$ processors are used to propose $n_p$ new samples according to the prefetching prediction. Only $n_k$ samples will be kept depending on the prediction accuracy, with $n_k \in [1, n_k]$.

Steps may be applied by either `All` the $n_p$ processors or by only the `Master` processor for sequential steps. The presented steps are simplified in order to highlight the main concepts of the presented methods. The exact steps can be found in the commented source code.

**Algorithm 1** Main steps of an iteration of the prefetching and adaptive MCMC algorithm (simplified). Equations are defined in the main article

```
All: Propose a new sample $\Phi$ according to the prefetching prediction. The proposal are defined by main article Eq. (2), (3) or (4) and is chosen in function of the presence of correlations between parameters.
All: Compute $\alpha(\Theta, \Phi)$.
All: Send $\alpha$ to the master processor.
Master: Apply the sequential acceptance check for the sequence of $\alpha$ representing the prefetching prediction.
Master: Send the result of the acceptance check (the $n_k$ kept samples $\Theta$ and the full sequence of $\alpha$ ($n_p$ elements)).
All: Receive the result of acceptance check and apply it locally.
All: Update $\bar{\alpha}$ and $\lambda$ using the sequence of $\alpha$ (using Eqs. (1)).
All: Use the $n_k$ kept samples to update $\bar{\Theta}$ and $\Sigma$ (using Eqs. (1)).
All: Each $k$ iteration, apply a component-wise update.
    - Propose univariate moves for each dimension $i \in d$ (decompose a multivariate proposal from Eq. (1), (2) or (3)).
    - Compute $\alpha_i$, the acceptance probability of each univariate move.
    - Update the local scaling $\lambda_i$ using the same formula as for the global scaling $\lambda$ (Eqs (1)).
    - Normalize the vector of $\lambda_i$ to obtain $W$.
```

## 1.2 Detecting the adaptive proposal convergence

The stochastic approximation scheme used to learn the values of $\lambda$, $\Lambda$ and $\Sigma$ is theoretically guaranteed to converge as the number of iterations tends to infinity. However, continuously refining the approximation of these values has a significant computational cost that may not be worth paying once a certain level of accuracy is reached. Indeed, in order to achieve an effective proposal kernel these variables must adequately approximate the form of the posterior distribution but are not required to exactly matching it. Therefore, in our framework, the convergence of these variables is monitored and their learning phase is stopped once a decent level of stability is reached such as to compromise between their accuracy and the computational effort spent for their approximation.

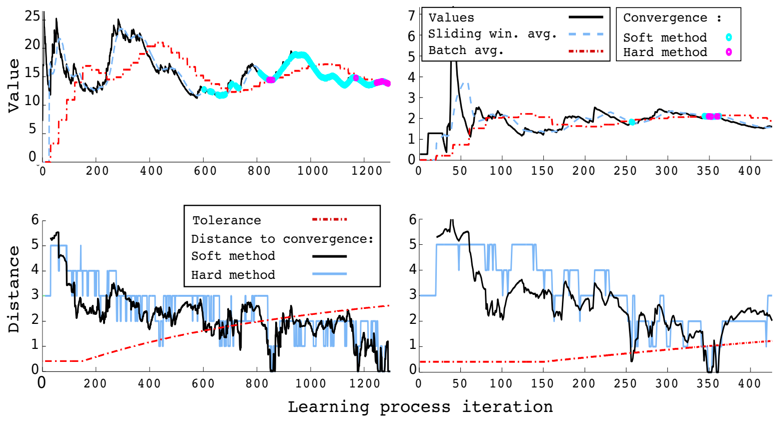


Figure 1: Early convergence detection. Upper figures show examples of learned variables values and their respective averages while dots represent the detection of a soft or hard convergence. Distance to convergence are shown with plain line in lower figures while the red dotted line depicts the tolerance threshold $\epsilon^{(k)}$ of the soft method and its relaxation.

Detecting the convergence of these variables without increasing the computational cost of the algorithm is challenging given that their approximated value through time may be highly volatile and sometimes nearly periodic (Fig. 1). Therefore the convergence detection at iteration $k$ is based on multiple measures of stability $\upsilon_i^{(k)}$ on different temporal scales such as the current value, the last $l$ values (sliding window) or the batch of average values (fixed batch windows). These measures $\upsilon_i^{(k)}$ represent, for instance, the distance between the current value and the average of the sliding window or the relative standard deviation of the batch windows.

Using these measures, the hard convergence of the learning phase can be defined as

$$\bigcap_{i=1}^{m}(\upsilon_i^{(k)} < \tau_i),$$

with $\tau_i$ being pre-defined convergence thresholds, each associated to one measure $\upsilon_i^{(k)}$. However, this condition of convergence may be difficult to achieve given that all the separate criterion must be fulfilled at the same time. Therefore a softer approach was designed to accept a small divergence $\epsilon$ on the set of thresholds $\tau_i$ such that

$$\left[\sum_{i=1}^{m} \max\left(\frac{\upsilon_i^{(k)} - \tau_i}{\tau_i}, 0\right)\right] < \epsilon.$$

In addition of being more flexible, this approach enables the relaxation of $\epsilon$ as the number of iterations $k$ increases and thus offers an interesting leverage to loosen the convergence criterion. This relaxation defines $\epsilon$ as

$$\epsilon = \beta_1 \cdot \beta_2 \cdot \epsilon_0,$$

where $\beta_1$ is the relaxation factor local to each of the learned variables and $\beta_2$ is the global relaxation factor that consider the overall state of the learning phase.

The local relaxation factor $\beta_1$ ensures that after a certain amount of time spent learning a given variable, $\epsilon$ is gradually relaxed such as to facilitate the convergence detection for volatile variables. Therefore, $\beta_1$ is defined as

$$\beta_1 = \max\left(1, \frac{k}{K_{lim}}\right)^{\Gamma_1},$$

where $K_{lim}$ defines a threshold of iterations after which the local relaxation should occur and $\Gamma_1$ tunes the rate of relaxation increase through time.

The global relaxation factor $\beta_2$ avoid that the overall adaptive process get stuck in the learning phase because of a small amount of non-converging variables. Assuming that a proportion $\rho$ of the approximated variables has been detected as having converged, the factor $\beta_2$ is given as

$$\beta_2 = \max\left(1, \frac{\rho}{\rho_{lim}}\right)^{\Gamma_2},$$

where $\rho_{lim}$ defines the threshold of proportions required to activate the global relaxation and $\Gamma_2$ serves the same purpose as $\Gamma_1$.

## 1.3 Creating smaller blocks

In theory, updating all parameters during one step of the M-H algorithm with an adequate proposal kernel offers the best possible mixing [2]. However, in practice, proposal kernels moving large amount of parameters affect negatively the efficiency and the stability of our framework. Indeed, its efficiency was identified to depend on the computational cost of linear algebraic operations that exponentially grows with the number of parameters. In addition to this unpleasant effect, the stability of adaptive proposals suffers from the difficulty to accurately approximate high-dimensional covariance matrices.

A natural solution addressing these problems is to consider the use of blocks containing small subsets of the $d$ parameters. Each of these blocks would propose moves on their parameters according to a proposal kernel dedicated to the block. However, to adequately benefit from this readily applicable solution, several considerations on the composition and size of the blocks have to be taken into account.

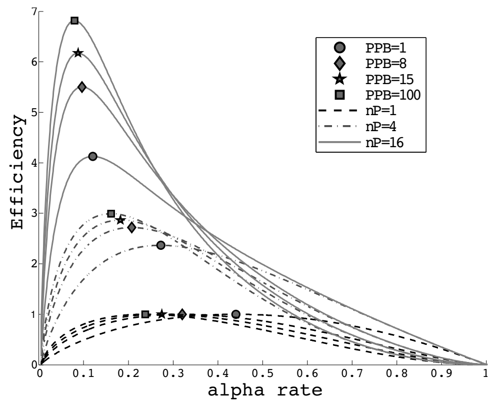


Figure 2: Alpha's efficiency, representing the average expected samples per prefetching iteration, in function of the number of parameters per block and processors

Indeed, the target acceptance rate $\alpha_\star$ defining the efficiency of our framework decreases as the number of processors increases (Fig. 2). However, small $\alpha_\star$ tends to produce large and bold moves that are difficult to tune reliably when few parameters are updated. Fortunately, increasing the number of parameters per block tends to limit the boldness of these moves. Indeed, assuming that the parameters in a block are independent, the optimal acceptance rate of a block of parameter identified by $i$ is given by

$$\alpha_\star^i \approx \prod\nolimits_{j=1}^{d^{(i)}} \alpha_j \tag{1}$$

where $d^{(i)}$ is the size of the block $i$ and $\alpha_j$ stands for the acceptance rate of the univariate moves along direction $j$. Therefore, parameters blocks should be big enough to benefit from the previously identified stability induced by large blocks.

Moreover, the creation of blocks of parameters, or *blocking* [3], is often used to decompose the likelihood in smaller sub-likelihoods, if the model permits. These sub-likelihoods define conditional probabilities that compose the full likelihood and an update of their assigned parameters enables their sampling at a lower computational cost.

All these considerations affecting the efficiency of the resulting sampler must be taken into account to define the optimal size of a parameters block. This daunting task is therefore either avoided by solely considering univariate proposal kernels or delegated to the end user. For this last option, general guidelines, obtained from empiric experiments with our framework, advises users to define block size in the range of 10 to 50 parameters.

Given that such block of parameters are not sufficiently large enough to consider that their size $d^{(i)}$ tends to infinity, the pre-fetching performance model defined in Eq. 4 (main article) has to be slightly adapted. A small correction on $E^1(\alpha^{(i)})$ is applied to take into account the block size $d_i$ as follows

$$E^1(\alpha^{(i)}) \propto \alpha^{(i)} \cdot \left[\varphi^{-1}\left(\frac{\alpha_i}{2}\right)\right]^\rho$$

with $\rho = 2 - 1.1e^{-d_i/10}$ and where $\varphi$ stands for the standard normal distribution. This correction is based on empirical observations made by Roberts *et al.* [4] and the one made during experiments with our framework.

Additionally to the block size, one last important aspect regarding the definition of block remains unaddressed: the choice of parameters that must be grouped together. This choice is an even bigger challenge than the one caused by the block size. Indeed, the mixing of the MCMC sampler depends directly from this choice given that it determines, by choosing which parameters to group in blocks, the parameters correlations that can be exploited.

## 2 Results

**Cluster setting.** All presented analysis were done on a dedicated cluster partition of Intel Xeon X5650 (2.67GHz) nodes with 64GB of RAM at the

University of Geneva.

## 2.1 Multivariate normal based models

Multivariate normal distributions are used to validate and study the behaviour of the methods previously discussed. The likelihood function used is the probability of the data $X$ formed by $m$ samples generated from a multivariate normal distribution $\mathcal{N}(\mu, \Sigma)$ with $\mu \in \mathbb{R}^d$ and $\Sigma \in \mathbb{R}^{d \times d}$. Setting $\Theta = (\mu, \Sigma)$, the likelihood function becomes $f(X|\Theta) = \prod_{k=1}^{m} f_\Theta(x_k)$ with $f_\Theta$ being the probability density function of the multivariate normal distribution given the parameters $\Theta$. This model is used first with a set of $d$ independent normal distributions such that $\Sigma = diag(V)$ with $V = \left(\sigma_1^2, .., \sigma_d^2\right)$ and thus $\Theta = (\mu, V)$ is sampled. Second, we use covariance matrices $\Sigma$ expressing the correlations between several of the $d$ dimensions of the distribution. For this variant, we sample $\Theta = (\mu)$ while $\Sigma$ is fixed.

### 2.1.1 Validation against an optimal proposal.

Our adaptive methods were more effective than the sub-optimal proposal distributions (`PF`) at sampling the parameters $\Theta = (\mu, V)$ of a simple model formed by independent normal distributions (Fig. 3a). Moreover, they were able to accurately learn the optimal proposals since their average ESS was comparable to the one of the optimal choice of the proposal distributions (`OPPF`). The `MIXED` and `PCA` methods showed a similar behaviour than `STD` on this model since nearly no correlation were detected to trigger the switch from the adaptive proposal based on independent parameters (Eq. 3 of main article) to the one exploiting correlations (Eqs. 1 or 2 of main article).

The improvements in ESS as a function of the number of processors was far from linear and represents well the limitation defined in the efficiency Eq. 4 (main article) of the pre-fetching method. This equation balances the sampling efficiency $E^1(\alpha)$ and the estimated amount of samples produced per iteration $D(\alpha, n_p)$. The former is optimal for a set $\alpha$ while the latter depends on $\alpha$ but also $n_p$. Indeed, as the number of processors $n_p$ grows, we wish to increase the amount of samples $D(\alpha, n_p)$ by improving the quality of prediction. This can be achieved by reducing $\alpha$ but it contradicts the previous statement that $\alpha$ should be constant to optimize $E^1(\alpha)$. This phenomena is well explained by the slow decay of the optimal target $\alpha_\star$ in function of $n_p$, which leads to a rapid loss of sampling information per iteration (inverse of ACT; Fig. 3b) and a steady increase of samples kept per iteration (Fig. 3c).

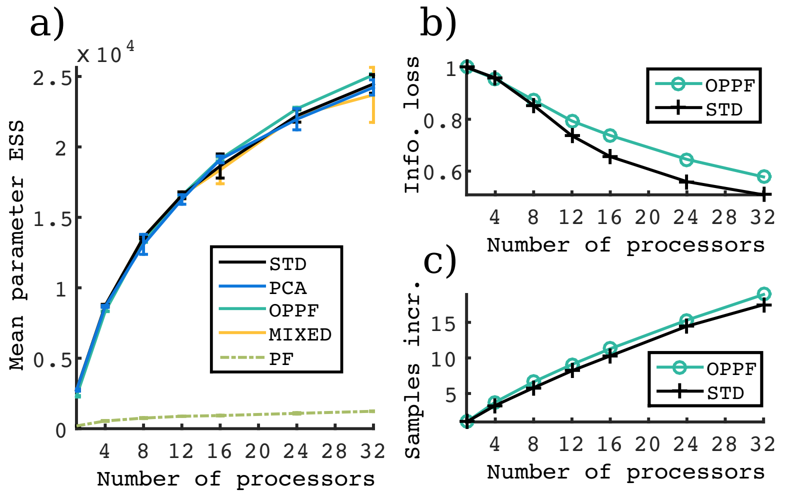


Figure 3: Averaged result for independent normal models with $d = 20$, $3 \times 10^3$ iterations and 5 runs. Left figure shows the mean ESS per parameter in function of the number of processor for each method (error bars represent standard deviation). Right figure represent the reduction in sampling effectiveness (*top*) and increase of the number of samples per iterations (*bottom*) as the number of processor increases.

### 2.1.2 Exploiting parameters correlation.

The `PCA` method outperformed significantly all other methods when sampling the parameters $\Theta = (\mu)$ of multivariate normal distributions with known $\Sigma$ expressing several correlations (Fig. 4a). All the adaptive methods showed comparable or superior average ESS than the `OPPF` method that performed independent normally distributed moves with the optimal variances. Since the `PF` method performed extremely poorly compared to the other methods, we discarded its results.

While the `MIXED` method obtained slightly better results than `OPPF`, it underperformed compared to `PCA` due to the component-wise scaling of the covariance matrix $\Sigma$ along each dimension, which impaired the information about correlations. This behaviour and the one of the different methods is well illustrated by the standard deviation of ACT over the number of processor (Fig. 4b). By correctly exploiting the observed correlations, `PCA` was the only method to carry out a balanced sampling over all parameters. The other methods over and under-sampled some of them, which lead to high variance in the ACT.

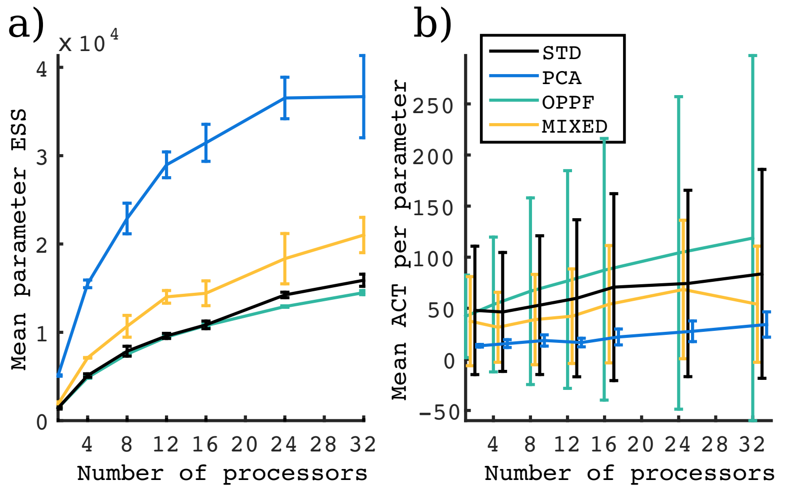


Figure 4: Averaged results for a correlated multivariate normal with $d = 20$, $6 \times 10^3$ iterations and 5 runs. The figure on the left shows the mean ESS per parameter as a function of the number of processor for each method. The figure on the right represents the average ACT per parameter. Error bars represent the standard deviation.

### 2.2 Performance gain on *PyRate* model

In addition of the empiric dataset used in the main article, our framework was challenged on this model using computer simulations. This dataset was simulated using the simulator of the *PyRate* model [5]. Close to 4,000 fossil occurrences assigned to 203 species were simulated over a 30 millions years (myr) time span. The preservation rate was set to 2.0 while two different speciation and extinction rates were defined over time. The speciation rate started at 0.4 from 30 to 20 myr and then reduced to 0.1, while the extinction rate started at 0.05 from 30 to 15 myr and then augmented to 0.4.

#### 2.2.1 Mixing efficiency.

Due to the targeted learning, adaptive methods were more efficient at sampling the *PyRate* model with simulated data when compared to `PF` and even surpassed slightly `OPPF` (Fig. 5). Among adaptive methods, `PCA` showed the best performance because of the correlation that existed in the model. Indeed, while this amount of correlation is rather small, `PCA` was able to detect and exploit it.

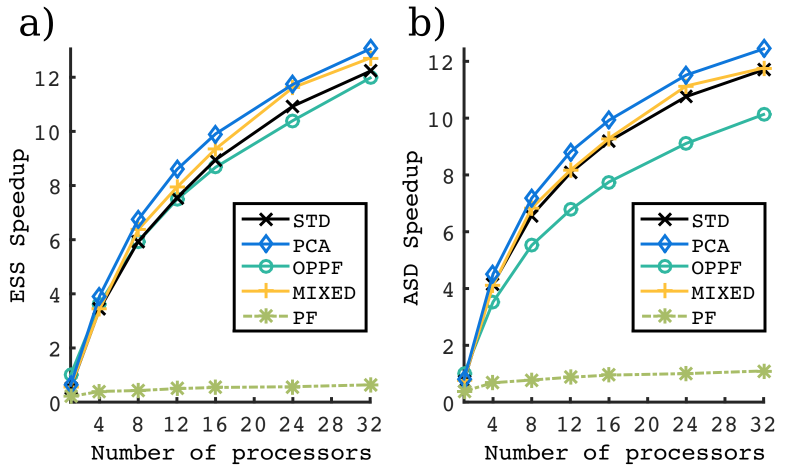


Figure 5: Speedup for *PyRate* model on simulated data using `OPPF` with 1 processor as the reference. The figure shows the ESS, respectively average squared distance (ASD) between moves, speedup in function of the number of processors for each method. Settings for this simulation were $d \approx 400$, $4 \times 10^6$ iterations and 4 runs.

Since the efficiency of the framework not only depends on the performance gain of the methods but also on their added computational cost and communications overhead, we compared the performance of our new methods with the reference non-adaptive method `OPPF` without pre-fetching (see Eq. 5 of main article). In this case, `OPPF` was set with the mean variance of each type of parameters (rates, times). In contrast, `PF` represented a sub-optimal choice with mean variance wrongly estimated by a tenfold factor.

The overheads due to processor communication of the pre-fetching methods amounted, on average, to 3% (4 processors) and 10% (32 processors) of the time spent per iteration. In contrast, the cost of the adaptive phase decreased from 13% (1 processor) to 6% (32 processors) and this decrease came from the faster convergence of the chain, the parallel component-wise update and the added information provided by the adaptive process. This decrease of the adaptive phase cost directly highlights how the adaptive proposals benefit from their coupling with pre-fetching.

### 2.2.2 Impact on the burn-in.

The length of the burn-in phase was assessed using the multivariate *potential* scale reduction factor $\hat{R}$ [1], which compares the within and pooled variance of multiple MCMC runs. This measure highlighted that adaptive methods were much faster than their non-adaptive counterpart to reach the MCMC convergence on the *PyRate* model with simulated data (Fig. 6). This is of utmost importance since the time required to converge towards the equilibrium of the Markov chain, or burn-in, impacts significantly the sampling of MCMC methods. Indeed, samples derived from this phase cannot be exploited. Moreover, empirical observations showed that the convergence of the adaptive process occurred after the equilibrium was reached. This makes it possible to use the convergence of the adaptive process as an indication for the burn-in and to start sampling only when it is detected.

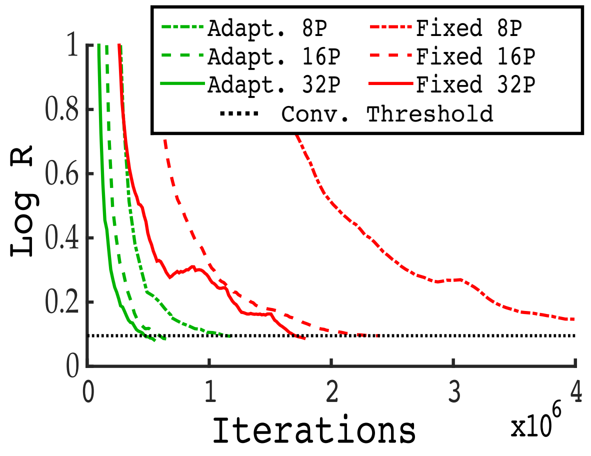


Figure 6: Convergence of adaptive versus non-adaptive methods on the *PyRate* model. The figure represents the *potential* scale reduction factor $\hat{R}$ based on 5 MCMC runs on simulated data. The black dotted lines represent the convergence threshold $\hat{R} = 1.1$.

A clear pattern about the learning process emerged from the large amount of experiments that we performed. First, the learned variables ($\Sigma$, $\lambda$, etc.) varied quickly during the burn-in, which created *wide* moves and improved the speed of convergence of the Markov chain. Second, the covariance matrix $\Sigma$ converged to its final value and the scaling factors $\lambda$ and $\Lambda$ continued to be adapted once the equilibrium was reached. Third, $\lambda$ and $\Lambda$ converged when the target acceptance rate $\alpha_\star$ was reached.

### 2.2.3 Block size

The size of parameters blocks were previously identified as playing an important role in the efficiency our framework. These theoretical postulates were confirmed by observations made during experiments on the `PyRate` model. On this model the use of blocks of parameters had a significant influence on the sampling performances. The ESS per second as a function of the size of blocks indicated that a block should have at least 2 parameters (Fig. 7). Indeed, blocks were made such as to group the two parameters related to the birth and death times of a species. This grouping enabled the strong correlation between both parameters to be exploited by our framework.

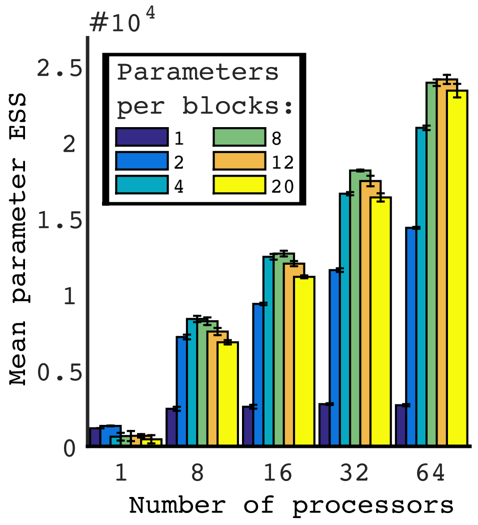


Figure 7: Illustrations of the effect of block size on the `PyRate` model on simulated data. This figure shows the averaged effective sample size per second (with its standard deviation) of the `PCA` method in function of the number of parameters per block as well as the number of processors.

A second, less striking, observations could be made from these measures for blocks having at least two parameters: the optimal blocks size slightly changed with the number of processors employed (Fig. 7). For low amount of processors, small blocks were more efficient given that the computational overhead of the adaptive phase was low. However, as the number of processors increased, larger blocks became more and more efficient due to the parallel estimation of the component-wise scaling factors that scale with the number of processors as well as the increased stability of the global scaling factor (Eq. (1)).

## 2.3 Detailed experiments settings and measures for substitution models

`MrBayes` measures were done with the version 3.2.5 compiled with SSE support and MPI enabled.

### 2.3.1 Codon-substitution models

**Settings** Two different phylogenetic trees were simulated using `INDELible` : the first one `Dataset 1` had 16 taxa and the second one `Dataset 2` had 32 taxa. For each phylogenetic tree, two separate alignments of 100 codons were simulated under mild purifying selection ($\omega = 0.8$).

Each method was run 4 times per simulated alignments with different starting random seed for $200 \cdot 10^3$ iterations. Therefore each dataset consisted of 8 independent MCMC runs. Measures were applied on the obtained samples after removing a burn-in phase of $40 \cdot 10^3$ samples (20%). The presented ESS and run time on each dataset correspond to the average measures over all respective dataset runs.

**Measures** Detailed measures for these experiments are shown in table 1 and 2

### 2.3.2 Convergence measures on phylogenetic tree distributions.

**Settings** Two empirical DNA datasets available on TreeBASE were used for this experiment :

- `M2017` (legacy ID `M336`) has 27 taxa and 1949 sites;
- `M2152` (legacy ID `M520`) has 67 taxa and 1098 sites.

Twenty runs with different seeds were simulated for each different methods for $\approx 6 \cdot 10^6$ iterations for `M2017` and $\approx 4 \cdot 10^6$ iterations for `M2152`. These runs started from the same initial random tree (generated by MrBayes). For each method, we found sufficient to drop the top three worst runs in order to remove run failing to reach convergence.

The average standard deviation of split frequency was measured according to the following protocol :

Table 1: Detailed results for `Dataset 1`

**MrBayes**

| Nb. Proc. | 1 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|
| Avg. ESS | 446.69 | - | - | - | - |
| Avg. Time [s] | 2189.93 | - | - | - | - |

**Default 1PPB**

| Nb. Proc. | 1 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|
| Avg. ESS | 692.82 | 1010.42 | 913.94 | 900.73 | 801.78 |
| Avg. Time [s] | 971.72 | 891.84 | 839.13 | 795.98 | 859.40 |

**PCA 12PPB**

| Nb. Proc. | 1 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|
| Avg. ESS | 1006.67 | 3329.60 | 4972.90 | 6446.45 | 7574.92 |
| Avg. Time [s] | 2008.97 | 1967.85 | 1871.64 | 1950.95 | 2009.62 |

Table 2: Detailed results for `Dataset 2`

**MrBayes**

| Nb. Proc. | 1 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|
| Avg. ESS | 212.84 | - | - | - | - |
| Avg. Time [s] | 3676.83 | - | - | - | - |

**Default 1PPB**

| Nb. Proc. | 1 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|
| Avg. ESS | 356.85 | 479.40 | 432.09 | 371.37 | 334.15 |
| Avg. Time [s] | 1196.94 | 1090.12 | 980.53 | 915.22 | 980.07 |

**PCA 12PPB**

| Nb. Proc. | 1 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|
| Avg. ESS | 556.51 | 1860.64 | 2743.32 | 3494.99 | 4021.09 |
| Avg. Time [s] | 2556.18 | 2405.78 | 2274.70 | 2191.97 | 2393.21 |

1. we insured that the split frequencies were the same for long MCMC run of `MrBayes` and our own implementation;
2. this reference split frequencies were then used to define the pairwise ASDSF of each run of the experiment.

The pairwise ASDSF was computed using tree log files resulting from `MrBayes` and our implementation by a tool made for the occasion in order to insure comparable results.

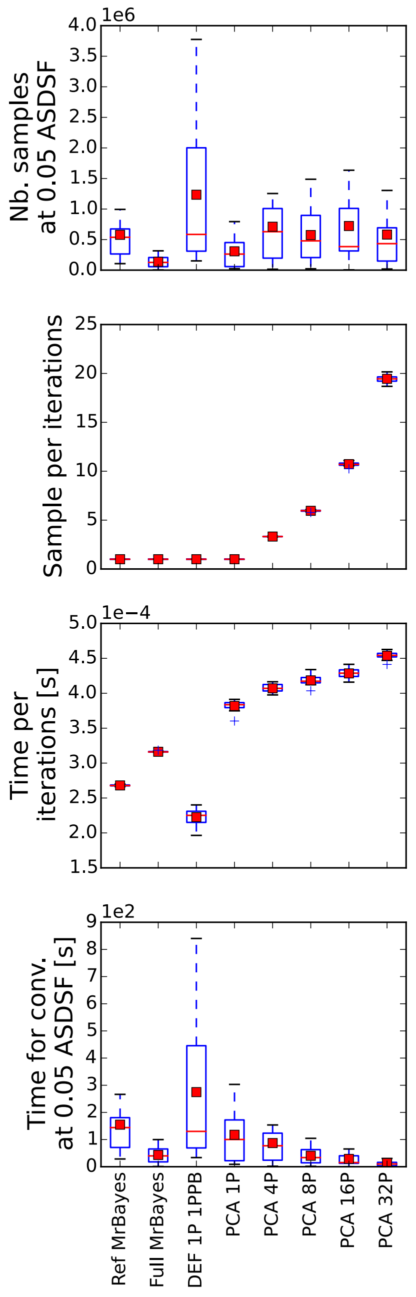
**Measures** Detailed measures for these experiments are shown in figure 8 for empirical dataset `M2017` and figure 9 for empirical dataset `M2152`. Each figure has four plots representing in descending order :

1. the number of **sample** required to reach the threshold of 0.05 ASDSF;
2. the number of **samples** per **iterations** (gain from prefetching);
3. the time required to compute one **iteration** (total number of iterations divided by total run time);
4. the time required to reach threshold of 0.05 ASDSF.

The red squares represent the average while red lines represent the median.

Figure 8: Detailed measures for empiric dataset `M2017`

(a) Without MC$^3$



(b) With MC$^3$

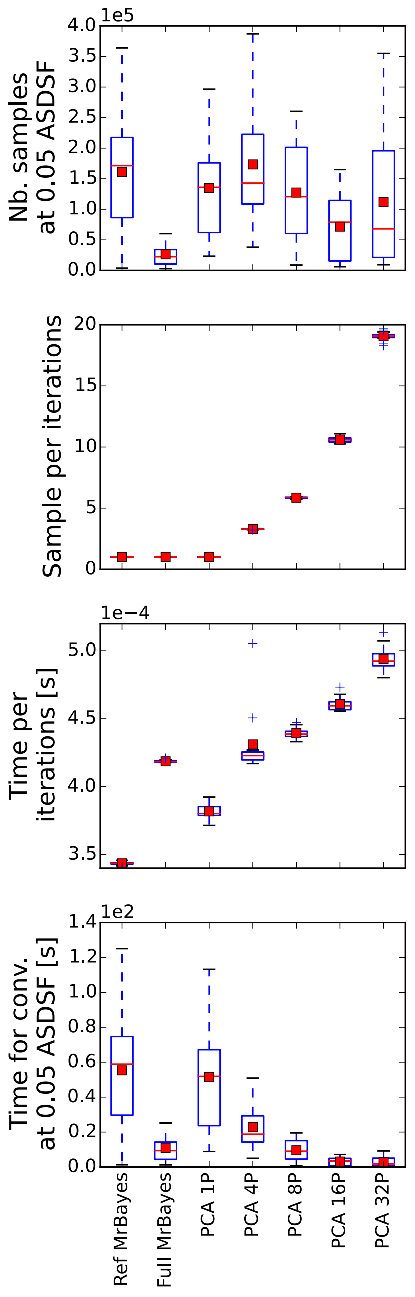
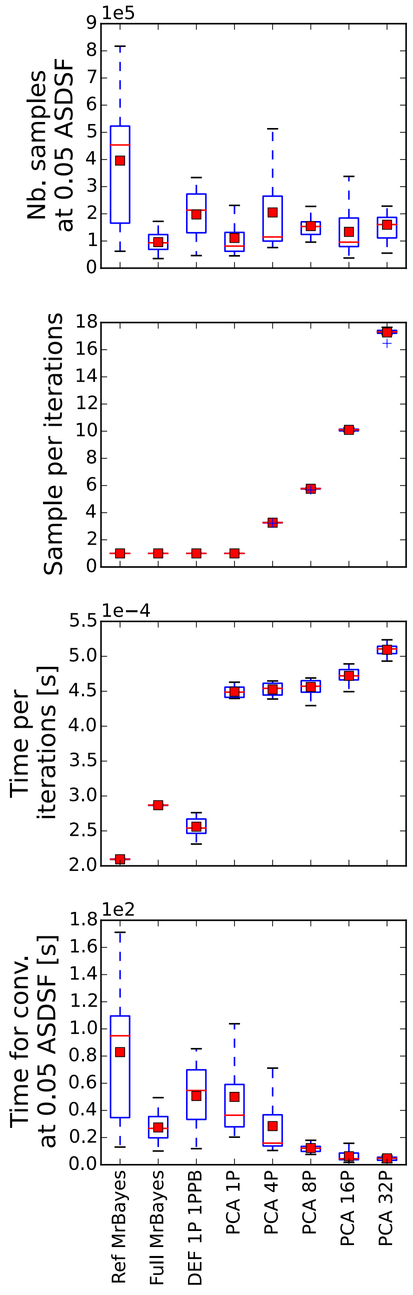
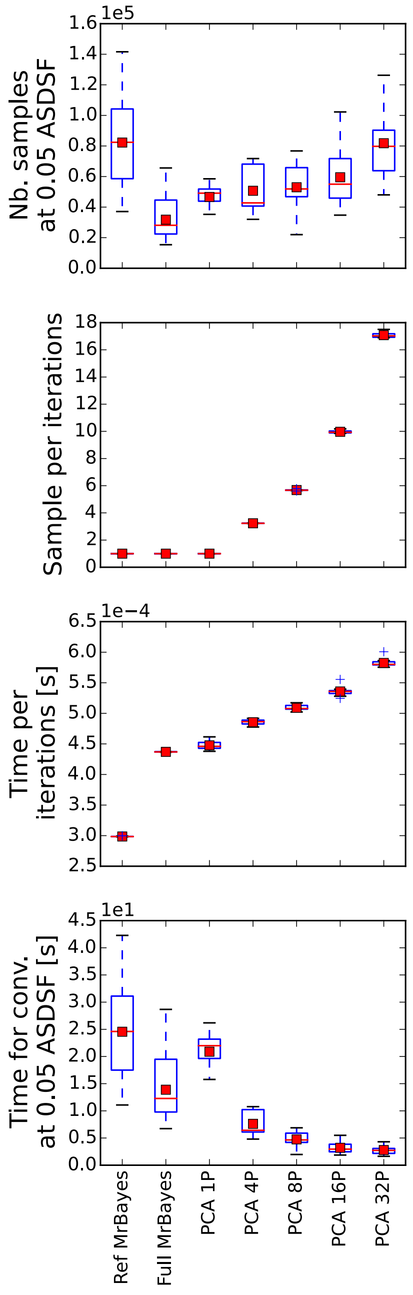


Figure 9: Detailed measures for empiric dataset `M2152`

(a) Without MC$^3$



(b) With MC$^3$



## References

[1] Brooks, Stephen P., and Andrew Gelman. "General Methods for Monitoring Convergence of Iterative Simulations." Journal of Computational and Graphical Statistics 7, no. 4 (December 1, 1998): 434–55. doi:10.2307/1390675.

[2] Gamerman, Dani, and Hedibert F. Lopes. Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference, Second Edition. CRC Press, 2006.

[3] Roberts, G. O., and S. K. Sahu. "Updating Schemes, Correlation Structure, Blocking and Parameterization for the Gibbs Sampler." Journal of the Royal Statistical Society: Series B (Statistical Methodology) 59, no. 2 (1997): 291–317. doi:10.1111/1467-9868.00070.

[4] Roberts GO, Rosenthal JS (2001) Optimal scaling for various Metropolis-Hastings algorithms. Statist Sci 16(4):351–367.

[5] Silvestro, Daniele, Jan Schnitzler, Lee Hsiang Liow, Alexandre Antonelli, and Nicolas Salamin. "Bayesian Estimation of Speciation and Extinction from Incomplete Fossil Occurrence Data." Systematic Biology, February 8, 2014, syu006. doi:10.1093/sysbio/syu006.