A Bayesian Optimization through Sequential Monte Carlo and Statistical Physics-Inspired Techniques

Anton Lebedev[1], M. Emre Şahin[1], and Thomas Warford[2]

[1] The Hartree Centre, Keckwick Ln, Warrington, UK {anton.lebedev, emre.sahin}@stfc.ac.uk
[2] University of Manchester, M13 9PL, Manchester, UK thomas.warford@student.machester.ac.uk

Abstract. In this paper, we propose an approach for an application of Bayesian optimization using Sequential Monte Carlo (SMC) and concepts from the statistical physics of classical systems. Our method leverages the power of modern machine learning libraries such as NumPyro and JAX, allowing us to perform Bayesian optimization on multiple platforms, including CPUs, GPUs, TPUs, and in parallel. Our approach enables a low entry level for exploration of the methods while maintaining high performance. We present a promising direction for developing more efficient and effective techniques for a wide range of optimization problems in diverse fields.

Keywords: Stochastic Methods · High-Performance Computing · Bayesian Inference

1 Introduction

Bayesian optimization of ever-growing models has become increasingly important in recent years and significant effort has been invested in achieving a reasonable runtime-to-solution. Unfortunately, most optimization tasks are implemented and optimized within a specific framework, resulting in a single optimized model. The proliferation of such implementations is difficult, as it requires both domain expertise and knowledge of the specific framework and programming language.

Probabilistic programming frameworks such as Stan [6] and (Num)Pyro [4], provide such support for efficient optimisation methods such as Hamiltonian Monte Carlo (HMC) algorithm which can explore complex high-dimensional probability distributions. These frameworks are powerful tools for statistical analysis and inference.

Stan excels at handling complex hierarchical models with ease, which is often challenging in other probabilistic programming frameworks. Its user-friendly interface makes it accessible to those with little experience in Bayesian inference and statistical modelling, making it a popular choice for the accurate and

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_49

efficient analysis of complex models. It provides domain experts with a performant tool to perform the said task with little to no programming knowledge. It achieves this by defining a "scripting language" for models and translation of these into C++ code. It suffers, however, from a lack of inherent parallelism and a formulation of its methods in heavily-templated C++. NumPyro, built on JAX [5], enables efficient exploration of high-dimensional probabilistic models using different methods, while JAX itself combines the flexibility and ease-of-use of NumPy with the power and speed of hardware accelerators for efficient numerical computing. Additionally, JAX offers compatibility and portability by supporting code execution on a variety of hardware.

Inspired by HMC and SMC descriptions in [7,8], we implemented an HMC algorithm in Python using the NumPyro and JAX frameworks and with DeepPPL [1] utilized to translate existing Stan models into their Python equivalents. In this paper we present preliminary findings from our implementation of SMC for Bayesian parameter searches developed with its physical origins intact.

2 Method Description

Upon review of the SMC algorithm [7, 8] and its HMC kernel, it has become apparent that the approach resembles the cooling process of an ideal gas in a potential field with unknown minima, and that the algorithm's development would benefit from an understanding based on physical systems. To facilitate this understanding, we have undertaken a reformulation of the SMC and HMC algorithms to more accurately reflect the particle ensembles of statistical physics. A simple first step was the reintroduction of a temperature into the expressions for probability, seeing as the probabilities used in these methods are the maximum-entropy probabilities of an ensemble (collection) of particles at a fixed energy:

$$p_i = \frac{e^{-E_i/(k_B T)}}{\sum_{j=1}^{N} e^{-E_j/(k_B T)}} \ . \tag{1}$$

Here k_B is the Boltzmann constant (carrying the dimensions of energy per degree of temperature), T the temperature and $E_i = \mathcal{H}(q_i, p_i)$ is the energy of the particle i at position q_i with a momentum p_i given a Hamiltonian:

$$\mathcal{H}(q, p) = \frac{p^2}{2m} + V(q) \ . \tag{2}$$

As is common, the Hamiltonian encodes the dynamics of the system. Since we seek to determine the parameters that maximise the log-likelihood of the model the potential can be defined [7] as:

$$V(q) := -\ln(P(q|X)) \ , \tag{3}$$

where $P(q|X)$ is the posterior probability density of the model, where X is the vector containing the observations and q is the (position) vector in parameter space - the parameters of the model.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_49

The formulation of (1) commonly used in mathematics implies $T = \frac{1}{k_B}$ or an arbitrary definition of $k_B := 1$. The former fixes a degree of freedom of the method, whilst the latter allows for a variation, but removes the dimensionality of the constant which, in conjunction with T ensures that the argument of the exponential remains a dimensionless number or quantity. Whilst *functionally* of no consequence retaining the physical dimensions of the respective quantities allows for sanity checks of formulae during development.

The distribution of the "auxiliary momentum" described in [7] is naturally dependent on T, with a higher temperature T resulting in a broader distribution of the momenta (faster particles) and a wider area of the initial sample space covered.

The HMC process is rather simple and described in [7] in sufficient detail. In contrast, sequential Monte-Carlo is provided in a less legible form in [8]. Hence we privde here the simple version we turn our attention to:

1. Initialise the position samples to a normal distribution on $\mathbb{R}^n$ and the momentum samples to a normal distribution with temperature T.
2. Iterate for a given number of SMC steps:
   (a) Propagate each particle in the ensemble using HMC.
   (b) Determine the lowest energy for all particles, subtract it from every energy (renormalisation) and store it for future processing.
   (c) Compute and store the average parameter value.
   (d) Determine the effective size $N_{effective}$ of the ensemble according with [8].
   (e) If resampling is necessary select $N_{effective}$ particles with largest weights (lowest energies) and duplicate them according to their probability until the original ensemble size is reached. Then reset the momenta to a thermal distribution and the weights of the particles to $\frac{1}{N}$.
3. Compute the moving average of the stored averages, using the stored energies as weights in accordance with (1).

Here we must note the rescaling of the weights by subtraction of the lowest energy, which is physically motivated by the freedom to choose the origin of the energy scale. Similarly, the weighted moving average in the last step results in a smoothing of the excursions of the mean after a resampling step by weighting means with large associated energies exponentially smaller (the higher the energy the more unlikely a configuration is to occur).

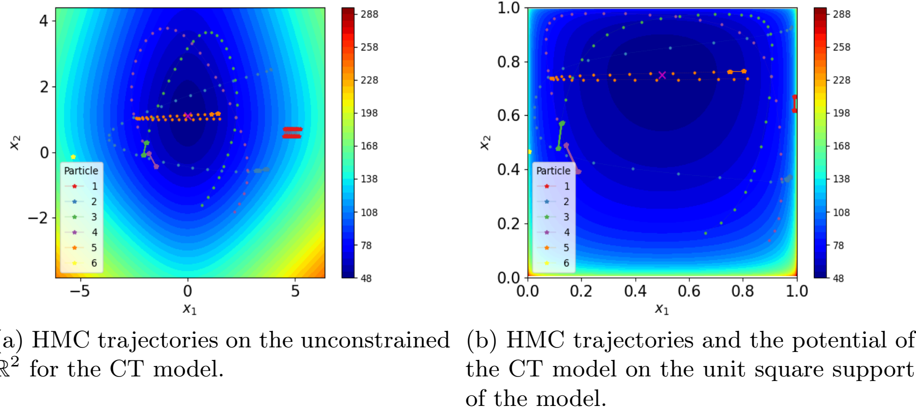
3 Numerical Experiments

3.1 Models

To demonstrate the effectiveness of the implementation we have selected two simple models:

1. A sequence of M independent tosses of two coins - the Coin Toss (CT) model.
2. Item Response Model with Two-Parameter Logistic (IRT 2PL).

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_49



(a) HMC trajectories on the unconstrained $\mathbb{R}^2$ for the CT model.

(b) HMC trajectories and the potential of the CT model on the unit square support of the model.

Fig. 1: Trajectories of our HMC implementation in the potential defined by (3) for the CT model. Note the dense zig-zag trajectories that result if the momentum inversion proposed in [7] is implemented.

CT model The CT model assumes complete ignorance of the a-priori coin bias p and its maximum a-posteriori probability estimator can be determined formally to be:

$$\hat{P}_{MAP} = \frac{K}{N} \,. \tag{4}$$

Here K is the number of observed heads and $N = 40$ is the total number of observations. This allows us to check the numerical approximation of (3) as well as its gradients, ensuring proper functioning of the implementation. The true parameters of the coin bias for each of the two coins are

$$p_1 = \frac{1}{2} \,,\; p_2 = \frac{3}{4} \,. \tag{5}$$

The potential of the CT model, along with a few sample trajectories of the particles propagating therein are shown in fig. 1a prior to the constraining to the support of the model, and in fig. 1b after. Here we note that we chose *not* to invert the momentum in case the new phase-space point (q, p) passes the Metropolis-Hastings acceptance test, contrary to alg. 1 of [7]. As can be seen in the figures, such an inversion results in a rather slow sampling of the potential in the case of a smooth potential. It is, however, beneficial for a rough potential and hence likely better in most practical applications.

IRT 2PL Item Response Theory (IRT) [2] is a statistical model that is widely used in a variety of research fields to analyze item responses in assessments or surveys. The two-parameter logistic (2PL) model is a specific type of IRT model that assumes each item has two parameters: the difficulty parameter and the

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_49

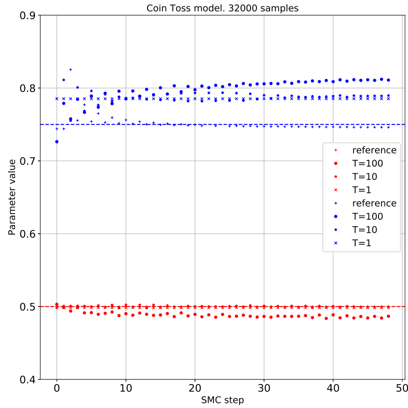


Fig. 2: Parameter estimates obtained with SMC with physics-motivated moving averages. The dashed lines represent the true parameter values. The crosses refer to a reference implementation not available to the public.

discrimination parameter. The model assumes that the item responses y_i for $i = 1, \ldots, I$ are Bernoulli distributed with a logit link function. The probability of responding correctly to item i is given by:

$$P(y_i = 1|\theta, a_i, b_i) = \frac{1}{1 + \exp(-a_i(\theta - b_i))} \tag{6}$$

where θ is the person's latent ability, a_i is the discrimination parameter for item i, and b_i is the difficulty parameter for item i.

3.2 Experimental Results

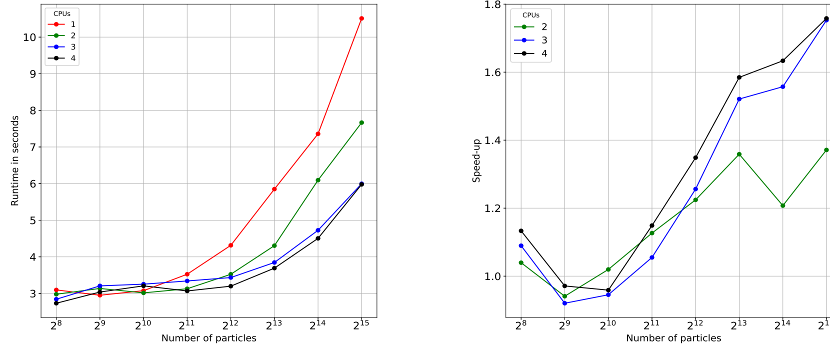
Given the physical interpretation of the SMC iteration and the weighted average derived therefrom we expected - a-priori - a rather rapid and smooth convergence towards the true parameter values. Given a smooth potential, as in the case of the CT model, it is furthermore expected that convergence to the estimated parameters is faster for lower temperatures T.

Quality of Estimation As can be seen in fig. 2 , all estimates of the bias of a fair coin converge within 5 iterations to the true value $\frac{1}{2}$ (indicated by the dashed red line). In case of a rather high temperature (more precisely: thermal energy) of $T = 10\frac{1}{k_B}$ the apparent limit value deviates noticeably from the true value, given the remarks above this is not unexpected and will be remedied in a future iteration of the method.

In the case of the biased coin, with $p_2 = \frac{3}{4}$ it becomes obvious that the current development stage of the method may suffer from a freeze-in (c.f. $T =$

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_49

1 case) resp. a bias in the implementation. For a comparison reference, non-public, implementation results are marked as '+', showing a similar convergence behaviour but without the offsets.



(a) Execution time of HMC for the Coin Toss model.

(b) Speed-up in comparison to 1 CPU core for the Coin Toss model.

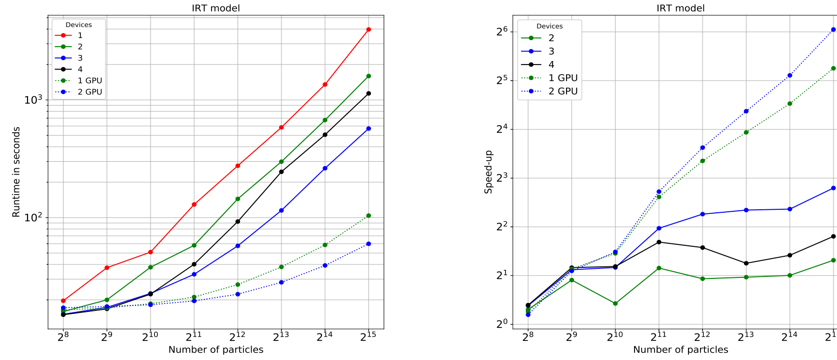
Fig. 3: Execution time and speed-up of our HMC implemetation when using mutliple CPU cores of a Ryzen 5 3600X to determine the coin biases of the CT model using HMC with 1000 Leap-Frog steps.

Performance - HMC Although a reference implementation is available for comparison, its drawbacks are its complexity and limited execution architectures.

Our development utilises JAX and NumPyro, allowing us to run the method for variable models on a variety of architectures. Here we present the performance data obtained for the pure HMC implementation on different architectures. One can observe, in fig. 3a the behaviour of the run time when running our version of HMC on multiple CPUs, parallelised via MPI. For the CT model run times for up to 2048 particles are dominated by communication and data management overhead. This is a surprisingly small number, given the simplicity of the model and the limited amount of observed data fed into it. Overall the run time growth is sub-linear up until 65538 particles for 1 CPU. The resulting speed-up of using more than one CPU is depicted in fig. 3b, demonstrating a sub-linear increase even with two CPUs. This observation confirms that the model is too small to scale effectively, at least for fewer than 128k particles (c.f. the IRT model below).

In contrast to the CT model is the IRT model, whose runtimes are displayed in fig. 4a. One can immediately see, that the run time is dominated by overhead only below 256 particles *per device* (here: CPU). It is also important to note, that the run time using 4 CPU cores is larger than when only 3 are utilised. We attribute this to the apparent and unintentional spawning of multiple Python

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_49



(a) Execution time for the IRT model. (b) Speed-up in comparison to 1 CPU core.

Fig. 4: Execution time and speed-up of our HMC implementation when using mutliple CPU cores of a Ryzen 5 3600X as well as a RTX 3060 and GTX 1080 Ti to determine the coin biases of the IRT model for HMC with 1000 Leap-Frog steps.

processes per MPI process. Our observations show that each MPI process spawns roughly 2 Python processes. This will be a point of future investigation. This observation explains the apparent super-linear speed-up for the CPUs displayed in fig. 4b. One can also observe the order of magnitude reduction in the overall run time in the case of 65538 particles, when one GPU is used. The sub-linear speed-up from 1 GPU to 2 GPUs can here be explained by the fact, that the devices were asymmetrically bound to the system (PCIE-x16 vs. PCIE-x8) as well as that two devices of different generations were used: a GTX 1080 Ti and an RTX 3060.

The latter point shows the flexibility of JAX and NumPyro, which allowed us to run the same method on multiple CPU cores and multiple, heterogeneous, GPUs using MPI without having to modify the code!

4 Conclusions and Future Work

In conclusion, we have demonstrated that using the NumPyro and JAX frameworks along with intuition from classical mechanics and statistical physics, it is possible to re-create an SMC Bayesian optimisation process, whilst enriching it with an intuitive understanding of the respective steps.

The selected framework enabled us to create a simple, easy to maintain, implementation of HMC and SMC that can be parallelised to multiple computing devices of varying architectures on demand. Our implementation outperforms a similar implementation in Stan by a factor of ~ 2 on a single CPU (core) and

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_49

scales well to multiple CPUs/GPUs, with larger gains obtained for larger models (or models with more observation data) and more sampling particles.

In the near future, we plan to extend MPI parallelism from the core HMC stage to the entire SMC iteration, as well as perform a thorough performance analysis and optimisation of our code, since preliminary checks indicate the existence of, e.g., unnecessary host-device data transfers.

On the theoretical side we plan to continue the reformulation of SMC in the language of statistical physics and expect the possibility to include maximum-entropy methods and thermalisation/annealing into the framework, utilising the long history of MC in physics [3]. Having a formulation of SMC that utilises terminology of (statistical) physics we hope to be able to extend the approach from the classical onto the quantum domain. This holds the potential of including existing quantum resources into Bayesian optimisation processes without requiring the user to know the intricacies of the new architecture.

5 Acknowledgements

During PRACE Summer of High-Performance Computing 2022, TW was able to implement parallel HMC using the discussed formulations and included correctness checks based on physical systems and models with known closed-form solutions in the implementation.

References

1. Baudart, G., Burroni, J., Hirzel, M., Mandel, L., Shinnar, A.: Compiling stan to generative probabilistic languages and extension to deep probabilistic programming. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. pp. 497–510 (2021)
2. Béguin, A.A., Glas, C.A.W.: Mcmc estimation and some model-fit analysis of multidimensional irt models. Psychometrika **66**, 541–561 (2001)
3. Binder, K., Heermann, D.W.: Monte Carlo Simulation in Statistical Physics An Introduction. Springer, 6 edn. (2019)
4. Bingham, E., Chen, J.P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., Goodman, N.D.: Pyro: Deep universal probabilistic programming. The Journal of Machine Learning Research **20**(1), 973–978 (2019)
5. Bradbury, J., Frostig, R., Hawkins, P., Johnson, M.J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., Zhang, Q.: JAX: composable transformations of Python+NumPy programs (2018), http://github.com/google/jax
6. Carpenter, B., Gelman, A., Hoffman, M.D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., Riddell, A.: Stan: A probabilistic programming language. Journal of statistical software **76**(1) (2017)
7. Hoffman, M.D., Gelman, A.: The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. Journal of Machine Learning Research **15**, 1593–1623 (2014)
8. Moral, P.D., Doucet, A., Jasra, A.: Sequential monte carlo samplers. J. R. Statist. Soc. B **68**, 411–436 (2006)

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_49