The School of Mathematics

THE UNIVERSITY
of EDINBURGH

Robust Optimisation Monte Carlo for Likelihood-Free Inference

by

Vasileios Gkolemis

Dissertation Presented for the Degree of
MSc in Operational Research with Data Science

August 2020

Supervised by
Dr. Michael Gutmann

Abstract

Acknowledgments

Own Work Declaration

Here comes your own work declaration

Contents

List of Tables

List of Figures

1 Introduction

1.1 Motivation

Explanation of Simulation-Based Models

A Simulator-Based model is a parameterized stochastic data generating mechanism [2]. The key characteristic is that although we are able to sample (simulate) data points, we cannot evaluate the likelihood of a specific set of observations y_0. Formally, a simulator-based model is described as a parameterized family of probability density functions $\{p_{y|\theta}(y)\}_\theta$, whose closed-form is either unknown or intractable to evaluate. Although, evaluating $p_{y|\theta}(y)$ is intractable, sampling is feasible. Practically, a simulator can be understood as a black-box machine M_r that, given parameter θ, produces samples y in a stochastic manner, i.e. $M_r(\theta) \rightarrow y$.

Simulator-Based models are particularly captivating due to the low-level of restrictions they demand in the modeling; any physical process that can be conceptualized as a computer program of finite (determinstic or stochastic) steps, can be modelled as a Simulator-Based model without any mathematical compromise. This includes any amount of hidden (unobserved) internal variables or logic-based decisions. On the other hand, this level of freedom comes at a cost; performing inference is particularly demanding from both computational and mathematical perspective. Unfortuenately, the algorithms deployed so far, allow the performance of inference only at low-dimensionality parametric spaces, i.e. $\theta \in \mathbb{R}^D$ where D is small.

Example

For underlying the importance of Simulator-Based models, lets use the tuberculosis disease spread example as described in [7]. At each stage we can observe the following events; (a) the transmission of a specific haplotype to a new host (b) the mutation to a different haplotype (c) the exclusion of an infectius host (recovers/dies). The random process, which stops when m infectius hosts are reached, can be parameterized (a) by the transmission rate α (b) the mutation rate τ and (c) the exclusion rate δ, creating a $3D$-parametric space $\theta = (\alpha, \tau, \delta)$. The outcome of the process is a variable-sized tuple y_θ, containg the size of all different infection groups, as described in figure 1. Computing $p(y = y_0|\theta)$ requires tracking all tree-paths that generate the specific tuple along with their probabilities and summing over them. Computing this probability becomes intractable when m grows larger as in real-case scenarios. On the other hand, modeling the data-generation process as a computer program is simple and computationally efficient, hence using a Simulator-Based Model is a perfect fit.

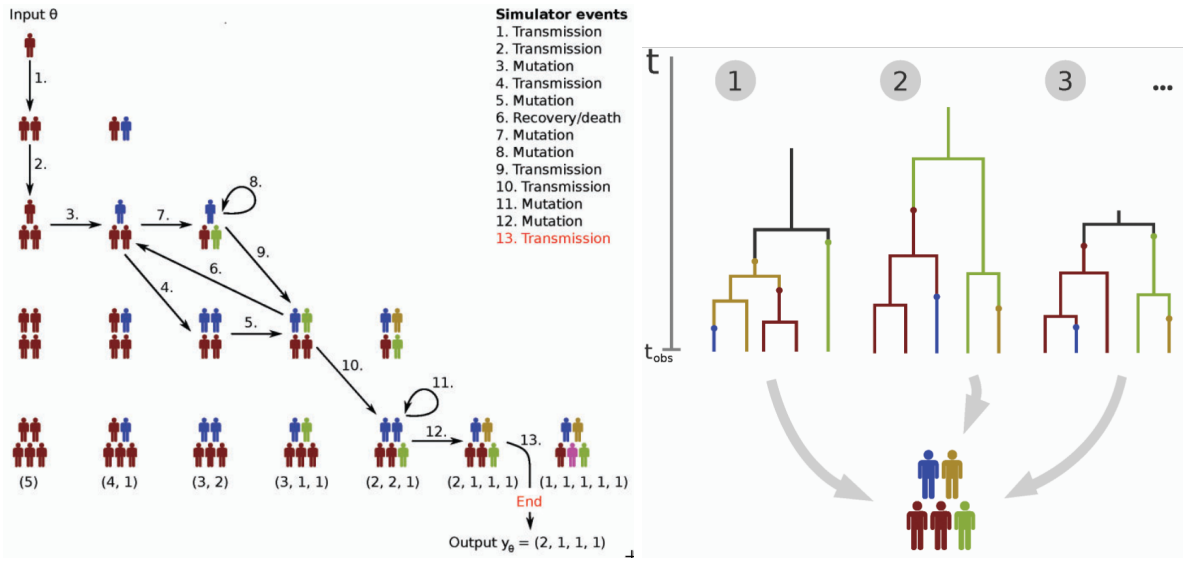

Figure 1: Image taken from [5]

Goal of Simulation-Based Models

As in all Machine Learning (ML) conceptes, the fundamental goal is the derivation of the parameter configuration(s) θ^* that *describe* well the data i.e. generate samples $M_r(\theta^*)$ that are as close as possible to the observed data y_0. Since Simulation-Based models belong to the broad category of Bayesian Machine Learning, the ultimate goal is to *infer* a posterior distribution $p(\theta|y_0)$ over of all possible configuration set-uPs and obtain some samples from this distribution $\theta \sim p(\theta|y_0)$. Doing so, we have uncovered the mechanism that produces the output, based on passed captured realisations of the phenomenon, and so we are able to achieve a wide range of tasks, such as predicting future outcomes or understanding the internals of the method.

Robust Optimisation Monte Carlo (ROMC) method

The ROMC method [3] is very a recent Likelihood-Free approach; its fundamental idea is the transformation of the stochastic data generation process $M_r(\theta)$ to a deterministic mapping $g(\theta)$, by pre-sampling the variables that produce the randomness $v_i \sim p(V)$. Formally, in every stochastic process the randomness is influenced by a vector of random variables v, whose state is unknown before the execution of the simulation; pre-sampling this state makes the procedure deterministic, namely $g_i(\theta) = M_d(\theta, V = v_i)$. This approach initially introduced by Meeds et. al [6] with the title Optimisation Monte Carlo (OMC). The ROMC extended this approach by improving a fundamental failure-mode of OMC. The ROMC describes a methodology for approximating the posterior through a series of steps, without explicitly enforcing which algorithms must be used for each step[1]; in this sense it can be thought as a meta-algorithm.

Implementation

The most important contribution of this work is the implementation of the ROMC method in the Python package Engine for Likelihood-Free Inference (ELFI) [4]. Since it is very recently published work the ROMC method was not implemented by now in any ML software. This works attempts to provide to the research community a tested and robust implementation for further experimentation.

1.2 Outline of Thesis

The remainder of the dissertation is organized as follows. In Chapter 2 we establish the mathematical formulation; more specifically we initially describe the Simulator-Based models and we provide some fundamental algorithms that have been proposed for performing statistical inference. Afterwards, we provide the mathematical description of the ROMC approach [3]. In Chapter 3, we deal with the implementation part; we initially provide some information regarding the Python package Engine for Likelihood-Free Inference (ELFI) [4] and subsequently we analyze the implementation details of ROMC in this package. In Chapter 4, we present the functionalities of the ROMC implementation at some real-world examples; this chapter wishes to illustrate the success of the ROMC method and of our implementation at Likelihood-Free tasks. Finally, in chapter 5, we conclude with some thoughts on the work we have done and some future research ideas.

1.3 Notation

In this section, we present an overview of the symbols that will be used in the rest of the document. At this level, the quantities are introduced quite informally, through general descriptions. Most of them will be defined formally in the next chapters. We try to keep the notation as consistent as possible throughout the document. The symbol $\mathbb{R}^N$, when used, describes that a variable belongs to a multi-dimensional space in $\mathbb{R}$; N doesn't represent a specific number.

[1]The implementation chooses a specific algorithm for each step, but the choice has just demonstrative value; any other appropriate algorithm can be used instead.

Random Generator

- $M_r(\boldsymbol{\theta}) : \mathbb{R}^D \to \mathbb{R}$: The black-box data simulator.

Parameters/Random Variables/Symbols

- $D \in \mathbb{R}$, the dimensionality of the parameter-space
- $\boldsymbol{\Theta} \in \mathbb{R}^D$, random variable representing the parameters of interest
- $\mathbf{y_0} \in \mathbb{R}^N$, the vector of observations
- $\epsilon \in R$, the threshold setting the limit on the region around $\mathbf{y_0}$
- $\mathbf{V} \in \mathbb{R}^N$, random variable representing the stochasticity of the generator. It is also called nuisance variable, because we are not interested in infering a postrior distribution on it.
- $\mathbf{v}_i \sim \mathbf{V}$, a specific sample drawn from $\mathbf{V}$
- $\mathbf{Y}_{\boldsymbol{\theta}}$, random variable descriping the simulator $M_r(\boldsymbol{\theta})$.
- $\mathbf{y}_i \sim \mathbf{Y}_\theta$, a sample drawn from $\mathbf{Y}_\theta$. It can be obtained by executing the simulator $\mathbf{y}_i \sim M_r(\boldsymbol{\theta})$

Sets

- $B_{d,\epsilon}(\mathbf{y_0})$, the set of points $\mathbf{y} := \{\mathbf{y} : d(\mathbf{y}, \mathbf{y_0}) < \epsilon\}$
- $B^i_{d,\epsilon}$, the set of points defined around $\mathbf{y}_i$ i.e. $B^i_{d,\epsilon} = B_{d,\epsilon}(\mathbf{y}_i)$

Generic Functions

- $p(\cdot)$, any valid pdf
- $p(\cdot|\cdot)$, any valid conditional distribution.
- $p(\boldsymbol{\theta})$, the prior distribution on the parameters
- $p(\mathbf{v})$, the prior distribution on the nuisance variables
- $p(\boldsymbol{\theta}|\mathbf{y_0})$, the posterior distribution
- $p_{d,\epsilon}(\boldsymbol{\theta}|\mathbf{y_0})$, the approximate posterior distribution
- $d(\mathbf{x}, \mathbf{y}) : \mathbb{R}^{2N} \to \mathbb{R}$: any valid distance e.g L_2 norm: $||\mathbf{x} - \mathbf{y}||_2^2$

Functions (Mappings)

- $M_d(\boldsymbol{\theta}, \mathbf{v}) : \mathbb{R}^D \to \mathbb{R}$, the deterministic generator; all stochastic variables that are part of the data generation process are represented by the **parameter v**
- $f_i(\boldsymbol{\theta}) = M_d(\boldsymbol{\theta}, \mathbf{v}_i)$, deterministic generator associated with sample $\mathbf{v}_i \sim p(\mathbf{v})$
- $g_i(\boldsymbol{\theta}) = d(f_i(\boldsymbol{\theta}), \mathbf{y_0})$, distance of the generated data $f_i(\boldsymbol{\theta})$ from the observations
- $T(\mathbf{x}) : \mathbb{R}^{D_1} \to \mathbb{R}^{D_2}$ where $D_1 > D_2$, the mapping that computes the summary statistic
- $\mathbb{1}_{B_{d,\epsilon}(\mathbf{y_0})}(\mathbf{y})$, the indicator function; returns 1 if $d(\mathbf{y}, \mathbf{y_0}) < \epsilon$, else 0
- $L(\boldsymbol{\theta})$, the likelihood
- $L_{d,\epsilon}(\boldsymbol{\theta})$, the approximate likelihood

2 Background

2.1 Simulator-Based (Implicit) Models

As already stated at chapter 1, in Simulator-Based models we cannot evaluate the posterior $p(\boldsymbol{\theta}|\mathbf{y_0}) \propto L(\boldsymbol{\theta})p(\boldsymbol{\theta})$, due to the intractability of the likelihood $L(\boldsymbol{\theta}) = p(\mathbf{y_0}|\boldsymbol{\theta})$. The following equation allows incorporating the simulator in the place of the likelihood and forms the basis of all Likelihood-Free inference approaches.

$$L(\boldsymbol{\theta}) = \lim_{\epsilon \to 0} c_\epsilon \int_{\mathbf{y} \in B_\epsilon(\mathbf{y_0})} p(\mathbf{y}|\boldsymbol{\theta}) d\mathbf{y} = \lim_{\epsilon \to 0} c_\epsilon Pr(M_r(\boldsymbol{\theta}) \in B_\epsilon(\mathbf{y_0})) \tag{2.1}$$

2.1.1 Approximate Bayesian Computation (ABC) Rejection Sampling

ABC Rejection Sampling is a modified version of the traditional Rejection Sampling method, for cases when likelihood evaluation is intractable. In typical Rejection Sampling, a sample is obtained from the prior $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$ and it is maintained with probability $L(\boldsymbol{\theta})/\max_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$. Although we cannot use this approach out of the box (evaluating $L(\boldsymbol{\theta})$ is impossible in our case), we can take modify the approach for using the simulator.

In the discrete case scenario where $\mathbf{Y_\theta}$ can take a finite set of values, the likelihood becomes $L(\boldsymbol{\theta}) = Pr(\mathbf{Y_\theta} = \mathbf{y_0})$ and the posterior $p(\boldsymbol{\theta}|\mathbf{y_0}) \propto Pr(\mathbf{Y_\theta} = \mathbf{y_0})p(\boldsymbol{\theta})$. We can sample from the prior $\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta})$, run the simulator $\mathbf{y}_i = M_r(\boldsymbol{\theta}_i)$ and maintain $\boldsymbol{\theta}_i$ only if $\mathbf{y}_i = \mathbf{y_0}$.

The method above becomes less helpfull as the finite set of $\mathbf{Y_\theta}$ values grows larger, since the probability of maintaining a sample (acceptance rate) becomes smaller. In the limit where the set becomes infinite (i.e. continuous case) the probability becomes zero. In order for the method to work in this set-up, a relaxation is introduced; we relax the acceptance criterion by letting $\mathbf{y}_i$ lie in a larger set of points i.e. $\mathbf{y}_i \in B_{d,\epsilon}(\mathbf{y_0}), \epsilon > 0$. The region can be defined as $B_{d,\epsilon}(\mathbf{y_0}) := \{\mathbf{y} : d(\mathbf{y}, \mathbf{y_0}) < \epsilon\}$ where $d(\cdot, \cdot)$ can represent any valid distance. With this modification, the maintained samples follow an approximate posterior,

$$p_{d,\epsilon}(\boldsymbol{\theta}|\mathbf{y_0}) \propto Pr(\mathbf{y} \in B_{d,\epsilon}(\mathbf{y_0}))p(\boldsymbol{\theta}) \tag{2.2}$$

This method is called *Rejection ABC*.

2.1.2 Summary Statistics

When the dimensionality of $\mathbf{y} \in \mathbb{R}^D$ is high, generating samples inside $B_{d,\epsilon}(\mathbf{y_0})$ becomes rare even with large acceptance region; this is the curse of dimensionality. As a representative example if (a) d is set to be the euclidean distance, then $B_{d,\epsilon}(\mathbf{y_0}) := \{\mathbf{y} : ||\mathbf{y} - \mathbf{y_0}||_2^2 < \epsilon^2\}$ is a hyper-sphere with radius ϵ and (b) if the prior $p(\boldsymbol{\theta})$ is a uniform distribution in a hyper-cube with side of length 2ϵ, then the probability of drawing a sample inside the hyper-sphere becomes:

$$Pr(\mathbf{y} \in B_{d,\epsilon}(\mathbf{y_0})) = Pr(\boldsymbol{\theta} \in B_{d,\epsilon}(\mathbf{y_0})) = \frac{V_{hypersphere}}{V_{hypercube}} = \frac{\pi^{D/2}}{D2^{D-1}\Gamma(D/2)} \to 0, \quad \text{as} \quad D \to \infty \tag{2.3}$$

We observe that the probality tends to 0, independently of ϵ; enlarging ϵ will not increase the acceptance rate. This produces the need for a mapping $T : \mathbb{R}^{D_1} \to \mathbb{R}^{D_2}$ where $D_1 > D_2$, for squeezing the dimensionality of the output. This intermediate step redefines the area as $B_{d,\epsilon}(\mathbf{y_0}) := \{\mathbf{y} : d(T(\mathbf{y}), T(\mathbf{y_0})) < \epsilon\}$. This dimensionality-reduction step is called *summary statistic* extraction, since the distance is not measured on the actual outputs, but on a summarization (i.e. lower-dimension representation) of them.

2.1.3 Approximations Introduced

So far, we have introduced some approximations for infering the posterior as $p_{d,\epsilon}(\boldsymbol{\theta}|\mathbf{y_0}) \propto Pr(\mathbf{Y_\theta} \in B_{d,\epsilon}(\mathbf{y_0}))p(\boldsymbol{\theta})$ where $B_{d,\epsilon}(\mathbf{y_0}) := \{\mathbf{y} : d(T(\mathbf{y}), T(\mathbf{y_0})) < \epsilon\}$. These approximations introduce two

different types of errors:

- ϵ is chosen to be *big enough*, so that enough samples are accepted
- T introduces loss of information, making possible a $\mathbf{y}$ far away from the $\mathbf{y_0}$ i.e. $\mathbf{y} : d(\mathbf{y}, \mathbf{y_0}) > \epsilon$, to enter the acceptance region after the dimensionality reduction $d(T(\mathbf{y}), T(\mathbf{y_0}) < \epsilon)$

In the following sections we will not use the summary statistics in our expressions, for the notation not to clutter. Though, all the following propositions are valid with the use of summary statistics.

2.1.4 Optimization Monte Carlo (OMC)

Based on $B_{d,\epsilon}$, we can define two useful entities; an indicator function and a conditional distribution.

Indicator Function

The indicator function $\mathbb{1}_{B_{d,\epsilon}(\mathbf{y})}(\mathbf{x})$ returns 1 if $\mathbf{x} \in B_{d,\epsilon}(\mathbf{y})$ and 0 otherwise. If $d(\cdot, \cdot)$ is a formal distance, due to symmetry $\mathbb{1}_{B_{d,\epsilon}(\mathbf{y})}(\mathbf{x}) = \mathbb{1}_{B_{d,\epsilon}(\mathbf{x})}(\mathbf{y})$.

$$\mathbb{1}_{B_{d,\epsilon}(\mathbf{y})}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in B_{d,\epsilon}(\mathbf{y}) \\ 0 & \text{else} \end{cases} \tag{2.4}$$

Boxcar Kernel

The boxcar kernel is the conditional distribution:

$$p_{d,\epsilon}(\mathbf{y}|\mathbf{x}) = \begin{cases} c & \text{if } d(\mathbf{y}, \mathbf{x}) \leq \epsilon \\ 0 & \text{else} \end{cases} \quad \text{where } c = \frac{1}{\int_{\{\mathbf{y}: d(\mathbf{y},\mathbf{x}) < \epsilon\}} d\mathbf{y}} \tag{2.5}$$

Observing the boxcar kernel from the view-point of a data generation process, where the point $\mathbf{y}$ is generated after $\mathbf{x}$, we can make two important notices:

- given a specific $\mathbf{x}$, all values $\mathbf{y} : \mathbf{y} \in B_{d,\epsilon}(\mathbf{x})$ have equal probability to be generated
- if a specific $\mathbf{y}$ value has been generated, all $\mathbf{x} : \mathbf{x} \in B_{d,\epsilon}(\mathbf{y})$ are equally probable to have been the initial point that provoked $\mathbf{y}$ generation

Finally, we can also observe that the kernel can be defined through the indicator function:

$$p_{d,\epsilon}(\mathbf{y}|\mathbf{x}) = c\mathbb{1}_{B_{d,\epsilon}(\mathbf{y})}(\mathbf{x}) = c\mathbb{1}_{B_{d,\epsilon}(\mathbf{x})}(\mathbf{y}) \tag{2.6}$$

Initial View

Based on equation 2.2 and the Boxcar kernel 2.5, we can approximate the likelihood as:

$$L_{d,\epsilon}(\boldsymbol{\theta}) = \int_{\mathbf{y} \in B_\epsilon(\mathbf{y_0})} p(\mathbf{y}|\boldsymbol{\theta}) d\mathbf{y} = \int_{\mathbf{y} \in \mathbb{R}^D} \mathbb{1}_{B_{d,\epsilon}(\mathbf{y_0})}(\mathbf{y}) p(\mathbf{y}|\boldsymbol{\theta}) d\mathbf{y} \tag{2.7}$$

$$\approx \frac{1}{N} \sum_i^N \mathbb{1}_{B_{d,\epsilon}(\mathbf{y_0})}(\mathbf{y}_i), \text{ where } \mathbf{y}_i \sim M_r(\boldsymbol{\theta}) \tag{2.8}$$

This approach is quite intuitive; approximating the likelihood of a specific $\boldsymbol{\theta}$ requires sampling from the data generator and count the fraction of samples that lie inside the area around the observations. Nevertheless, for each distinct evaluation of $L_{d,\epsilon}(\boldsymbol{\theta})$, N new samples are needed to be sampled; this make this approach quite inconvenient from a computational point-of-view.

Alternative View

For overcoming the disadvantage introduced above, OMC attempts an alternative approximation. It samples all the nuisance variables once $\mathbf{v}_i \sim p(\mathbf{v})$ and it converts the random simulator to a deterministic mapping $M_d(\boldsymbol{\theta}, \mathbf{v}_i)$,

$$L_{d,\epsilon}(\boldsymbol{\theta}) = \int_{\mathbf{y}\in B_\epsilon(\mathbf{y_0})} p(\mathbf{y}|\boldsymbol{\theta})d\mathbf{y} = \int_{\mathbf{y}\in\mathbb{R}^D} \mathbb{1}_{B_{d,\epsilon}(\mathbf{y_0})}(\mathbf{y})p(\mathbf{y}|\boldsymbol{\theta})d\mathbf{y} \tag{2.9}$$

$$= \int_{\mathbf{y}}\int_{\mathbf{v}} \mathbb{1}_{B_{d,\epsilon}(\mathbf{y_0})}(\mathbf{y})p(\mathbf{y}|\boldsymbol{\theta},\mathbf{v})p(\mathbf{v})d\mathbf{y}d\mathbf{v} \tag{2.10}$$

$$= \int_{\mathbf{v}} \mathbb{1}_{B_{d,\epsilon}(\mathbf{y_0})}(\mathbf{y} = M_d(\boldsymbol{\theta},\mathbf{v}))p(\mathbf{v})d\mathbf{v} \tag{2.11}$$

$$\approx \frac{1}{N}\sum_i^N \mathbb{1}_{B_{d,\epsilon}(\mathbf{y_0})}(\mathbf{y}_i) \tag{2.12}$$

$$\text{where } \mathbf{y}_i = M_d(\boldsymbol{\theta},\mathbf{v}_i), \mathbf{v}_i \sim p(\mathbf{v}) \tag{2.13}$$

Based on this approach, the unnormalized approximate posterior can be defined as:

$$p_{d,\epsilon}(\boldsymbol{\theta}|\mathbf{y_0}) \propto p(\boldsymbol{\theta})\sum_i^N \mathbb{1}_{B_{d,\epsilon}(\mathbf{y_0})}(\mathbf{y}_i) \tag{2.14}$$

Forming an analogy with the previous approach, we sample many nuisance variables in order to absorb the randomness of the generator and we count the fraction of times the deterministic generator produce outputs close to the observed data. Though it is conceptually similar to the previous approach, we now overcome the previous disadvantage; we sample the nuisance variables once (training part) and afterwards evaluate every θ based on the predefined expression (inference part).

2.2 Robust Optimistation Monte Carlo (ROMC) approach

Apart from defining a tractable aproximation of the posterior, ROMC provides a method for sampling from the posterior and computing an expectation.

Weighted Sampling

Sampling could be performed in a straightforward fashion using importance sampling; using the prior as the proposal distribution $\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta})$ and computing the weight as $w_i = \frac{L_{d,\epsilon}(\boldsymbol{\theta}_i)}{p(\boldsymbol{\theta}_i)}$. This approach has the same drawbacks as ABC rejection sampling; when the prior is wide or the dimensionality D high, drawing a sample with non-zero weight is rare, leading to either poor Effective Sample Size (ESS) or huge execution time. The ROMC method proposes a better sampling approach by constructing an appropriate proposal distribution q_i per nuisance variable $\mathbf{v}_i$[2]. Therefore the weight is computed as,

$$w_{ij} = \frac{L_{d,\epsilon}(\boldsymbol{\theta}_{ij})p(\boldsymbol{\theta}_{ij})}{q(\boldsymbol{\theta}_{ij})}, \boldsymbol{\theta}_{ij} \sim q_i(\boldsymbol{\theta}) \tag{2.15}$$

Computing an expectation

Having defined the procedure for obtaining weighted samples, any expectation $E_{p(\boldsymbol{\theta}|\mathbf{y_0})}[h(\boldsymbol{\theta})]$, can be approximated as,

$$E_{p(\boldsymbol{\theta}|\mathbf{y_0})}[h(\boldsymbol{\theta})] \approx \frac{\sum_{ij} w_{ij}h(\boldsymbol{\theta}_{ij})}{\sum_{ij} w_{ij}} \tag{2.16}$$

[2]We describe the proposal area construction in the next chapter

2.2.1 Define and solve deterministic optimisation problems

For each set of nuisance variables $\mathbf{v}_i, i = \{1, 2, ..., n_1\}$ a deterministic function is defined as $f_i(\boldsymbol{\theta}) = M_d(\boldsymbol{\theta}, \mathbf{v}_i)$. For constructiong the proposal region, we search for a point $\boldsymbol{\theta}_* : d(f_i(\boldsymbol{\theta}_*), \mathbf{y_0}) < \epsilon$; this point can be obtained by solving the the following optimisation problem:

$$\min_{\boldsymbol{\theta}} \quad g_i(\boldsymbol{\theta}) = d(\mathbf{y_0}, f_i(\boldsymbol{\theta})) \tag{2.17a}$$

$$\text{subject to} \quad g_i(\boldsymbol{\theta}) < \epsilon \tag{2.17b}$$

We maintain a list of the solutions $\boldsymbol{\theta}_i^*$ of the optimisation problems. If for a specific set of nuisance variables $\mathbf{v}_i$, there is no feasible solution we add nothing to the list. The optimisation problem 2.17a can be treated as unconstrained, accepting the optimal point $\boldsymbol{\theta}_i^* = \text{argmin}_{\boldsymbol{\theta}} g_i(\boldsymbol{\theta})$ only if $g_i(\boldsymbol{\theta}_i^*) < \epsilon$.

2.2.2 Gradient-Based Approach

The nature of the generative model $M_r(\theta)$, specifies the properties of the objective function g_i. If g_i is continuous with smooth gradients $\nabla_\theta g_i$ any gradient-based iterative algorithm can be used for solving 2.17a. The gradients $\nabla_\theta g_i$ can be either provided in closed form or approximated by finite differences.

2.2.3 Gaussian Process Approach

In cases where the gradients are not available, the Bayesian Optimisation scheme provides an alternative choice. With this approach, apart from ontaining an optimal $\boldsymbol{\theta}_i^*$, a surrogate model $\hat{d}_i$ of the distance g_i is fitted; this approximate model can be used in the following steps, instead of g_i, providing a noteworthy speed-up. Specifically, in the construction of the proposal region and in equations 2.2, 2.15, 2.16 it could replace g_i in the evaluation of the indicator function 2.4, providing a major speed-up.

2.2.4 Construction of the proposal area q_i

Independently of the approach chosen above, the constraction of the proposal region follows a common method. The search directions $\mathbf{v}_d$ are computed as the eigenvectors of the curvature at θ_i^* and a line-search method is used to obtain the limit point where $g_i(\boldsymbol{\theta}_i^* + \kappa \mathbf{v}_d) \geq \epsilon$. Algorithm 4 describes analytically the method.

2.3 Algorithmic Description of ROMC

In this section, we attempt the depiction of the mathematical description of ROMC in algorithms. Specificaly, in chapter 2.3.1 we present the general algorithmic description of ROMC as a meta-algorithm and in chapter 2.3.2 the proposals of ROMC for solving the training and the inference parts.

2.3.1 ROMC as a Meta-Algorithm

As stated at the introductory chapter, ROMC can be understood as step-by-step alorithmic approach for perfroming the inference in Simulator-Based Models. The particular methods used for solving the sub-tasks are left as a free choice to the user. As presented in Algorithm 1, the methods involved in solving the optimistation problem (step 4) and constructing the bounding box (step 5) are not restricted. The practiosioner may choose any convenient algorithm, judging the trade-offs between accuracy, robustness, efficiency and complexity. In particular for the optimisation step, the choice of the appropriate optimiser should also consider the properties of $g_i(\boldsymbol{\theta})$. Some important questions that should be considered are whether the function differentiable and if so whether we know the gradients

Algorithm 1 ROMC as a Meta-Algorithm. Requires $M_r(\theta), y_0$. Hyperparameters n_1, n_2.

for $i \leftarrow 1$ to n_1 **do**
  Sample a random state $\mathbf{v}_i \sim p(\mathbf{v})$
  Define the deterministic mapping $f_i(\boldsymbol{\theta}) = M_d(\boldsymbol{\theta}, \mathbf{v})$ and therefore $g_i(\boldsymbol{\theta}) = d(f_i(\theta), y_0)$.
  Obtain $d_i^* = \min_{\boldsymbol{\theta}} [g_i(\boldsymbol{\theta})]$ and $\boldsymbol{\theta}_i^* = \text{argmin}_{\boldsymbol{\theta}} [g_i(\boldsymbol{\theta})]$ using any convenient optimiser.
  Approximate the local area $\{\boldsymbol{\theta} : g_i(\boldsymbol{\theta}) < \epsilon \text{ and } d(\boldsymbol{\theta}, \boldsymbol{\theta}_i^*) < M\}$ with a Bounding Box, using any convenient method.
  Define a uniform distribution $q_i(\boldsymbol{\theta})$ over the Bounding Box.
  for $j \leftarrow 1$ to n_2 **do**
    $\boldsymbol{\theta}_{ij} \sim q_i(\boldsymbol{\theta})$
    Accept $\boldsymbol{\theta}_{ij}$ as posterior sample with weight $w_{ij} = \frac{p(\boldsymbol{\theta}_{ij})}{q_i(\boldsymbol{\theta}_{ij})} \mathbb{1}_{B_{d,\epsilon}^i}(\boldsymbol{\theta}_{ij})$
return (List with samples $\boldsymbol{\theta}_{ij}$ and weights w_{ij})

$\nabla_{\boldsymbol{\theta}}[g_i]$ in closed-form. As described in sections 2.2.2 and 2.2.3, ROMC proposes two alternative optimisation schemes (gradient-based and gaussian-process approach) depending on whether the gradients are available or not.

2.3.2 Training and Inference Algorithms

In this section, we will provide the algorithmic description of the ROMC method; (a) the procedures for solving the optimisation problems using either the gradient based approach or the Gaussian Process alternative and (b) the construction of the Bounding Box. Afterwards, we will discuss the advantages and the disadvantages of each choice both in terms of accuracy and efficiency.

At a high-level, the ROMC method can be split into the training and the inference part.

At the training (fitting) part, the goal is the estimation of the proposal regions q_i. The steps include (a) sampling the nuisance variables $\mathbf{v}_i \sim p(\mathbf{v})$ (b) defining the optimisation problems $\min_{\boldsymbol{\theta}}[g_i(\boldsymbol{\theta})]$ (c) obtaining $\boldsymbol{\theta}_i^*$ (d) checking whether $d_i^* < \epsilon$ and (e) building the bounding box for obtaining the proposal region q_i. If gradients are available, using a gradient-based method is adviced for obtaining $\boldsymbol{\theta}_i^*$ much faster. Providing $\nabla_{\boldsymbol{\theta}} g_i$ in closed-form provides an upgrade in both accuracy and efficiency; If closed-form description is not available, approximate gradients with finite-differences $\frac{\partial g_i(\boldsymbol{\theta})}{\boldsymbol{\theta}_d} = \frac{g_i(\boldsymbol{\theta}_d + h\mathbf{e_d}) - g_i(\boldsymbol{\theta}_d)}{h}$ requires two evaluations of g_i for **every** parameter $\boldsymbol{\theta}_d$. For low-dimensional problems though, this approach still works well. When gradients are not available or g_i is not differentiable, using the Gaussian Process is the only solution. In this case, the training part is much slower due to the fitting of the surrogate model and the ignorance of the slope throughout the optimisation procedure. Nevertheless, computing the proposal region q_i becomes faster since $\hat{d}_i$ can be used instead of g_i which involves running the whole simulator $M_d(\boldsymbol{\theta}, \mathbf{v}_i)$ for each query. The algorithms are presented in 2 and 3.

Performing the inference includes (a) evaluating the unnormalised posterior $p_{d,\epsilon}(\theta_b|\mathbf{y_0})$ (b) sampling from the posterior $\boldsymbol{\theta}_i \sim p_{d,\epsilon}(\theta_b|\mathbf{y_0})$ and (c) computing an expectation $E_{\boldsymbol{\theta}|\mathbf{y_0}}[h(\boldsymbol{\theta})]$. Computing an expectation can be done easily after weighted samples are obtained 2.16, so we will not discuss it seperately.

For evaluating the unnormalized posterior in the gradient-based approach, only the deterministic functions g_i and the prior distribution $p(\boldsymbol{\theta})$ are required; there is no need for solving the optimisation problems and building the proposal regions. The evaluation requires iterating over all g_i and evaluating the distance from the observed data. In contrast, using the GP approach, the optimisation part should be performed first for fitting the surrogate models $\hat{d}_i(\boldsymbol{\theta})$ and evaluate the indicator function on them. This provides an important speed-up, especially when running the simulator is computationally expensive. The evaluation of the posterior is presented analytically in 5 and 6.

Sampling is performed by getting n_2 samples from each proposal region q_i. For each sample $\boldsymbol{\theta}_{ij}$, the indicator function is evaluated $\mathbb{1}_{B_{d,\epsilon}^i(\mathbf{y_0})}(\boldsymbol{\theta}_{ij})$ for checking if it lies inside the acceptance region. If so the corresponding weight is computed as in [**eq:sampling**]. As before, if a surrogate model $\hat{d}$ has been fitted, it can be used for the evaluation of the indicator function providing again a speedup.

Apparently, the compuational benefit is more important compared to posterior evaluation, because the indicator function must be evaluated for a total of $n_1 \times n_2$ points. The sampling algorithms are presented step-by-step in 7 and 8.

As a conclusion, we can state that the choise of using a bayesian optimisation approach provides a significant speed-up in the inference part with the cost of making the training part slower and a possible approximation error. It is typical in many Machine-Learning use cases, being able to provide enough time and computational resources for the training phase, but asking for efficiency in the inference part. Having that in mind, we can say that the Gaussian-Process is a quite usefull alternative.

Algorithm 2 Training Part - Gradient approach. Requires $g_i(\theta), p(\theta)$

for $i \leftarrow 1$ to n **do**
  Obtain θ_i^* using a Gradient Optimiser
  if $g_i(\theta_i^*) > \epsilon$ **then**
    go to 1
  else
    Approximate $H_i \approx J_i^T J_i$
    Use algorihm 4 to obtain q_i
return $q_i, p(\theta), g_i(\theta)$

Algorithm 3 Training Part - GP approach. Requires $g_i(\theta), p(\theta)$

for $i \leftarrow 1$ to n **do**
  Obtain $\theta_i^*, \hat{d}_i(\theta)$ using a GP approach
  if $g_i(\theta_i^*) > \epsilon$ **then**
    go to 1
  else
    Approximate $H_i \approx J_i^T J_i$
    Use algorihm 4 to obtain q_i
return $q_i, p(\theta), \hat{d}_i(\theta)$

Algorithm 4 Proposal Region q_i construction; Needs, a model of distance d ($\hat{d}$ or g_i), optimal point θ_i^*, number of refinements K, step size η and curvature matrix $\mathbf{H}_i$ ($J_i^T J_i$ or GP Hessian)

Compute eigenvectors $\mathbf{v}_d$ of H_i $(d = 1, \ldots, ||\theta||)$
for $d \leftarrow 1$ to $||\theta||$ **do**
  $\tilde{\theta} \leftarrow \theta_i^*$
  $k \leftarrow 0$
  repeat
    repeat
      $\tilde{\theta} \leftarrow \tilde{\theta} + \eta\ \mathbf{v}_d$ ▷ Large step size η.
    until $d((\tilde{\theta}, i),) \geq \epsilon$
    $\tilde{\theta} \leftarrow \tilde{\theta} - \eta\ \mathbf{v}_d$
    $\eta \leftarrow \eta/2$ ▷ More accurate region boundary
    $k \leftarrow k + 1$
  until $k = K$
  Set final $\tilde{\theta}$ as region end point.
  Repeat steps 3 - 13 for $\mathbf{v}_d = -\mathbf{v}_d$
Fit a rectangular box around the region end points and define q_i as uniform distribution

Algorithm 5 Evaluate unnormalised posterior - Gradient approach. Requires $g_i(\theta), p(\theta)$

$k \leftarrow 0$
for $i \leftarrow 1$ to n_1 **do**
  if $g_i(\theta) > \epsilon$ **then**
    $k \leftarrow k + 1$
return $kp(\theta)$

Algorithm 6 Evaluate unnormalised posterior - GP approach. Requires $\hat{d}_i(\theta), p(\theta)$

$k \leftarrow 0$
for $i \leftarrow 1$ to n_1 **do**
  if $d_i(\theta) > \epsilon$ **then**
    $k \leftarrow k + 1$
return $kp(\theta)$

Algorithm 7 Sampling - Gradient Based approach. Requires $g_i(\theta), p(\theta), q_i$

for $i \leftarrow 1$ to n_1 **do**
  for $j \leftarrow 1$ to n_2 **do**
    $\theta_{ij} \sim q_i$
    if $g_i(\theta_{ij}) > \epsilon$ **then**
      Reject θ_{ij}
    else
      $w_{ij} = \frac{p(\theta_{ij})}{q(\theta_{ij})}$
      Accept θ_{ij}, with weight w_{ij}

Algorithm 8 Sampling - GP approach. Requires $\hat{d}_i(\theta), p(\theta), q_i$

for $i \leftarrow 1$ to n_1 **do**
  for $j \leftarrow 1$ to n_2 **do**
    $\theta_{ij} \sim q_i$
    if $\hat{d}_i(\theta_{ij}) > \epsilon$ **then**
      Reject θ_{ij}
    else
      $w_{ij} = \frac{p(\theta_{ij})}{q(\theta_{ij})}$
      Accept θ_{ij}, with weight w_{ij}

2.4 Engine for Likelihood-Free Inference (ELFI) Package

!! NOT fully written, I have to add some more info !!

The Engine for Likelihood-Free Inference (ELFI) [4] is a Python software library dedicated to likelihood-free inference (LFI). ELFI models in a convenient manner all the fundamental components of a Probabilistic Model such as priors, simulators, summaries and distances. Furthermore, in the ELFI there are implemented a wide range of likelihood-free inference methods.

2.4.1 Modelling

ELFI models the Probabilistic Model as Directed Acyclic Graph (DAG); it implements this fucntionality based on the package NetworkX, which is designed for creating general purpose graphs. Although not restricted to that, in most cases the structure of a likelihood-free model follows the pattern presented in figure 2; there are edges that connect the *prior* distributions to the simulator, the simulator is connected to the summary statistics which are connected to the distance, which is the output node. Samples can be obtained from all nodes through sequential sampling. The nodes that are defined as *elfi.Prior*[3] are automatically considered as the parameters of interest and they are the only nodes that should provide pdf evaluation, apart from sampling. The function passed as argument in the *elfi.Summary* node can be any valid Python function with arguments the prior variabless.

```
# Define the simulator, the summary and the observed data
def simulator(t1, t2, batch_size=1, random_state=None):
    # Implementation comes here. Return 'batch_size'
    # simulations wrapped to a NumPy array.
def summary(data, argument=0):
    # Implementation comes here...
y = # Observed data, as one element of a batch.

# Specify the ELFI graph
t1 = elfi.Prior('uniform', -2, 4)
t2 = elfi.Prior('normal', t1, 5)  # depends on t1
SIM = elfi.Simulator(simulator, t1, t2, observed=y)
S1 = elfi.Summary(summary, SIM)
S2 = elfi.Summary(summary, SIM, 2)
d = elfi.Distance('euclidean', S1, S2)

# Run the rejection sampler
rej = elfi.Rejection(d, batch_size=10000)
result = rej.sample(1000, threshold=0.1)
```

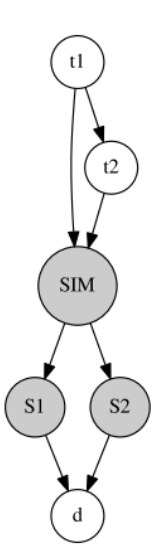

Figure 2: Image taken from [4]

[3]The *elfi.Prior* functionality is a wrapper around the scipy.stats package.

2.4.2 Inference Methods

All Inference Methods that are implemented in ELFI, follow some common guidelines; (a) their initialization should by defined by passing the output graph as the initial argument and afterwards come the rest hyperparameters of the method and (b) they must provide a basic inference fucntionality, e.g. *<method>.sample()*, which returns a predefined *elfi.Result* object containing the obtained samples along with some other useful functionalitities (e.g. ploting the marginal posteriors).

A good collection of likelihood-free inference methods is implemented so far, such as the *ABC Rejection Sampler* and *Sequential Monte Carlo ABC Sampler*. A quite central method implemented by ELFI is the *Bayesian Optimization for Likelihood-Free Infernce (BOLFI)*, which is methodologically quite close to the ROMC method we implement in the current dissertation.

3 Implementation

3.1 General Design

3.2 Training

3.3 Performing the Infernce

3.4 Utilities

3.5 Computational Complexity

References

[1] Yanzhi Chen and Michael U Gutmann. "Adaptive Gaussian Copula ABC". In: *Proceedings of Machine Learning Research*. Vol. 89. 2019, pp. 1584–1592. URL: http://proceedings.mlr.press/v89/chen19d.html.

[2] Michael U. Gutmann and Jukka Corander. *Bayesian optimization for likelihood-free inference of simulator-based statistical models*. 2016. arXiv: 1501.03291.

[3] Borislav Ikonomov and Michael U. Gutmann. "Robust Optimisation Monte Carlo". In: (2019). arXiv: 1904.00670. URL: http://arxiv.org/abs/1904.00670.

[4] Jarno Lintusaari et al. *ELFI: Engine for Likelihood Free Inference*. 2018. eprint: arXiv:1708.00707.

[5] Jarno Lintusaari et al. "Fundamentals and recent developments in approximate Bayesian computation". In: *Systematic Biology* 66.1 (2017), e66–e82. ISSN: 1076836X. DOI: 10.1093/sysbio/syw077.

[6] Edward Meeds and Max Welling. "Optimization Monte Carlo: Efficient and embarrassingly parallel likelihood-free inference". In: *Advances in Neural Information Processing Systems*. 2015. arXiv: 1506.03693.

[7] Mark M. Tanaka et al. "Using approximate bayesian computation to estimate tuberculosis transmission parameters from genotype data". In: *Genetics* (2006). ISSN: 00166731. DOI: 10.1534/genetics.106.055574.

Appendices

A An Appendix

Some stuff.

B Another Appendix

Some other stuff.