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

This dissertation is mainly focused on the implementation of the Robust Optimisation Monte Carlo (ROMC) method as it was proposed by (**Ikonomov2019**) at the python package ELFI (Engine For Likelihood-Free Inference) (**1708.00707**). The ROMC method describes a novel likelihood-free inference approach for simulator-based models.

1.1 Motivation

Explanation of simulation-based models

A simulator-based model is a parameterised stochastic data generating mechanism (**Gutmann2016**). The key characteristic of these models is that although we can sample (simulate) data points, we cannot evaluate the likelihood for a specific set of observations $\mathbf{y_0}$. Formally, a simulator-based model is described as a parameterised family of probability density functions $\{p_{\mathbf{y}|\boldsymbol{\theta}}(\mathbf{y})\}_{\boldsymbol{\theta}}$, whose closed-form is either unknown or intractable to evaluate. Whereas evaluating $p_{\mathbf{y}|\boldsymbol{\theta}}(\mathbf{y})$ is intractable, sampling is feasible. Practically, a simulator can be understood as a black-box machine M_r[1] that given a set of parameters $\boldsymbol{\theta}$, produces samples $\mathbf{y}$ in a stochastic manner i.e. $M_r(\boldsymbol{\theta}) \rightarrow \mathbf{y}$.

Simulator-based models are particularly captivating due to the modelling freedom they provide; any physical process that can be conceptualised as a computer program of finite (deterministic or stochastic) steps can be modelled as a simulator-based model with any more mathematical compromise. This includes any amount of hidden (unobserved) internal variables or logic-based decisions. As always, this degree of freedom comes at a cost; performing the inference is particularly demanding from both the computational and the mathematical perspective. Unfortunately, the algorithms deployed so far, permit the inference only at low-dimensionality parametric spaces, i.e. $\boldsymbol{\theta} \in \mathbb{R}^D$ where D is small.

Example

For illustrating the importance of simulator-based models, let us use the tuberculosis disease spread example as described in (**Tanaka2006**). An overview of the disease spread model is presented at figure **??**. At each stage one of the following *unobserved* events may happen; (a) the transmission of a specific haplotype to a new host (b) the mutation of an existent haplotype (c) the exclusion of an infectious host (recovers/dies) from the population. The random process, which stops when m infectious hosts are reached[2], can be parameterised (a) by the transmission rate α (b) the mutation rate τ and (c) the exclusion rate δ, creating a $3D$-parametric space $\boldsymbol{\theta} = (\alpha, \tau, \delta)$. The outcome of the process is a variable-size tuple $\mathbf{y_\theta}$, containing the population contaminated by each different haplotype, as described in figure **??**. Lets say that the disease has been spread in a real population and when m hosts where contaminated simultaneously, the vector with the infectious populations has been measured to be $\mathbf{y_0}$. We would like to discover the parameters $\boldsymbol{\theta} = (\alpha, \tau, \delta)$ that describe the spreading process and lead to the specific outcome $\mathbf{y_0}$. Computing $p(\mathbf{y} = \mathbf{y_0}|\boldsymbol{\theta})$ requires tracking all tree-paths that generate the specific tuple along with their probabilities and summing over them. Computing this probability by enumerating each possible tree-path that may lead to the specific outcome becomes intractable when m grows larger, as in real-case scenarios. This can be easily observed in the tree presented at figure **??**. On the other hand, modelling the data-generation process as a computer program is simple and computationally efficient, hence using a simulator-based Model is a perfect fit.

Goal of Simulation-Based Models

As in most Machine Learning (ML) concepts, the fundamental goal is the derivation of one(many) parameter configuration(s) $\boldsymbol{\theta}^*$ that *describe* well the data i.e. generate samples $M_r(\boldsymbol{\theta}^*)$ that are as close as possible to the observed data $\mathbf{y_0}$. In our case, following the approach of Bayesian Machine

[1] The subscript r in M_r indicates the *random* simulator. In the next chapters we will introduce M_d witch stands for the *deterministic* simulator.

[2] We suppose that the unaffected population is infinite, so a new host can always be added until we reach m simultaneous hosts.

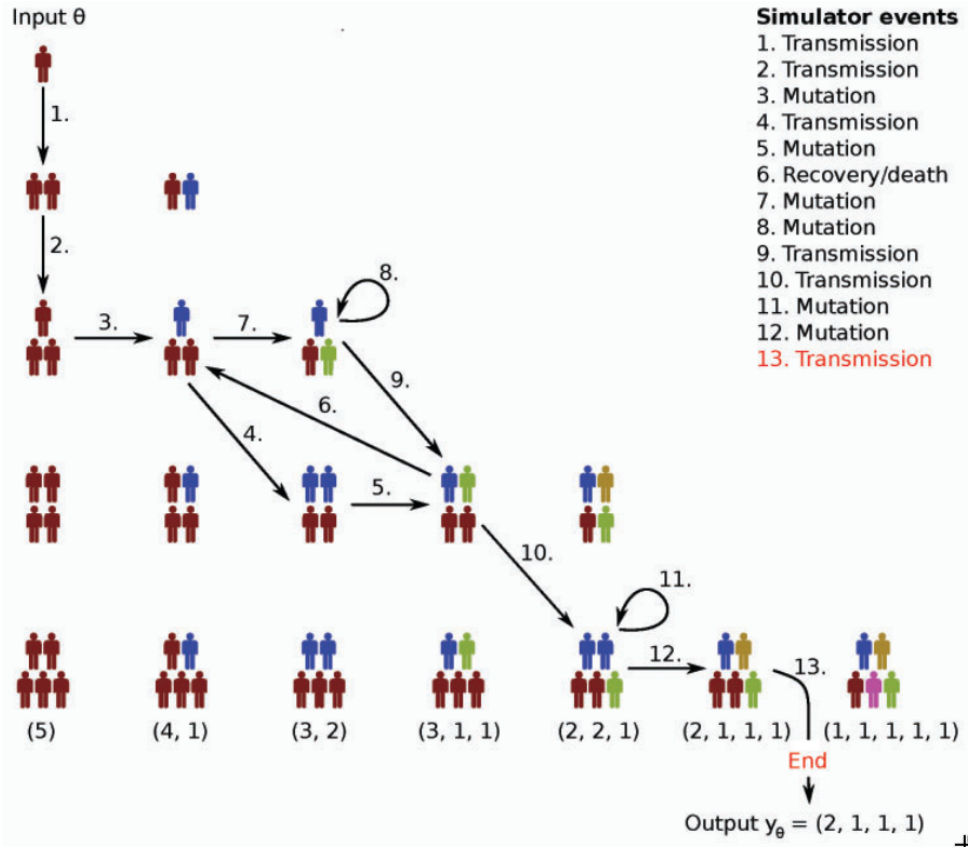

Figure 1: Depiction of a spread outcome of the tuberculosis spreading process. The image has been taken from (**Lintusaari2017**)

Learning, we treat the parameters of interest $\boldsymbol{\theta}$ as random variables and we try to *infer* a posterior distribution $p(\boldsymbol{\theta}|\mathbf{y_0})$ on them.

Robust Optimisation Monte Carlo (ROMC) method

The ROMC method (**Ikonomov2019**) is very a recent Likelihood-Free approach; its fundamental idea is the transformation of the stochastic data generation process $M_r(\boldsymbol{\theta})$ to a deterministic mapping $g_i(\boldsymbol{\theta})$, by sampling the variables that produce the randomness $\mathbf{v}_i \sim p(\mathbf{V})$. Formally, in every stochastic process the randomness is influenced by a vector of random variables $\mathbf{V}$, whose state is unknown before the execution of the simulation; sampling the state makes the procedure deterministic, namely $g_i(\boldsymbol{\theta}) = M_d(\boldsymbol{\theta}, \mathbf{V} = \mathbf{v}_i)$. This approach initially introduced at (**Meeds2015**) with the title *Optimisation Monte Carlo (OMC)*. The ROMC extended this approach by resolving a fundamental failure-mode of OMC. The ROMC describes a methodology for approximating the posterior through a series of algorithmic steps, without explicitly enforcing which algorithms must be utilised for each step[3]; 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) (**1708.00707**). Since the method by published quite recently, if has not been implemented until now in any ML software. This work attempts to provide to the research community a tested and robust implementation for further experimentation and possible extensions.

[3]The implementation chooses a specific algorithm for each task, but this choice has just a demonstrative value; any appropriate algorithm can be used instead.

1.2 Outline of Thesis

The remainder of the dissertation is organised as follows. In Chapter 2, we establish the mathematical formulation; specifically, we initially describe the simulator-based models and provide some fundamental algorithms that have been proposed for performing inference in these set-ups. Afterwards, we provide the mathematical description of the ROMC approach (**Ikonomov2019**). Finally, we depict the mathematical description into an algorithmic view. In Chapter 3, we illustrate the implementation part; we initially provide some information regarding the Python package Engine for Likelihood-Free Inference (ELFI) (**1708.00707**) and subsequently, we present the implementation details of ROMC in this package. In general, the logical connectivity of the dissertation unit Chapter 3 follows the scheme; Mathematical modelling $\rightarrow$ Algorithm $\rightarrow$ Software.

In Chapter 4, we demonstrate the functionalities of the ROMC implementation at some real-world examples; this chapter desires to demonstrate the success of the ROMC method and our implementation's 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 provide an overview of the symbols utilised in the rest of the document. At this level, the quantities are introduced quite informally; 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 the $N-$*dimensional* euclidean space; N does not represent a specific number.

Random Generator

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

Parameters/Random Variables/Symbols

- $D \in \mathbb{N}$, 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 with the 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 randomness of the generator. It is also called nuisance variable, because we are not interested in inferring a posterior distribution on it.
- $\mathbf{v}_i \sim \mathbf{V}$, a specific sample drawn from $\mathbf{V}$
- $\mathbf{Y}_{\boldsymbol{\theta}}$, random variable describing 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 $\mathbf{y}$ points close to the observations, i.e. $\mathbf{y} := \{\mathbf{y} : d(\mathbf{y}, \mathbf{y_0}) \leq \epsilon\}$
- $B^i_{d,\epsilon}$, the set of points defined around $\mathbf{y}_i$ i.e. $B^i_{d,\epsilon} = B_{d,\epsilon}(\mathbf{y}_i)$
- S_i, the set of $\boldsymbol{\theta}$ parameters that generate data close to the observations using the i-th deterministic generator, i.e. $\{\boldsymbol{\theta} : M_d(\boldsymbol{\theta}, \mathbf{v}_i) \in B_{d,\epsilon}(\mathbf{y_0})\}$

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, the L_2 norm: $||\mathbf{x} - \mathbf{y}||_2$

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 $\mathbf{v}$
- $f_i(\boldsymbol{\theta}) = M_d(\boldsymbol{\theta}, \mathbf{v}_i)$, 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 $d(\mathbf{y}, \mathbf{y_0}) \leq \epsilon$, else 0
- $L(\boldsymbol{\theta})$, the likelihood
- $L_{d,\epsilon}(\boldsymbol{\theta})$, the approximate likelihood

2 Background

2.1 Simulator-based models

As already stated at Chapter **??**, 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,

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

where c_ϵ is a proportionality factor dependent on ϵ, needed when $\Pr(M_r(\boldsymbol{\theta}) \in B_{d,\epsilon}(\mathbf{y_0})) \to 0$, as $\epsilon \to 0$. Equation **??** describes that the likelihood of a specific parameter configuration $\boldsymbol{\theta}$ is proportional to the probability that the simulator will produce outputs equal to the observations, using this configuration.

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 the evaluation of the likelihood is intractable. In the typical rejection sampling, a sample obtained from the prior $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$ gets accepted with probability $L(\boldsymbol{\theta})/\max_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$. Though we cannot use this approach out-of-the-box (evaluating $L(\boldsymbol{\theta})$ is impossible in our case), we can modify the method incorporating the simulator.

In the discrete case scenario where $\mathbf{Y}_{\boldsymbol{\theta}}$ can take a finite set of values, the likelihood becomes $L(\boldsymbol{\theta}) = \Pr(\mathbf{Y}_{\boldsymbol{\theta}} = \mathbf{y_0})$ and the posterior $p(\boldsymbol{\theta}|\mathbf{y_0}) \propto \Pr(\mathbf{Y}_{\boldsymbol{\theta}} = \mathbf{y_0})p(\boldsymbol{\theta})$; hence, 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 accept $\boldsymbol{\theta}_i$ only if $\mathbf{y}_i = \mathbf{y_0}$.

The method above becomes less useful as the finite set of $\mathbf{Y}_{\boldsymbol{\theta}}$ values grows larger, since the probability of accepting a sample 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 $B_{d,\epsilon}(\mathbf{y_0}) := \{\mathbf{y} : d(\mathbf{y}, \mathbf{y_0}) \leq \epsilon\}$ where $d(\cdot, \cdot)$ can represent any valid distance. With this modification, the maintained samples follow the 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 $\mathbf{y} \in \mathbb{R}^D$ lies in a high-dimensional space, generating samples inside $B_{d,\epsilon}(\mathbf{y_0})$ becomes rare even when ϵ is relatively large; this is the curse of dimensionality. As a representative example lets make the following hypothesis;

- d is set to be the Euclidean distance, hence $B_{d,\epsilon}(\mathbf{y_0}) := \{\mathbf{y} : ||\mathbf{y} - \mathbf{y_0}||_2^2 < \epsilon^2\}$ is a hyper-sphere with radius ϵ and volume $V_{hypersphere} = \frac{\pi^{D/2}}{\Gamma(D/2+1)}\epsilon^D$
- the prior $p(\boldsymbol{\theta})$ is a uniform distribution in a hyper-cube with side of length 2ϵ and volume $V_{hypercube} = (2\epsilon)^D$
- the generative model is the identity $\mathbf{y} = f(\boldsymbol{\theta}) = \boldsymbol{\theta}$

then the probability of drawing a sample inside the hypersphere equals the fraction of the volume of a hypersphere inscribed in a hypercube:

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

We observe that the probability tends to 0, independently of ϵ; enlarging ϵ will not increase the acceptance rate. Intuitively, we can think that in high-dimensional spaces the volume of the hypercube concentrates at its corners. This generates 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 dimensionality-reduction step, that redefines the area as $B_{d,\epsilon}(\mathbf{y_0}) := \{\mathbf{y} : d(T(\mathbf{y}), T(\mathbf{y_0})) \leq \epsilon\}$, is called *summary statistic* extraction, since the distance is not measured on the actual outputs, but on a summarisation (i.e. lower-dimension representation) of them.

2.1.3 Approximations introduced so far

So far, we have introduced some approximations for inferring 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. This modification leads to the approximate posterior introduced in (**??**)
- T introduces some 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 $d(T(\mathbf{y}), T(\mathbf{y_0})) \leq \epsilon$

In the following sections, we will not use the summary statistics in our expressions for the notation not to clutter. One could understand it as absorbing the mapping $T(\cdot)$ inside the simulator. In any case, all the propositions that will be expressed in the following sections are valid with the use of summary statistics.

2.1.4 Optimisation Monte Carlo (OMC)

Before we define the likelihood approximation as introduced in the OMC, approach lets define the indicator function based on $B_{d,\epsilon}(\mathbf{y})$. 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 $\mathbb{1}_{B_{d,\epsilon}(\mathbf{y})}(\mathbf{x}) = \mathbb{1}_{B_{d,\epsilon}(\mathbf{x})}(\mathbf{y})$, so the expressions can be used interchangeably.

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

Likelihood approximation

Based on equation (**??**) and the indicator function as defined above (**??**), we can approximate the likelihood as:

$$L_{d,\epsilon}(\boldsymbol{\theta}) = \int_{\mathbf{y} \in B_\epsilon(\mathbf{y_0})} p(\mathbf{y}|\boldsymbol{\theta}) d\mathbf{y} = \int_{\mathbf{y} \in \mathbb{R}^D} \mathbb{1}_{B_{d,\epsilon}(\mathbf{y_0})}(\mathbf{y}) p(\mathbf{y}|\boldsymbol{\theta}) d\mathbf{y} \tag{2.5}$$

$$\approx \frac{1}{N} \sum_i^N \mathbb{1}_{B_{d,\epsilon}(\mathbf{y_0})}(\mathbf{y}_i), \text{ where } \mathbf{y}_i \sim M_r(\boldsymbol{\theta}) \tag{2.6}$$

$$\approx \frac{1}{N} \sum_i^N \mathbb{1}_{B_{d,\epsilon}(\mathbf{y_0})}(\mathbf{y}_i) \text{ where } \mathbf{y}_i = M_d(\boldsymbol{\theta}, \mathbf{v}_i), \mathbf{v}_i \sim p(\mathbf{v}) \tag{2.7}$$

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, by using the approximation of equation (**??**) we need to draw N new samples for each distinct evaluation of $L_{d,\epsilon}(\boldsymbol{\theta})$; this makes this approach quite inconvenient from a computational point-of-view. For this reason, we choose to approximate the integral as in equation (**??**); the nuisance variables are sampled once $\mathbf{v}_i \sim p(\mathbf{v})$ and we count the fraction of samples that lie inside the area

using the deterministic simulators $M_d(\boldsymbol{\theta}, \mathbf{v}_i) \forall i$. Hence, the evaluation $L_{d,\epsilon}(\boldsymbol{\theta})$ for each different $\boldsymbol{\theta}$ does not imply drawing new samples all over again. Based on this approach, the unnormalised approximate posterior can be defined as:

$$p_{d,\epsilon}(\boldsymbol{\theta}|\mathbf{y_0}) \propto p(\boldsymbol{\theta}) \sum_{i}^{N} \mathbb{1}_{B_{d,\epsilon}(\mathbf{y_0})}(\mathbf{y}_i) \tag{2.8}$$

Further approximations for sampling and computing expectations

The posterior approximation in (**??**) does not provide any obvious way for drawing samples. In fact, the set $S_i = \{\boldsymbol{\theta} : M_d(\boldsymbol{\theta}, \mathbf{v}_i) \in B_{d,\epsilon}(\mathbf{y_0})\}$ can represent any arbitrary shape in the D-dimensional Euclidean space; it can be non-convex, can contain disjoint sets of $\boldsymbol{\theta}$ etc. This observation leads to the need for a further simplification of the posterior for being able to sample from it.

As a side-note, sampling could be performed in a straightforward fashion with 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 is high, drawing a sample with non-zero weight is rare, leading to either poor Effective Sample Size (ESS) or huge execution time.

The OMC proposes a quite drastic simplification of the posterior; it squeezes all regions S_i into a single point $\boldsymbol{\theta}_i^* \in S_i$ attaching a weight w_i proportional to the volume of S_i. For obtaining $\boldsymbol{\theta}_i^*$, a gradient based optimiser is used for minimising $g_i(\boldsymbol{\theta}) = d(\mathbf{y_0}, f_i(\boldsymbol{\theta}))$ and the estimation of the volume of S_i is done using the Hessian approximation $\mathbf{H}_i \approx \mathbf{J}_i^{*T}\mathbf{J}_i^*$, where $\mathbf{J}_i^*$ is the Jacobian matrix of $g_i(\boldsymbol{\theta})$ at $\boldsymbol{\theta}_i^*$. Hence,

$$p(\boldsymbol{\theta}|\mathbf{y_0}) \propto p(\boldsymbol{\theta}) \sum_{i}^{N} w_i \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_i^*) \tag{2.9}$$

$$\boldsymbol{\theta}_i^* = \text{argmin}_{\boldsymbol{\theta}}\, g_i(\boldsymbol{\theta}) \tag{2.10}$$

$$w_i \propto \frac{1}{\sqrt{det(\mathbf{J}_i^{*T}\mathbf{J}_i^*)}} \tag{2.11}$$

The distribution (**??**) provides weighted samples automatically and an expectation can be computed easily with the following equation,

$$E_{p(\boldsymbol{\theta}|\mathbf{y_0})}[h(\boldsymbol{\theta})] = \frac{\sum_i^N w_i p(\boldsymbol{\theta}_i^*) h(\boldsymbol{\theta}_i^*)}{\sum_i^N w_i p(\boldsymbol{\theta}_i^*)} \tag{2.12}$$

2.2 Robust Optimisation Monte Carlo (ROMC) approach

The simplifications introduced by OMC, although quite useful from a computational point-of-view, they suffer from some significant failure modes:

- The whole acceptable region S_i, for each nuisance variable, shrinks to a single point $\boldsymbol{\theta}_i^*$; this simplification may add significant error when then the area S_i is relatively big.

- The weight w_i is computed using information from $\boldsymbol{\theta}_i^*$, i.e. the curvature of g_i at the point $\boldsymbol{\theta}_i^*$. This approach can introduce significant error when g_i is almost flat at $\boldsymbol{\theta}_i^*$, leading to a $\det(\mathbf{J}_i^{*T}\mathbf{J}_i^*) \to 0 \Rightarrow w_i \to \infty$, thus dominating the posterior.

- There is no way to solve the optimisation problem $\boldsymbol{\theta}_i^* = \text{argmin}_{\boldsymbol{\theta}}\, [g_i(\boldsymbol{\theta})]$ when g_i is not differentiable.

2.2.1 Sampling and computing expectation in ROMC

The ROMC approach resolves the aforementioned issues. Instead of collapsing the acceptance regions into single points, it tries to approximate them with a bounding box and then defines a uniform distribution over it.[4], which serves as the proposal distribution for importance sampling. If we define as q_i, the uniform distribution defined on the $i-th$ bounding box, weighted sampling is performed as:

$$\boldsymbol{\theta}_{ij} \sim q_i \tag{2.13}$$

$$w_{ij} = \frac{\mathbb{1}_{B_{d,\epsilon}(\mathbf{y_0})}(M_d(\boldsymbol{\theta}_{ij}, \mathbf{v}_i))p(\boldsymbol{\theta}_{ij})}{q(\boldsymbol{\theta}_{ij})} \tag{2.14}$$

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

$$E_{p(\boldsymbol{\theta}|\mathbf{y_0})}[h(\boldsymbol{\theta})] \approx \frac{\sum_{ij} w_{ij} h(\boldsymbol{\theta}_{ij})}{\sum_{ij} w_{ij}} \tag{2.15}$$

2.2.2 Construction of the proposal region

In this section we will describe mathematically the steps needed for computing the proposal distributions q_i. There will be also presented a Bayesian optimisation alternative when gradients are not available.

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 constructing 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:

$$\min_{\boldsymbol{\theta}} \quad g_i(\boldsymbol{\theta}) = d(\mathbf{y_0}, f_i(\boldsymbol{\theta})) \tag{2.16a}$$

$$\text{subject to} \quad g_i(\boldsymbol{\theta}) \leq \epsilon \tag{2.16b}$$

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 can be treated as unconstrained, accepting the optimal point $\boldsymbol{\theta}_i^* = \text{argmin}_{\boldsymbol{\theta}} g_i(\boldsymbol{\theta})$ only if $g_i(\boldsymbol{\theta}_i^*) < \epsilon$.

Gradient-based approach

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

Bayesian optimisation approach

In cases where the gradients are not available, the Bayesian optimisation scheme provides an alternative choice (**Shahriari2016**). With this approach, apart from obtaining 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 for making the method more efficient. Specifically, in the construction of the proposal region and in equations (**??**), (**??**), (**??**) it could replace g_i in the evaluation of the indicator function, providing a major speed-up.

[4]The description on how to estimate the bounding box is provided in the following chapters.

Construction of the proposal area q_i

After obtaining a $\boldsymbol{\theta}_i^*$ such that $g_i(\boldsymbol{\theta}_i^*) < \epsilon$, we need to construct a bounding box around it. The bounding box must contain the acceptance region around $\boldsymbol{\theta}_i^*$, i.e. $\{\boldsymbol{\theta} : g_i(\boldsymbol{\theta}) < \epsilon \text{ and } d(\boldsymbol{\theta}, \boldsymbol{\theta}_i^*) < M\}$. The second condition $d(\boldsymbol{\theta}, \boldsymbol{\theta}_i^*) < M$ is meant to describe that if $S_i := \{\boldsymbol{\theta} : g_i(\boldsymbol{\theta}) < \epsilon\}$ contains a number of disjoint sets of $\boldsymbol{\theta}$ that respect $g_i(\boldsymbol{\theta}) < \epsilon$, we want our bounding box to fit only the one that contains $\boldsymbol{\theta}_i^*$. We seek for a bounding box that is as tight as possible to the local acceptance region (enlarging the bounding box without a reason decreases the acceptance rate) but large enough for not discarding accepted areas.

In contrast with the OMC approach, we construct the bounding box by obtaining search directions and querying the indicator function as we move on them. The search directions $\mathbf{v}_d$ are computed as the eigenvectors of the curvature at $\boldsymbol{\theta}_i^*$ and a line-search method is used to obtain the limit point where $g_i(\boldsymbol{\theta}_i^* + \kappa \mathbf{v}_d) \geq \epsilon$[5]. The Algorithm **??** describes the method in-depth. After the limits are obtained along all search directions, we define bounding box and the uniform distribution q_i. This is the proposal distribution used for the importance sampling as explained in (**??**).

2.3 Algorithmic description of ROMC

In this section, we will provide the algorithmic description of the ROMC method; how to solve the optimisation problems using either the gradient-based approach or the Bayesian optimisation alternative and the construction of the bounding box. Afterwards, we will discuss the advantages and disadvantages of each choice in terms of accuracy and efficiency.

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

Training part

At the training (fitting) part, the goal is the estimation of the proposal regions q_i. The tasks are (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 $d_i^* \leq \epsilon$ and (e) building the bounding box for obtaining the proposal region q_i. If gradients are available, using a gradient-based method is advised 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 requires two evaluations of g_i for **every** parameter $\boldsymbol{\theta}_d$, which works adequately well for low-dimensional problems. When gradients are not available or g_i is not differentiable, the Bayesian optimisation paradigm exists as an alternative solution. In this scenario, the training part becomes slower due to fitting of the surrogate model and the blind optimisation steps. Nevertheless, the subsequent task of computing the proposal region q_i becomes faster since $\hat{d}_i$ can be used instead of g_i; hence we avoid to run the simulator $M_d(\boldsymbol{\theta}, \mathbf{v}_i)$ for each query. The algorithms **??** and **??** present the above procedure.

Inference Part

Performing the inference includes one or more of the following three tasks; (a) evaluating the unnormalised posterior $p_{d,\epsilon}(\boldsymbol{\theta}|\mathbf{y_0})$ (b) sampling from the posterior $\boldsymbol{\theta}_i \sim p_{d,\epsilon}(\boldsymbol{\theta}|\mathbf{y_0})$ (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 using the equation **??**, so we will not discuss it separately.
Evaluating the unnormalised posterior requires solely the deterministic functions g_i and the prior distribution $p(\boldsymbol{\theta})$; 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.
Sampling is performed by getting n_2 samples from each proposal distribution q_i. For each sample $\boldsymbol{\theta}_{ij}$, the indicator function is evaluated $\mathbb{1}_{B^i_{d,\epsilon}(\mathbf{y_0})}(\boldsymbol{\theta}_{ij})$ for checking if it lies inside the acceptance region. If

[5] $-\kappa$ is used as well for the opposite direction along the search line

so the corresponding weight is computed as in (**??**). As before, if a surrogate model $\hat{d}$ is available, it can be utilised for evaluating the indicator function. At the sampling task, the computational benefit of using the surrogate model is more valuable compared to the evaluation of the posterior, 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 algorithms **??** and **??**.

In summary, we can state that the choice 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.

Algorithm 1 Training Part - Gradient approach. Requires $g_i(\boldsymbol{\theta}), p(\boldsymbol{\theta})$

for $i \leftarrow 1$ to n **do**
  Obtain $\boldsymbol{\theta}_i^*$ using a Gradient Optimiser
  if $g_i(\boldsymbol{\theta}_i^*) > \epsilon$ **then**
    go to 1
  else
    Approximate $\mathbf{J}_i^* = \nabla g_i(\theta)$ and $H_i \approx \mathbf{J}_i^T \mathbf{J}_i$
    Use Algorithm **??** to obtain q_i
return $q_i, p(\theta), g_i(\theta)$

Algorithm 2 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 $H_i \approx \mathbf{J}_i^T \mathbf{J}_i$
    Use Algorithm **??** to obtain q_i
return $q_i, p(\theta), \hat{d}_i(\theta)$

Algorithm 3 Computation of the proposal distribution q_i; Needs, a model of distance d, optimal point $\boldsymbol{\theta}_i^*$, number of refinements K, step size η and curvature matrix $\mathbf{H}_i$ ($\mathbf{J}_i^T \mathbf{J}_i$ or GP Hessian)

Compute eigenvectors $\mathbf{v}_d$ of $\mathbf{H}_i$ $(d = 1, \ldots, ||\boldsymbol{\theta}||)$
for $d \leftarrow 1$ to $||\boldsymbol{\theta}||$ **do**
  $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}_i^*$
  $k \leftarrow 0$
  repeat
    repeat
      $\tilde{\boldsymbol{\theta}} \leftarrow \tilde{\boldsymbol{\theta}} + \eta\ \mathbf{v}_d$ ▷ Large step size η.
    until $d(f_i(\tilde{\boldsymbol{\theta}}), \mathbf{y_0}) > \epsilon$
    $\tilde{\boldsymbol{\theta}} \leftarrow \tilde{\boldsymbol{\theta}} - \eta\ \mathbf{v}_d$
    $\eta \leftarrow \eta/2$ ▷ More accurate region boundary
    $k \leftarrow k + 1$
  until $k = K$
  Set final $\tilde{\boldsymbol{\theta}}$ as region end point.
  Repeat steps **??** - **??** for $\mathbf{v}_d = -\mathbf{v}_d$
Fit a rectangular box around the region end points and define q_i as uniform distribution

Algorithm 4 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 5 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) **1708.00707** 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 functionality 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 **??**; 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*[6] 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 variables.

```
# 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)
```

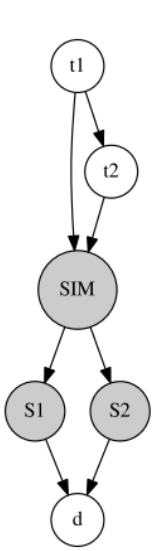

Figure 2: Image taken from **1708.00707**

2.4.2 Inference Methods

All Inference Methods that are implemented in ELFI, follow some common guidelines; (a) their initialisation should by defined by passing the output graph as the initial argument and afterwards come the rest hyper-parameters of the method and (b) they must provide a basic inference functionality, e.g. *<method>.sample()*, which returns a predefined *elfi.Result* object containing the obtained samples along with some other useful functionalities (e.g. plotting 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 Optimisation for Likelihood-Free Inference (BOLFI)*, which is methodologically quite close to the ROMC method we implement in the current dissertation.

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

3 Implementation

In this chapter, we will analyse the details of the implementation of the ROMC inference method to the ELFI package. Sections **??**, **??**, **??**, **??** provide an overview of the functionalities provided by our implementation. These three chapters analyse the implementation steps for training, performing the inference and evaluate the method from the point of view of the user. For illustration purposes, we will use a simple running example throughout the steps. In contrast, in the final section **??** we will delve into the internal details of the implementation in order to provide the information pieces for a possible extension of the method using user-defined methods as the internal building blocks.

3.1 General Design

lala

3.2 Training

3.3 Performing the Inference

3.4 Utilities

3.5 Implementation details for developers

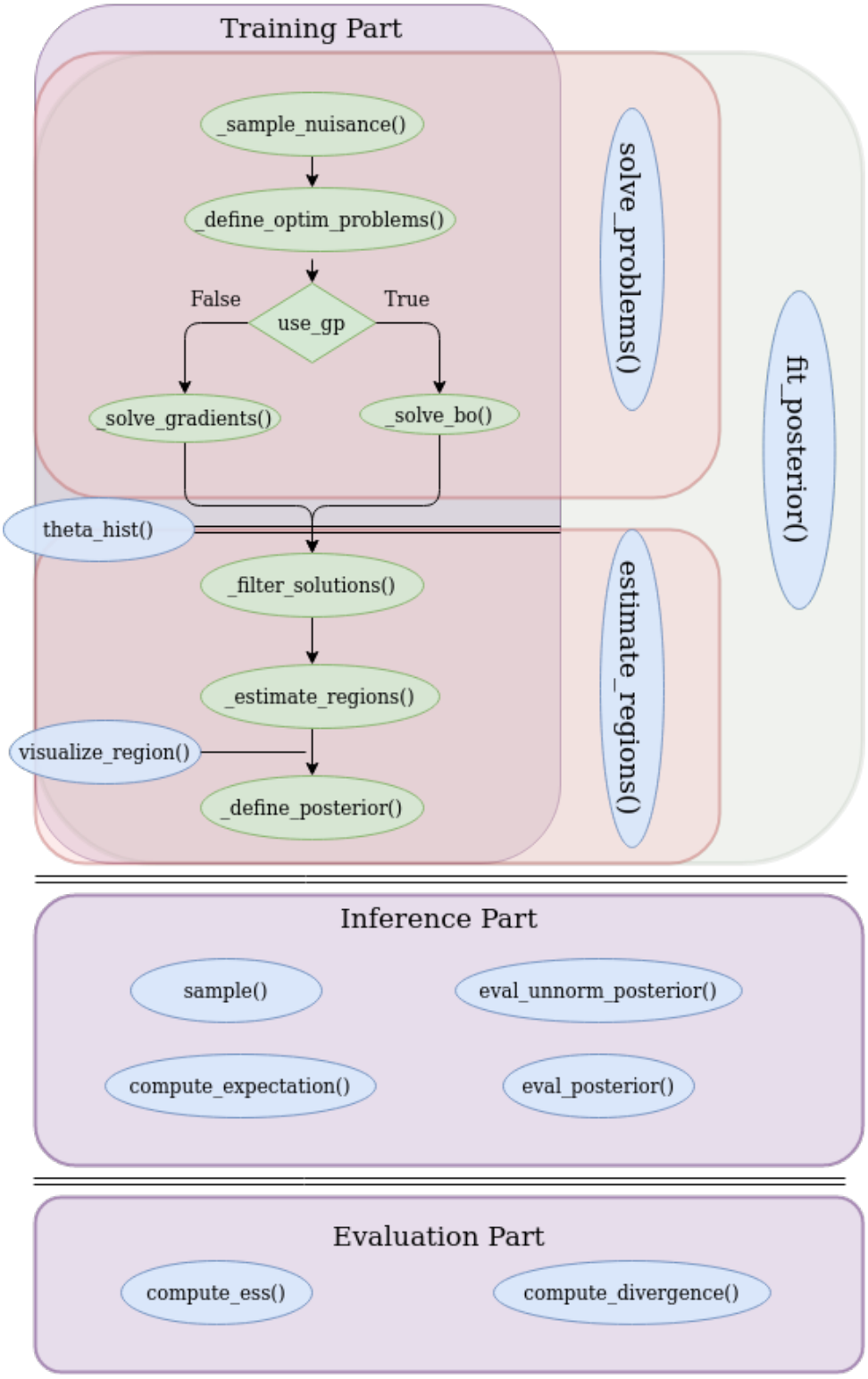

Figure 3: Overview of the ROMC implementation. The training part follows a sequential pattern; the functions in the green ellipses must be called in a sequential fashion for completing the training part and define the posterior distribution. The functions in blue ellipses are the API calls that are called by the user.

Appendices

A An Appendix

Some stuff.

B Another Appendix

Some other stuff.