UNIVERSIDAD AUTÓNOMA DE MADRID

Advanced Methods for Bayesian Optimization in Complex Scenarios

by

Eduardo C. Garrido Merchán

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Escuela Politécnica Superior

Computer Science Department

under the supervision of Daniel Hernández Lobato

June 2021

What is the essence of life? To serve others and to do good.

Aristotle.

Abstract

Some optimization problems require the evaluation of an expensive objective function, whether in economical, computational time or other resources. The analytical expression of the objective function may be unknown. Without an analytical expression, gradients are not accessible and hence are not available for optimization procedures. The evaluation of the objective can also be corrupted by noise. In this case, for the same value of the parameters, the evaluation will return different values. The functions that have the previous characteristics are defined as black-boxes. An example of a black-box function is estimating the generalization error of a machine learning algorithm in terms of its hyper-parameters. Optimizing black-boxes is a task that has recently gained special importance in the machine learning community.

Bayesian optimization (BO) is a set of methods that has been successfully applied for the optimization of black-boxes. BO methods are generally used when the budget of evaluations of the objective function is limited. They deliver great results as they think carefully about which is the best possible next evaluation of the desired objective to be optimized. In order to do so, BO methods rely on a probabilistic model of the objective function. This model generates a predictive distribution of the objective at each input location, that captures the uncertainty about the potential values of the objective. This information is used by BO methods to guide the optimization process. The key for success is that computing the predictive distribution is very cheap compared with evaluating the objective. The probabilistic model is often a Gaussian process (GP) since its predictive distribution can be easily computed.

This thesis proposes several methods to extend the applicability of BO to a broader scope of scenarios. One of these settings involves the simultaneous optimization of multiple objectives under several constraints. For example, the optimization of the generalization error of a deep neural network and its prediction time, with respect to its hyper-parameters, under the constraint that the associated energy consumption on a mobile device is below a specific value. The objectives are conflicting since an accurate network will be large and hence require long prediction times. The solution of a multi-objective problem is a set of points that show the best trade-off for all the objectives. This set is called the Pareto set. Of course, these points must be feasible, i.e., they must satisfy all the constraints. This thesis describes a BO method that uses information theory to solve constrained multi-objective problems. This method, called PESMOC, has state-of-the-art results in this task.

In some practical situations we may have the possibility of evaluating the objective of the problem, at several points, simultaneously. For example, when a cluster of computers is available. Most BO methods are, however, sequential. Therefore, they can may not use all the available computational nodes during the optimization process, resulting in a waste of resources. In this thesis we propose an extension of PESMOC that, at each iteration, suggests a batch of points to be evaluated in parallel. Our results show that such a method, called PPESMOC, improves the results of simple extensions to the parallel setting of PESMOC and other related methods.

One issue of GPs is that they assume real-valued input variables. However, real problems also involve integer-valued and categorical variables. For example, the hyper-parameters of a deep neural network may involve, besides the learning rate, the number of layers, which is an integer-valued variable, and the activation function, which is a categorical variable. This thesis proposes a transformation of the covariance function of a GP to deal simultaneously with integer, categorical and real variables. Our experiments show that the use of this transformation leads to better results in BO methods that rely on using GPs as the probabilistic model.

Finally, this thesis illustrates the use of BO methods to solve a real problem involving robust ocean wave features prediction. With this goal, BO methods are used to tune the hyper-parameters a hybrid Grouping Genetic Algorithm for attribute selection combined with an Extreme Learning Machine (GGA-ELM) for prediction. The proposed BO methodology has been tested in a real problem involving buoys data from the Western coast of the USA. The results obtained show that BO methods outperform a uniform search strategy and the hyper-parameter configuration specified by a human expert.

Resumen

Existen problemas de optimización cuyas funciones objetivo son muy costosas de evaluar, bien en términos del coste computacional, económico o de otro tipo. Además, la expresión analítica de la función objetivo puede ser desconocida. Sin esta información no es posible calcular los gradientes de la función objetivo, que no podrán ser utilizados durante el proceso de optimización. Más aún, la evaluación de la función objetivo puede estar contaminada por ruido. En este caso, para la misma configuración de parámetros, dos evaluaciones pueden devolver valores distintos. Las funciones objetivo que tienen estas características se denominan cajas negras. Un ejemplo de una caja negra es la estimación del error de generalización de un algoritmo de aprendizaje automático en términos de sus hiper-parámetros. En particular, la optimización de cajas negras ha adquirido recientemente gran importancia en el campo del aprendizaje automático.

La optimización Bayesiana (OB) define un conjunto de métodos se pueden utilizar para optimizar cajas negras. Estos métodos son efectivos cuando el número posible de evaluaciones de la función objetivo es limitado. En general, proporcionan buenos resultados de optimización ya que analizan cuidadosamente dónde llevar a cabo la siguiente evaluación de la función objetivo a optimizar. Para ello, emplean un modelo probabilístico de la función objetivo que genera una distribución predictiva, para cada valor de la función objetivo, en cada punto del espacio. Esta distribución captura la incertidumbre sobre los posibles valores que puede tomar la función objetivo. La distribución predictiva se usa en la OB para guiar el proceso de optimización. La clave del éxito radica en que calcular la distribución predictiva es muy poco costoso, comparado con el coste de evaluar de la función objetivo. Como modelo probabilístico se suele emplear un un proceso Gaussiano (PG) debido a la sencillez de calcular la distribución predictiva.

Esta tesis extiende la OB a escenarios mas complejos que los descritos. En primer lugar se aborda la optimización simultánea de múltiples cajas negras bajo la presencia de restricciones. Por ejemplo, cuando se optimiza el error de generalización y la velocidad de predicción de una red neuronal profunda, en función a sus hiper-parámetros, tales que el consumo de energía de la red en un teléfono móvil sea inferior a un cierto valor. Estos objetivos entran en conflicto uno con otro, ya que redes con poco error serán más grandes y requerirán mayor tiempo de predicción. Por ello, la solución de un problema multi-objetivo es el conjunto de puntos que captura el mejor equilibrio entre objetivos, llamado conjunto de Pareto. Cuando hay restricciones, estos puntos deben respetar todas las restricciones. Esta tesis describe PESMOC, un método de OB basado en la teoría de la información que permite solucionar los problemas descritos. PESMOC produce resultados que se condsideran como el estado del arte en este tipo de problemas.

En determinadas ocasiones puede existir la posiblidad de evaluar la caja negra múltiples veces en paralelo. Por ejemplo, cuando se dispone de un cluster de ordenadores. Sin embargo, la OB estandar es secuencial, desperdiciando recursos al considerar una única evaluación de la función objetivo por iteración. En esta tesis se propone una generalización de PESMOC que propone, en cada iteración, un conjunto de puntos para ser evaluados en paralelo. Este método, PPESMOC, mejora los resultados de extensiones simples de PESMOC y de otros métodos relacionados.

Muchos problemas de optimización tienen variables de tipo entero y/o categórico. Por ejemplo, ajustar en una red neuronal la tasa de aprendizaje, variable de valor real; el número de capas, variable de tipo entero; y la función de activación, variable categórica. Por desgracia, los PGs asumen variables de valores reales. Se propone una transformación de la función de covarianza de un PG para permitir simultáneamente variables reales, de tipo entero y categóricas. Mostramos experimentos en los que la transformación mejora los resultados de la OB basada en PGs.

Finalmente, esta tesis muestra un caso real de uso de la OB para la predicción robusta de características de olas oceánicas. En este problema, la OB se usa para optimizar los hiper-parámetros de un algoritmo híbrido genético agrupado, encargado de la selección de atributos, usado en combinación con una máquina de aprendizaje extremo para hacer predicciones. Este sistema se ha evaluado en datos reales de boyas de la Costa Oeste de EEUU. Los resultados obtenidos muestran que la OB mejora a la búsqueda aleatoria y al criterio de un experto.

Acknowledgements

Quisiera empezar enfatizando que esta tesis no hubiera podido ser firmada por mi sino hubiera gozado de circunstancias personales absolutamente privilegiadas que, por desgracia, muy poca gente puede tener: nacer en el país y en la época en la que he nacido, ser criado en un entorno amigable, tener mis necesidades cubiertas y recibir mucho amor. Ninguna semilla da fruto sino es regada, independientemente de las propiedades de dicha semilla. Este fruto es un resultado del amor de mis seres queridos y de mucha gente que, en el pasado, se ha sacrificado por todos para construir nuestra sociedad. Gracias a cada persona que ha dejado mas bien que mal en el mundo y que, por ello, ha posibilitado que esta tesis ha podido salir adelante. Yo no hubiera escrito una tesis si hubiera nacido pobre, en un entorno violento o sin amor. Esta tesis es para cada persona que alguna vez ha querido hacer una tesis y por las circunstancias no ha podido. Toda vuestra.

Mi director de tesis ha sido Daniel Hernández Lobato. Entre sus muchísimas virtudes destaca su ilimitada paciencia, su cordialidad, su afán de superación, su empatía, su perfeccionismo, su humildad y su preocupación por los problemas que tienen sus estudiantes. No tengo palabras para agradecer lo muchísimo que me ha ayudado y ha enseñado a lo largo de estos años. Creo estar seguro de que me va a ser imposible devolverle ni una décima parte de todo lo que me ha dado. Aún así, lo intentaré.

Quiero agradecer a mis abuelos todo su amor. A mi abuelo Juan Merchán, gracias por ser un ejemplo de fortaleza. A mi abuela Antonia Asensio, gracias por ser un ejemplo de firmeza. A mi abuelo Antonio Garrido, gracias por la ilusión que tenías depositada en mi. A mi abuela Lorenza Santos, gracias por los ratos que compartiste conmigo.

Quisiera agradecer también a mis suegros por su gran preocupación con respecto al bienestar de nuestra familia. A Jose Luis Córdoba por ser un ejemplo de solidaridad. A Maria Sánchez por ser un ejemplo de fuerza de voluntad.

He tenido la inmensa fortuna de tener los mejores padres que podría desear. Personas a las que les definen actos, no palabras. Personas que me han enseñado que el respeto, el amor, la libertad y el sacrificio a los demás son esenciales. Personas que no me han enseñado con palabras, sino con actos. A mi padre Eduardo Garrido, por ser un ejemplo de sacrificio, constancia y abnegación. A mi madre María Merchán, por ser un ejemplo de amor filial, ascetismo y por hacer de mi todo lo que soy yo.

Durante este doctorado he tenido la bendición de recibir a dos niños maravillosos: a Ramón y a Juan. No tengo palabras para describir el gran amor que siento por ellos. Gracias por vuestro cariño. Gracias por siempre tener un abrazo guardado que surge en el instante mas impensable. Gracias por ser quienes sois, que siempre seais felices.

He conocido a muchas personas, pero a nadie como mi mujer: Irene Córdoba Sánchez. Irene es pura luz. No lo digo porque sea mi mujer. Lo digo objetivamente. Me ha tocado la mayor lotería y a la vez la mayor responsabilidad que a nadie le ha tocado nunca. Ser el marido de Irene. Ninguna palabra te hace justicia, pues es inefable describir lo mucho que te agradezco todo lo que has hecho y todo lo que eres. Gracias por iluminar cada día y cada minuto de mi vida. Los ángeles existen y dejo aquí testimonio de ello.

Deseo concluir, como creyente, manifestando mi mas humilde agradecimiento al Señor. Gracias por darme a conocer tu obra. Gracias por tu ejemplo de amor infinito. Gracias por otorgarme un cuerpo con el que pueda, espero, poder ayudar a los que mas lo necesitan. Gracias por darme amor para que lo pueda compartir. Gracias, siempre.

Contents

Abbreviations

ADF	**A**ssumed **D**ensity **F**iltering
AF	**A**cquisition **F**unction
BO	**B**ayesian **O**ptimization
DGP	**D**eep **G**aussian **P**rocess
EI	**E**xpected **I**mprovement
EP	**E**xpectation **P**ropagation
GP	**G**aussian **P**rocess
KL	**K**ullback **L**iebler
MCMC	**M**arkov **C**hain **M**onte **C**arlo
PPESMOC	**P**arallel **P**redictive **E**ntropy **S**earch for **M**ultiobjective **O**ptimization with **C**onstraints
PES	**P**redictive **E**ntropy **S**earch
PESMOC	**P**redictive **E**ntropy **S**earch for **M**ultiobjective **O**ptimization with **C**onstraints
RS	**R**andom **S**earch
UCB	**U**pper **C**onfidence **B**ound

To my family

Chapter 1

Introduction

We begin this manuscript with an introduction to Bayesian optimization (BO), a methodology to optimize black-box functions, *i.e.*, functions with an unknown analytical expression, with noisy evaluations and expensive to evaluate. We provide a visual example that illustrates the operations performed in BO to better understand it. BO is usually applied to the optimization of a single black-box but it can be extended to consider broader scenarios, such as the optimization of multiple black-boxes under the presence of several constraints. We enumerate these scenarios here, since they are going to be targeted by proposed BO methods in the following chapters. We continue the exposition with a list of publications that constitute the work related with this thesis. A summary of the chapters of the document is provided along with a guide on how to read them depending on the interests of the reader. Finally, we conclude the chapter with an exposition of the notation that is going to be used in this thesis.

1.1 Introduction

Optimization is the subfield of computer science that is concerned with finding the global extremum (maximum or minimum) of a mathematically defined function (the objective function) in some region of interest (Törn and Žilinskas, 1989). This process is expected to deliver a solution within the region of interest that is close to the global extremum with little or no human supervision. Assuming minimization, this scenario can be represented as: $\mathbf{x}^\star = \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$, where $f : \mathbb{R}^d \to \mathbb{R}$ is an example of the defined function, $\mathcal{X}$ is the region of interest or input space and $\mathbf{x}^\star$ is the global extremum belonging to the input space, which is a d-dimensional real-valued space $\mathbb{R}^d$. Optimization algorithms recommend solutions $\mathbf{y}$ at the end of the process whose regret $r = |f(\mathbf{x}^\star) - \mathbf{y}|$ is, desirably, as close as zero as possible (Pardalos and Romeijn, 2013). If the algorithm achieves zero regret, then, it has found the global extremum, which is the ideal solution.

The classical optimization framework makes assumptions about the function whose global extremum needs to be retrieved, as, for example, that the function needs to be mathematically defined. That is, we must have access to its analytical expression and its gradients. However, real problems include scenarios where the analytical expression of the function to be optimized is unknown. Hence, gradients are not accesible. For example, consider a scenario where the taste of a low calorie cookie needs to be optimized. There is no analytical expression for its taste. However, it is possible to perform evaluations by

letting people taste the cookie to give their opinion about it. Another example would be to optimize the hyper-parameters of a deep neural network. There is not an analytical expression of the prediction error on a validation set of a deep neural network.

The gradients that are computed by having access to the analytical expression of the objective function are used by gradient-based optimization algorithms, like l-BFGS (Zhu et al., 1997). Therefore, these algorithms cannot be applied when we do not have access to the gradients of the functions that we want to optimize. When gradients are not available, a set of algorithms called metaheuristics, that do not need the gradients of a function to optimize it, can be applied (Glover and Kochenberger, 2006). Metaheuristics, as global optimization algorithms, also attempt to find the global extremum, but without the need of gradients. Metaheuristics are defined as iterative processes which guide a subordinate heuristic. This heuristic intelligently balances the exploration and exploitation of the search space. It does so by learning strategies that are used to structure information in order to efficiently find near-optimal solutions (Osman and Laporte, 1996). Some examples of these strategies are genetic algorithms or constraint logic programming (Davis, 1991; Jaffar and Maher, 1994). One issue that emerges with these algorithms is that, to deliver a solution with low regret, they need a high number of evaluations of the objective. Therefore, these algorithms are not thought for situations where the available budget of evaluations is low. They assume that evaluations are cheap, so determining the location of a new evaluation must not be more expensive that evaluating the objective function. As a consequence, these algorithms may require a high number of evaluations to achieve a solution with low regret with respect to the optimum of the objective function.

An example of an expensive to evaluate function is the hyper-parameter tuning of a machine learning algorithm. The hyper-parameter tuning problem consists in retrieving the values of the hyper-parameters of machine learning algorithms that minimize the estimation of the generalization error of these algorithms for a given dataset. In particular, the goal is to find $\theta^* \in \mathbf{\Theta} : \theta^* = \arg\min_{\theta \in \mathbf{\Theta}} f(\theta, \mathcal{D})$. In the previous expression, θ^* are the optimum values of the hyper-parameters of a given machine learning algorithm trained on a particular dataset, $\mathbf{\Theta}$ is the parameter space of the machine learning algorithm and f is the function that estimates the generalization error of the machine learning algorithm on a given dataset $\mathcal{D}$. The evaluation of f in such a setting can demand a high computational and economical cost. In particular, for big datasets and machine learning methods that need to fix a high amount of parameters in training time. As metaheuristics need a high number of iterations to provide a result with low regret and the hyper-parameter tuning problem requires a big amount of time for each evaluation, estimating the global extremum of f with a metaheuristic is infeasible. For example, some metaheuristics, as genetic algorithms, are based on mixing a population of individuals. The total number of potential combinations of individuals are huge, so exploring a relevant subset of these combinations requires a high number of iterations.

In BO scenarios it is common that the observations of f may be contaminated by a random variable which is essentially noise (Brochu et al., 2010). More formally, $y = f(\mathbf{x}) + \epsilon$, where y is the observed value at a given iteration of the optimization process, f is the objective function that we do not have direct access to and ϵ is the random variable that represents the noise in the problem. The noise ϵ is generally modelled using a Gaussian distribution, although it is possible to model it with some other distribution. As the objective function f is contaminated with noise ϵ, the result of evaluating f is stochastic, not deterministic. This means that two evaluations of f at the same input location $\mathbf{x}$ will produce different observation values y. Therefore, BO methods need to take into account the stochastic property of the evaluation process in order to provide

an acceptable result. In this chapter, we introduce BO, which is a methodology that efficiently solves the optimization of functions with the characteristics that are described above (Shahriari et al., 2015).

We have briefly discussed the scenario where we optimized a single function, but there are broader scenarios than the one described before, which is defined as the classical BO scenario. For example, in broader scenarios, we can consider several functions to be optimized simultaneuosly instead of a single one. Another example will consider the optimization of functions of categorical, integer and real-valued variables instead of considering only functions of real-valued variables, as in the classical BO setting. This thesis will provide methods that enhance BO in order to address these broader scenarios. All these settings, that will be described further in detail, are found in real situations. Hence, the techniques that are going to be described in this manuscript are relevant and useful for society.

This chapter gives an introduction for this thesis. We begin with an initial motivating visual example illustrating the basic operations of BO. The enhancements of BO methods described in this thesis apply to generalizations of the standard BO scenario, that we will describe in Section 1.2. Concretely, we will introduce the BO of multiple objectives under several constraints. We also consider optimizing functions that contain integer-valued and categorical variables and parallel BO, where several candidate points are evaluated simultaneusly. We continue with a section that enumerates the publications that the research work described in this thesis has produced. We also describe related publications of the methods that are going to be proposed. We continue the introduction with a summary by chapters of this work in Section 1.5. Next, we introduce alternatives to read this thesis that may satisfy different interests of the readers and lastly, we conclude the chapter with a section that illustrates the definitions and notation that are going to be used to represent mathematical objects and concepts.

1.2 Bayesian Optimization: A Visual Example

The functions described in the previous section, which are expensive to evaluate, whose evaluations are contaminated by noise and that have no analytical expression, and hence no gradient information available, are called *black-boxes* (Jones et al., 1998). Some optimization scenarios are limited to a budget of evaluations or computational time, making only possible to evaluate the black-box a fixed amount of times. Hence, every single evaluation of the black-box needs to be carefully chosen to minimize the regret. In this setting, the computational time spent by the BO methodology required to suggest a new candidate point at which to evaluate the objective in every iteration is not critical. The computational time is hence going to be governed by time spent in evaluating the black-box. In other words, the complexity of the BO procedure is not critical as we assume that the evaluation cost is going to be much bigger than the computational cost of determining the next point to evaluate. Due to this fact, BO can make use of complex models and procedures to determine, in an intelligent way, a new point to be evaluated.

BO algorithms are designed to optimize black-boxes. They determine the next point to be evaluated based on the predictive distribution of the objective function given by a surrogate model (Ross et al., 1996). The model is fitted sequentially to the observations of the black-box (Shahriari et al., 2015). The probabilistic model generates a predictive distribution of the potential values of the objective function in each point of the input space. The probabilistic model makes hypotheses about the shape of the function, as for example its smoothness. BO assumes that the objective function can be explained

by the probabilistic model, *i.e.*, the assumptions of the model about the objective function are correct. BO iteratively suggests a new point that represents the best tradeoff between exploration and exploitation using the predictive distribution (Lizotte, 2008). In particular, it is interesting to focus the search on unexplored areas because they may contain the optimum of the objective function but also to exploit areas where the prediction of the probabilistic model shows promising values. This tradeoff is computed by a member of a set of heuristic criteria called acquisition functions (Hoffman et al., 2011). Some acquisition functions use *Information Theory* to determine this tradeoff, such as the Entropy Search acquisition function (Hennig and Schuler, 2012; Villemonteix et al., 2009). Acquisition functions return a real positive value for every point of the input space. This value represents the expected utility of evaluating that point in the search for the optimum. A key fact is that acquisition functions only depend on the model, being cheap to compute. Also, optimizing the acquisition function is much cheaper than evaluating the objective function.

As we have previously seen, we can make assumptions about the function that we want to optimize or have prior knoweldge about it. If we do not know anything about the objective function, we can still assume, without loss of generality: smoothness, stationarity or continuity. If we apply a grid search or random search, we do not exploit these assumptions or use any available prior knowledge about the function. In this case, we just perform pure exploration. Ideally, we want a mechanism that, given a set of assumptions, can use them to perform an intelligent search. The probabilistic surrogate model, for example a Gaussian process (GP), is a mechanism to encode those assumptions about the objective function (Rasmussen, 2003). A GP is a stochastic process where any point $\mathbf{x} \in \mathbb{R}^d$ is assigned a random variable $f(\mathbf{x})$ and where the joint distribution of a finite number of these variables $p(f(\mathbf{x_1}), ..., f(\mathbf{x}_N))$ is a multivariate Gaussian: $p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\boldsymbol{\mu}, \mathbf{K})$, where $\mathbf{f} = (f(\mathbf{x}_1), ..., f(\mathbf{x}_N))^{\mathrm{T}}$, $\boldsymbol{\mu} = (m(\mathbf{x}_1), ..., m(\mathbf{x}_N))^{\mathrm{T}}$ and $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. $m(\cdot)$ is the mean function and $k(\cdot, \cdot)$ is a positive definite covariance function (Rasmussen, 2003). The GP will have a lower uncertainty in the neighbourhood of an observed value, as it assumes a certain degree of smoothness. The level of smoothness, for every dimension of the input space, can be specified by the hyper-parameters of the GP. The uncertainty over all the space of values gets lower every new time that we fit the GP to a new observation of the objective. Hence, the uncertainty about the objective function decreases with every step of BO.

The GP posterior distribution, the acquisition function and the objective function can be visualized in low dimensional problems to get an intuition of how BO works. This visualization is useful to show how and why does BO work. The behavior of BO in a toy scenario is visualized as follows. Let us consider the optimization of the 1-dimensional function given by the next analytical expression:

$$f(x) = \sin(x) + \sin(\frac{10}{3}x), x \in [0, 4] \subset \mathbb{R}^1 \,. \tag{1.1}$$

We can visualize the previous objective function in Figure 1.1. Suppose now that we do not have access to the evaluation of $f(x)$ directly. We do only have access to a corrupted version of the evaluations of f that has been contaminated by i.i.d Gaussian noise with a 0.05 value of standard deviation in all its range. In particular, we would like to optimize the following function:

$$g(x) = f(x) + \epsilon, \epsilon \sim \mathcal{N}(0, 0.05) \,, \tag{1.2}$$

where $\mathcal{N}(0, 0.05)$ denotes a univariate Gaussian distribution with mean equal to 0 and

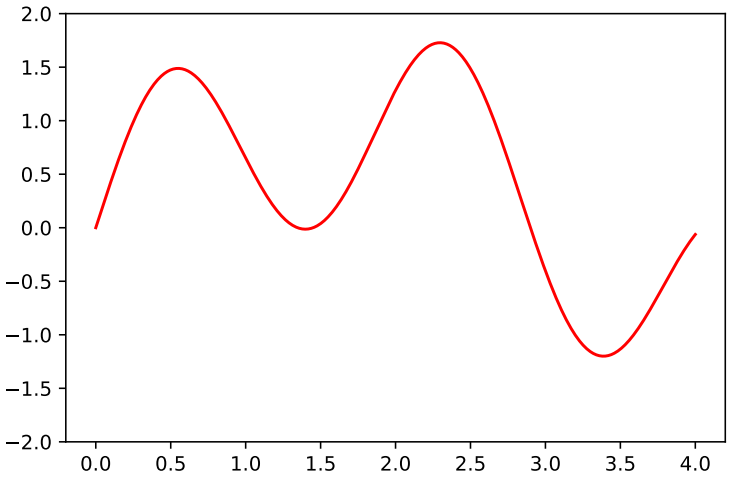

FIGURE 1.1: Toy objective function without being corrupted by noise.

variance 0.05. The ground truth $f(x)$ is now a latent function and we can only observe $g(x)$. For example, we can evaluate $g(x)$ on 5 random points sampled uniformly on $[0, 4]$ as Figure 1.2 shows. But given the observed data of the objective function, there is an infinite number of possible hypotheses than could explain it. We can use a GP

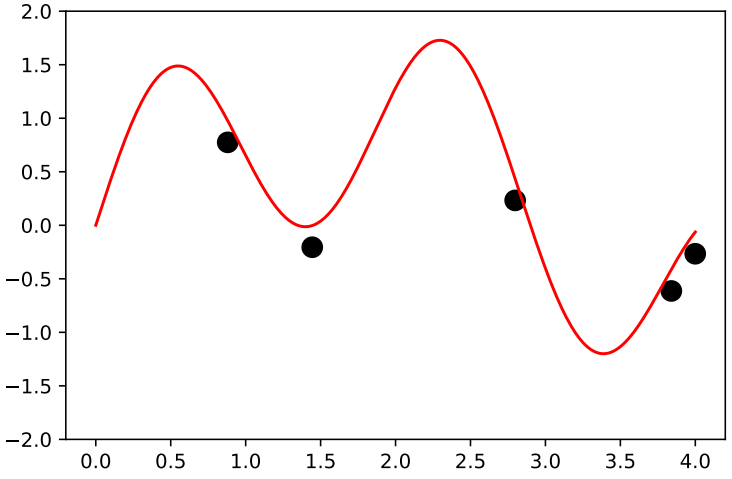

FIGURE 1.2: Observations (black dots) of the toy objective function without being corrupted by noise (red line). As it can be seen, the observations are perturbed by noise added to the toy objective function.

to predict the potential values of the function on the input space. The functions that explain the observed data are potential solutions of the problem. There is an infinite amount of them. Figure 1.3 shows some of these functions. We define these functions as fantasies. Nevertheless, we still do not know which fantasy explains better the black-box to be optimized. Hence, we need to consider all the possible hypotheses pondered by the probability that they explain the data. The GP posterior distribution considers precisely all these functions, as Figure 1.4 represents. In Figure 1.4 we can see the ground truth plotted on red and the prediction that the GP does of the ground truth. We can observe

that both the predictive mean and, logically, the ground truth, explain the data. The Figure shows the predictive distribution of the objective function, characterized by an expected value and a standard deviation of it for each point of the input space. We can observe how the objective function lies within the plotted standard deviation intervals of the predictive distribution of the GP in each point.

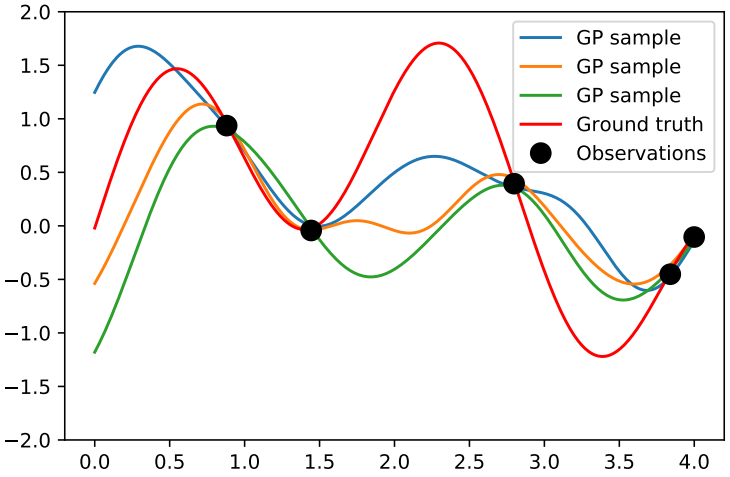

FIGURE 1.3: Hypotheses that explain the data (colored lines except red) which are samples from a GP conditioned to the observed data. The observations are shown as black dots and the ground truth is shown as a colored red line.

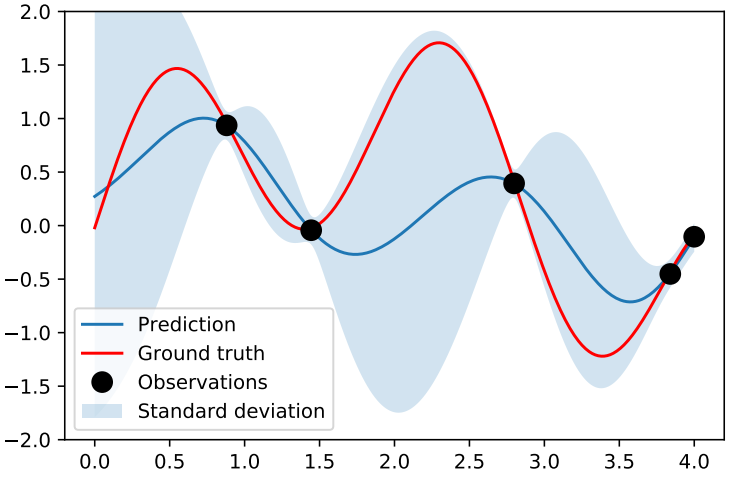

FIGURE 1.4: GP predictive distribution over the ground truth. The predictive mean is shown as a blue line, the GP standard deviation is shown as a blue area, observations are black dots and the ground truth is plotted as a red line. Recall that the ground truth is covered by the GP standard deviation.

We can use the GP to build an acquisition function. In order to do so, we use the predictive distribution to compute the acquisition function. BO uses the information given by the GP to build, in each iteration, a criterion that represents a tradeoff between exploration and exploitation of the global extremum (Rasmussen, 2003). These tradeoffs are encoded in analytical expressions that are defined as acquisition functions. The

acquisition function is a D-dimensional function, where D is the dimension of the input space. This acquisition function indicates, on each point, the expected utility of performing an evaluation of the objective there. By maximizing this acquisition function, we retrieve the point to be evaluated in the next iteration of BO. For example, a popular acquisition function that is simple to evaluate is the GP-UCB acquisition function, given by the following expression:

$$\alpha(x) = -\mu(x) - \kappa\sigma(x)\,. \tag{1.3}$$

This acquisition function, $\alpha(x)$, is an intuitive example of the exploitation-exploration tradeoff. Its analytical expression combines the predictive mean of the GP, $\mu(x)$, to exploit in areas of the input range suspicious of containing the extremum, and the uncertainty given by the standard deviation of the GP predictive distribution, $\sigma(x)$. The uncertainty is high in unexplored areas where we have not evaluated a point, as we can see in Figure 1.4. Those areas could contain the extremum, but we do not know it. It is desirable that BO does not skip unexplored areas that could potentially contain the extremum. So, the acquisition function takes high values on areas with high uncertainty. This is the exploration behavior of BO. The acquisition function also takes high values when the mean $\mu(\mathbf{x})$ takes low values. This behavior of the acquisition function enforces BO to evaluate promising solutions, which will give lower values, close to the minimum. This is the exploitation behavior of BO. In Eq. (1.3), κ is a tunable parameter, that balances exploration and exploitation. The acquisition function assigns a real value for each input space point. This value represents the expected utility of evaluating the function at each input space point in the next iteration. The point associated with the maximum of these values will be evaluated in the next BO iteration. Chapter 3 analyzes the most popular acquisition functions, with a special emphasis in those acquisitions that use information theory. Figure 1.5 illustrates graphically the BO steps and shows the shape of GP-UCB and a GP fitted to the data, the acquisition function and its maximum. We can see that the input space point associated with the maximum value of the acquisition function is evaluated in the next BO iteration. We can also observe that the acquisition function has higher values in areas where the prediction of the GP model is lower and the uncertainity is higher. The evaluation value is used to condition the GP at that point. The acquisition will be rebuilt in the next iteration according to the new predictive distribution. Then, the algorithm continues performing the same steps in all remaining iterations. The process ends when the budget of evaluations is consumed. When the iterations stop, we can recommend a solution for the optimization problem by optimizing the mean of the GP or returning the best-observed evaluation. This is the standard scenario where BO operates, but it can be adapted to provide a solution to more complex scenarios. It is important to remark that the visual examples that have been shown so far illustrate the execution of BO in a single dimension. One can think that there are other more effective and less computational costly methods to optimize 1-dimensional functions. Whilst this is true, BO can be applied to D-dimensional problems.

1.3 Bayesian Optimization in Complex Scenarios

This thesis proposes BO extensions to solve broader scenarios than the classical BO setting. The described vanilla scenario consists in retrieving the global extremum of a single function of real-valued variables. Broader scenarios arise in real problems of great interest to the machine learning community. For example, in the hyper-parameter tuning

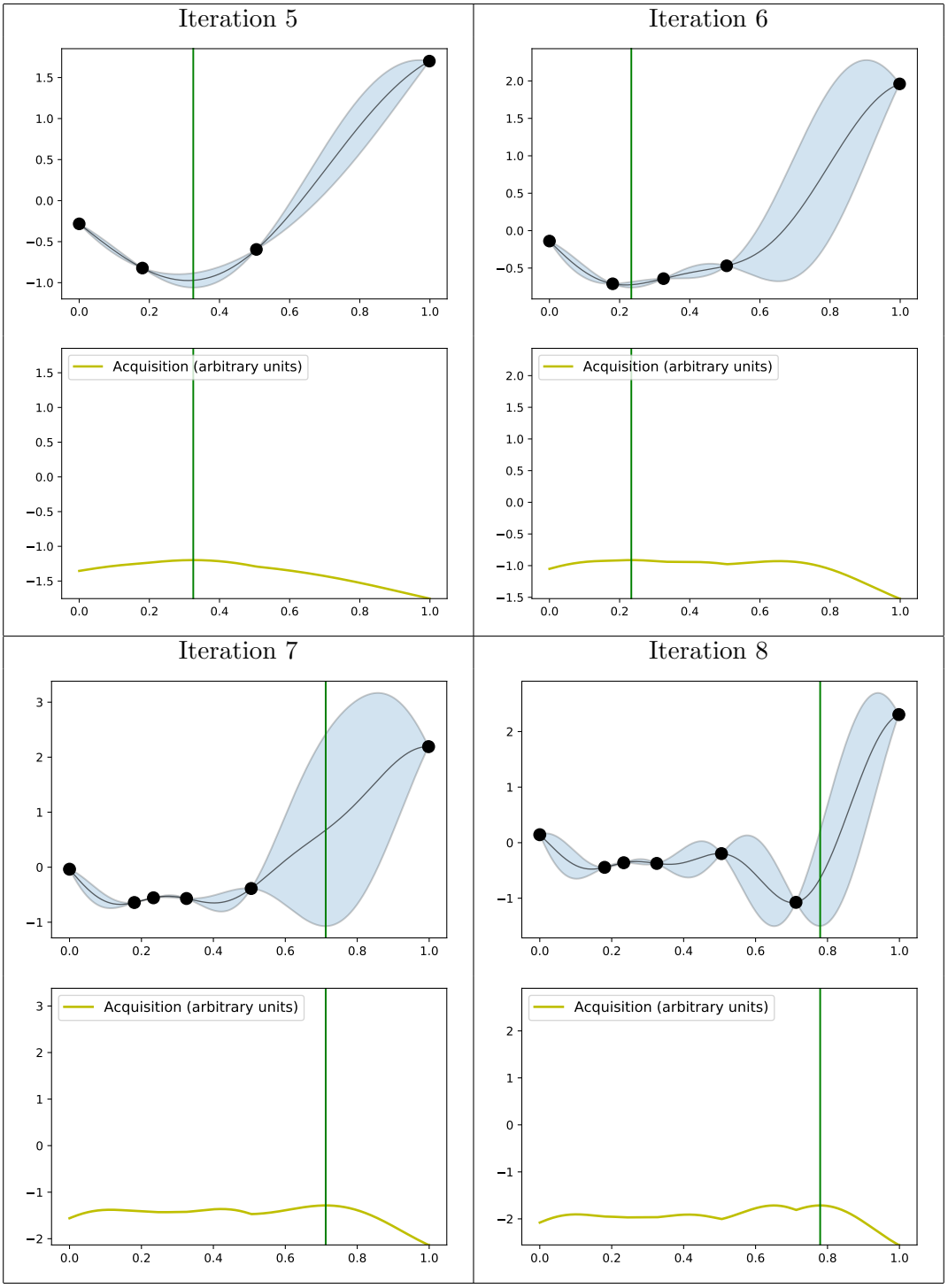

FIGURE 1.5: GP-UCB acquisition function (yellow) built from a GP (blue) conditioned on 5,6,7 and 8 points (black dots) recommended by previous iterations of BO. The next suggestion of BO will be the point given by the maximum of GP-UCB, shown by a vertical green line.

of machine learning algorithms (Snoek et al., 2012). Other examples of these scenarios include: optimizing functions of integer or categorical-valued variables, apart from real-valued ones; optimizing multiple objectives under the presence of several constraints; or providing a batch of points to be evaluated at every iteration. The following paragraphs describe the contributions of this thesis, that are focused on making BO able to address the mentioned broader optimization scenarios:

- **Constrained multi-objective Bayesian optimization**: Some problems concern the optimization of multiple black-boxes simultaneously. Usually, these black-boxes are inversely correlated. Let us consider a hyper-parameter tuning problem, where the estimation of the generalization error and the prediction time of a neural network are minimized. Solutions that minimize the prediction time of the neural network are not expected to deliver a low estimate of the generalization error and vice-versa. The objectives are conflicting since an accurate network will be large and hence require long prediction times. The solution of this problem is a set of points that show the best trade-off for all the objectives. This set is called the Pareto set. The values in function space associated to those points are the Pareto frontier. This scenario is known as multi-objective BO, but there are even more complex scenarios. In particular, it is also interesting to deal with the presence of one or multiple black-box constraints $c_j(\mathbf{x})$ on the solution of the problem. In these scenarios, we will only consider a point as valid if it satisfies all the constraints, *i.e.*, if $c_j(\mathbf{x}) \geq 0 \quad \forall j < C$, where C is the number of constraints. As an example of a constraint, the previous solutions may only be valid if the associated energy consumption on a mobile device where the neural network is implemented is below a specific value. We illustrate this scenario in Figure 1.6, where two objective functions and a constraint are plotted. We want to extract the Pareto set of these objectives under valid values according to the constraint. This produces a shrinkage of the valid input and Pareto set and its respective Pareto frontier, as we can see in Figure 1.6. We will propose an information-theoretical acquisition function to address these problems (Garrido-Merchán and Hernández-Lobato, 2019b). Several synthetic, benchmark and real experiments demonstrate its usefulness.

- **Integer-valued and categorical variables in Bayesian optimization**: Real problems, such as the hyper-parameter tuning of machine learning algorithms, involve the optimization of functions of integer-valued and categorical variables. For example, the hyper-parameters of a deep neural network may involve, besides the learning rate, the number of layers, which is an integer-valued variable, and the activation function, which is a categorical variable. BO typically uses a GP as the probabilistic surrogate model to build an acquisition function at each iteration. The majority of the kernels of GPs, and concretely those that are used for BO, assume real-valued variables. Hence, standard GPs cannot be used for the BO of functions that contain integer-valued or categorical variables. We include a chapter that illustrates a transformation of the input space points that the kernel of the GP receives to tackle this problem (Garrido-Merchán and Hernández-Lobato, 2019a). That transformation allows a valid GP kernel to deal with integer and categorical-valued variables. With this transformation, BO obtains better results in real problems that include integer-valued and categorical variables.

- **Batch Bayesian optimization**: The sequential nature of BO allows to evaluate only a single point in every iteration. This results in a waste of resources when

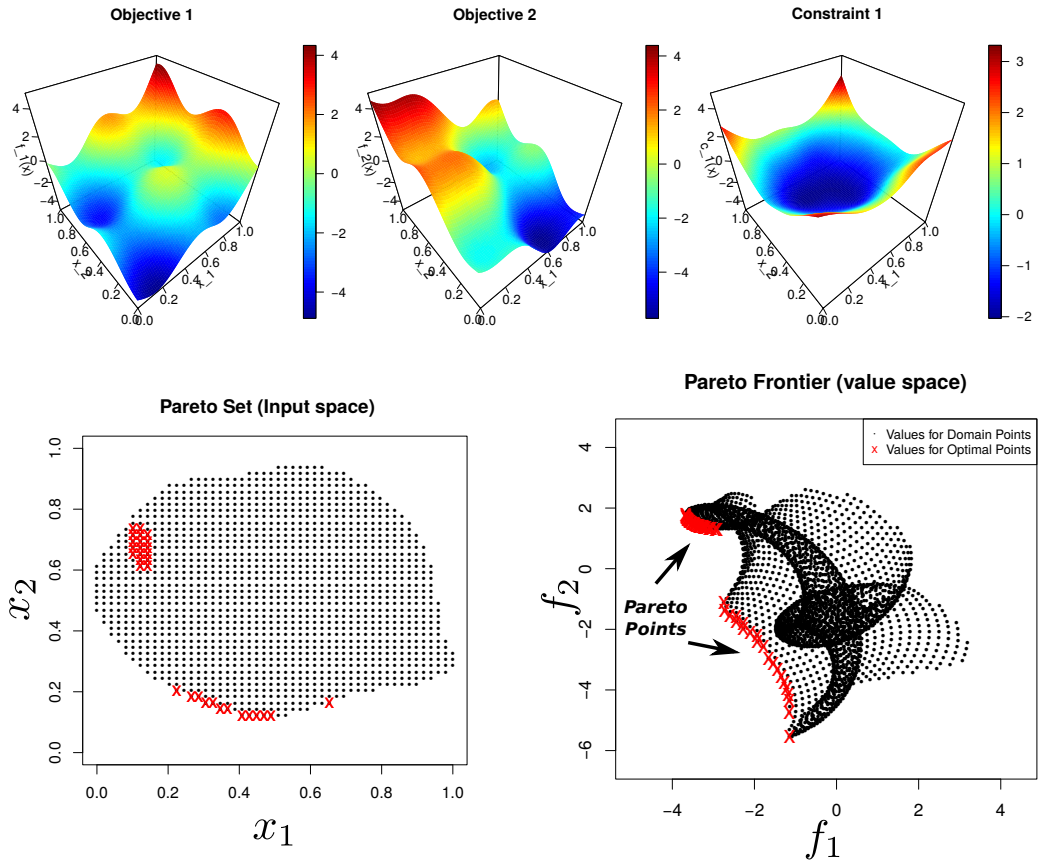

FIGURE 1.6: Multi-objective constrained Bayesian optimization scenario illustrating objectives, constraints, the pareto set and its frontier.

we can make use of a cluster of nodes, where we can evaluate several points simultaneously. Due to this fact, it is desirable to design a procedure that allows BO to provide a set of points at every iteration at which to evaluate the objective. We define this set of points as a batch. Batch BO methods suggest a batch of points for evaluation at every iteration. There are batch methods proposed in literature but none of them applies to the multi-objective constrained scenario. This thesis proposes an extension of an information-theoretical method to carry out parallel evaluations in constrained multi-objective scenarios.

- **Bayesian optimization applied to oceanic waves features prediction**: A novel application of BO to tune the hyper-parameters of a hybrid Grouping Genetic Algorithm for attribute selection combined with an Extreme Learning Machine (GGA-ELM) for prediction is presented. This GGA-ELM approach for prediction of ocean wave features prediction is optimized with BO (Cornejo-Bueno et al., 2018). The application of BO to the optimization of the parameters of the genetic algorithm illustrates how BO can be used in real problems. The proposed BO methodology has been tested in a real problem involving buoys data from the Western coast of the USA. The BO-GGA-ELM approach is applied on two experiments: predicting the significant wave height at a goal marine structure facility and the wave energy flux at the same facility. The results obtained show that BO methods outperform

a uniform search strategy and the hyper-parameter configuration specified by a human expert.

1.4 Publications

This section presents, in chronological order, the work published during the doctoral period in which this thesis was written. We also include other research work related to this thesis, but not directly included on it. Finally, this document includes content that has not been published yet and is under revision.

- Cornejo-Bueno, Laura, Garrido-Merchán, Eduardo C., Hernández-Lobato, Daniel and Salcedo-Sanz, Sancho, 2017. Bayesian optimization of a hybrid prediction system for optimal wave energy estimation problems. In *International Work-Conference on Artificial Neural Networks*, (pp. 648-660).
- Cornejo-Bueno, Laura, Garrido-Merchán, Eduardo C., Hernández-Lobato, Daniel and Salcedo-Sanz, Sancho, 2018. Bayesian optimization of a hybrid system for robust ocean wave features prediction. *Neurocomputing*, 275, pp.818-828.
- Garrido-Merchán, Eduardo C. and Hernández-Lobato, Daniel, 2019. Predictive entropy search for multi-objective Bayesian optimization with constraints. *Neurocomputing*.
- Garrido-Merchán, Eduardo C. and Hernández-Lobato, Daniel, 2019. Dealing with categorical and integer-valued variables in Bayesian optimization with Gaussian processes. *Neurocomputing*.

Related Work

- Balázs, C., van Beekveld, M., Caron, S., Dillon, B. M., Farmer, B., Fowlie, A., Garrido-Merchán, Eduardo C. and White, M. (2021). A comparison of optimisation algorithms for high-dimensional particle and astrophysics applications. Journal of High Energy Physics, 2021(5), 1-46.
- Córdoba, Irene, Garrido-Merchán, Eduardo C., Hernández-Lobato, Daniel, Bielza, Concha and Larrañaga, Pedro, 2018, October. Bayesian optimization of the PC algorithm for learning Gaussian Bayesian networks. In *Conference of the Spanish Association for Artificial Intelligence* (pp. 44-54).
- Garrido-Merchán, Eduardo C. and Albarca-Molina, Alejandro, 2018, November. Suggesting Cooking Recipes Through Simulation and Bayesian Optimization. In *International Conference on Intelligent Data Engineering and Automated Learning*, (pp. 277-284).
- Villacampa-Calvo, Carlos, Zaldivar, Bryan, Eduardo C. Garrido-Merchán, and Hernández-Lobato, Daniel. Multi-class Gaussian Process Classification with Noisy Inputs. Journal of Machine Learning Research, 22(36), 1-52.
- Garrido Merchán, Eduardo C., and Jariego Pérez, Luis. Towards Automatic Bayesian Optimization: A first step involving acquisition functions. Accepted. CAEPIA 2021.
- Asencio-Martín, L., and Garrido-Merchán, E. C. (2021). A similarity measure of Gaussian process predictive distributions. Accepted. CAEPIA 2021.

Work In Progress

- Garrido-Merchán, Eduardo C., and Hernández-Lobato, Daniel. "Parallel Predictive Entropy Search for Multi-objective Bayesian Optimization with Constraints." arXiv preprint arXiv:2004.00601 (2020).

1.5 Summary by Chapters

In this section, we provide an organization of the chapters of this thesis and a summary of every chapter. This organization is shown below:

Chapter 2 provides an introduction to GPs and the expectation propagation algorithm. Both are necessary concepts for the BO methods that we will describe in the following chapters. This chapter reviews the fundamentals of GPs and why they are so interesting for BO. More concretely, we review the most popular kernels, the analysis of the posterior and predictive distribution and how to tune the hyper-parameters of GPs: whether by maximizing the marginal likelihood or by generating samples from the hyper-parameter posterior distribution. Other alternative probabilistic surrogate models are also described briefly. Some of the proposed approaches of this thesis are extensions of an acquisition function called predictive entropy search, that is based on the expectation propagation approximate inference technique. That is why we provide in this chapter an explanation of the expectation propagation algorithm.

Chapter 3 introduces the basics of BO and information theory. BO works with probabilistic models such as GPs and with acquisition functions such as predictive entropy search, that uses information theory. Having studied GPs in Chapter 2, BO can be now understood and it is described in detail. This chapter will also describe the most popular acquisition functions, how information theory can be applied in BO and why BO is useful for the hyper-parameter tuning of machine learning algorithms.

Chapter 4 describes an information-theoretical mechanism that generalizes BO to simultaneously optimize multiple objectives under the presence of several constraints. This algorithm is called predictive entropy search for multi-objective BO with constraints (PESMOC) and it is an extension of the predictive entropy search acquisition function that is described in Chapter 3. The chapter compares the empirical performance of PESMOC with respect to a state-of-the-art approach to constrained multi-objective optimization based on the expected improvement acquisition function. It is also compared with a random search through a set of synthetic, benchmark and real experiments.

Chapter 5 addresses the problem that faces BO when not only one but multiple input points can be evaluated in parallel that has been described in Section 1.3. This chapter introduces an extension of PESMOC called parallel PESMOC (PPESMOC) that adapts to the parallel scenario. PPESMOC builds an acquisition function that assigns a value for each batch of points of the input space. The maximum of this acquisition function corresponds to the set of points that maximizes the expected reduction in the entropy of the Pareto set in each evaluation. Naive adaptations of PESMOC and the method based on expected improvement for the parallel scenario are used as a baseline to compare their performance with PPESMOC. Synthetic,

benchmark and real experiments show how PPESMOC obtains an advantage in most of the considered scenarios. All the mentioned approaches are described in detail in this chapter.

Chapter 6 addresses a transformation that enables standard GPs to deliver better results in problems that contain integer-valued and categorical variables. We can apply BO to problems where we need to optimize functions that contain integer-valued and categorical variables with more guarantees of obtaining a solution with low regret. A critical advantage of this transformation, with respect to other approaches, is that it is compatible with any acquisition function. This transformation makes the uncertainty given by the GPs in certain areas of the space flat. As a consequence, the acquisition function can also be flat in these zones. This phenomenom raises an issue with the optimization of the acquisition function, that must consider the flatness of these areas. We use a one exchange neighbourhood approach to optimize the resultant acquisition function. We test our approach in synthetic and real problems, where we add empirical evidence of the performance of our proposed transformation.

Chapter 7 shows a real problem where BO has been applied with success. In this problem, BO has been used to obtain the optimal parameters of a hybrid Grouping Genetic Algorithm for attribute selection. This genetic algorithm is combined with an Extreme Learning Machine (GGA-ELM) approach for prediction of ocean wave features. Concretely, the significant wave height and the wave energy flux at a goal marine structure facility on the Western Coast of the USA is predicted. This chapter illustrates the experiments where it is shown that BO improves the performance of the GGA-ELM approach. Most importantly, it also outperforms a random search of the hyper-parameter space and the human expert criterion.

Chapter 8 provides a summary of the work done in this thesis. We include the conclusions retrieved by the multiple research lines covered in the chapters. We also illustrate lines for future research.

1.6 How to Read this Thesis

This thesis is the result of several publications concerning BO algorithms applied to broader scenarios than the standard one. The reader may be interested only in one of those scenarios or in all of them. We suggest different itineraries to read this document that we hope satisfy the interests of the reader.

The first two chapters cover the fundamentals of BO, GPs and some concepts related with this thesis such as information theory of the expectation propagation algorithm. If the reader is familiarized with these concepts, the reader can skip the first two chapters and start reading from Chapter 4. On the other hand, if the reader is not familiarized with GPs or BO, the reader is encouraged to study these two chapters.

Chapters 4-6 involve generalizations of standard BO. The reader may only be interested in the generalizations of predictive entropy search, in this case, the reader must read Chapters 4 and 5. If the reader is interested in dealing with integer-valued or categorical variables in GP-BO, the reader must skip those chapters and go to Chapter 6.

Lastly, Chapter 7 illustrates an application of BO to wave energy and it is independent of the other chapters. We recommend the reader to read it if the reader is interested in applications of BO.

1.7 Definitions and Notation

This section addresses the notation that is going to be used in the chapters of this thesis. The thesis introduces analytical expressions that are represented by the following notation:

- x: Scalar.
- $\mathbf{x}$: Vector.
- $\mathbf{X}$: Matrix.
- $\mathcal{X}$: Set.
- x_i: Element i of a vector $\mathbf{x}$.

We sum up the main mathematical objects involved in this thesis in the following list. There are more definitions in the algorithms of this thesis, but, as they are particular to certain sections, are not included in this list.

- $\mathbf{x}^\star$: Optimum of an optimization problem.
- $\mathbf{x}$: Point belonging to the input space of an optimization problem.
- y: Potentially noisy observation of the objective of an optimization problem at a particular input location.
- $\mathbf{y}$: Vector of all the potentially noisy observations of objectives of an optimization problem at a particular input location.
- n: Iteration n of an optimization problem.
- T: Total number of iterations of a BO algorithm.
- $f(\mathbf{x})$: Objective function of an optimization problem.
- $\mathcal{D} = \{(\mathbf{x}_i, y_i) | i = 1, ..., n\}$: Dataset of evaluations y_i of input space points $\mathbf{x}_i$.
- $\alpha(\mathbf{x})$: Acquisition function.
- ϵ: Noise that contaminates the objective function of an optimization problem, typically assumed Gaussian.
- $\mathcal{X}$: Input space of an optimization problem.
- m: Probabilistic surrogate model.
- $\mu(\mathbf{x})$: Mean of a probabilistic surrogate model.
- $\sigma(\mathbf{x})$: Variance of a probabilistic surrogate model.
- $\mathcal{N}(x|\mu, \sigma)$: Gaussian probability density function.
- $\mathbb{E}[\cdot]$: Expectation of a random variable.
- $H(\cdot)$: Entropy of a random variable.
- $k(\cdot, \cdot)$: Covariance function or kernel of a Gaussian process.

- $GP(\cdot, \cdot)$: Gaussian process.
- $\mathbf{I}$: Identity matrix.
- Z: Normalization constant.
- U: Grid over the input space. Sample of a Sobol sequence or uniform.

Chapter 2

Gaussian Processes And Approximate Inference

This chapter presents an overview of Gaussian processes (GPs) and the expectation propagation (EP) algorithm. Bayesian optimization (BO) typically uses a GP as the probabilistic surrogate model of the objective function. Hence, it is important to understand the details of GPs before studying BO. This chapter will describe how GPs are adequate for BO. We give the definition of a GP and include how to compute the GP posterior and predictive distributions. One important component of GPs is the covariance function or kernel. We include a description of the most popular covariance functions used in GPs. Covariance functions make different hypotheses about the objective function. They depend on a set of hyper-parameters. We cover the estimation of hyper-parameters via maximizing the marginal likelihood and by approximately sampling from the GP hyper-parameter posterior distribution. In the case of non-Gaussian likelihoods, the posterior and predictive distribution of the GP is non-Gaussian. In this case, the computation of these distributions is intractable. Approximate inference algorithms provide a solution to this issue. A simple approach is to approximate non-Gaussian factors by un-normalized Gaussian factors. By approximating the likelihood by an un-normalized Gaussian distribution, the posterior and predictive distributions can now be computed. This is precisely the approach followed by Expectation Propagation (EP), an approximate inference algorithm that will be described in detail in this chapter. The BO methods described in subsequent chapters employ EP to approximate the required computations.

2.1 Introduction

This chapter introduces one of the most common probabilistic surrogate models used in BO: a Gaussian process (GP) (Rasmussen, 2003). First, let us assume $y_i = f(\mathbf{x}_i) + \epsilon_i$, where ϵ_i is an additive Gaussian noise $\epsilon_i \sim \mathcal{N}(0, \boldsymbol{\Sigma}\boldsymbol{I})$ such that $\boldsymbol{I}$ is the identity matrix. y_i is the observed value of the objective function $f(\mathbf{x}_i)$ evaluated at the point $\mathbf{x}_i$. The objective function $f(\mathbf{x}_i)$ is corrupted by the additive Gaussian noise ϵ_i so $f(\mathbf{x}_i)$ is a latent function value that we do not observe. For every point $\mathbf{x}$ of the input space $\mathcal{X}$ belonging to the range of the objective function $f(\mathbf{x})$, a GP predicts a Gaussian distribution for the value $f(\mathbf{x})$ associated to $\mathbf{x}$. The potential values for $f(\mathbf{x})$ are specified by the predictive distribution of the GP, $p(f(\mathbf{x})|\mathcal{D})$, where $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., N\}$ is

a dataset of previous observations. As the expected value for $f(\mathbf{x})$ and its uncertainty of every input space value is critical for BO, the computation of the GP predictive distribution will be explained in detail in this chapter. The GP predictive distribution is determined by the GP covariance function, which will introduce the assumptions made about the target function. Covariance functions contain hyper-parameters, that condition the GP predictive distribution. It is hence very important to adequately estimate these hyper-parameters to make an accurate fit of the objective function. We will explain how to properly estimate the model hyper-parameters. An understanding of these concepts is critical to study the proposed methods for BO introduced in this thesis, that are described in the following chapters. GPs are not the only alternative to be used as a surrogate model for BO. Other models can also be used as surrogates. For example, we can take the predictive distribution given by the different trees that a random forest generates (Breiman, 2001). Another alternative is to employ Bayesian deep neural networks, that introduce a prior on every weight of a deep neural network (Springenberg et al., 2016). Each of these models may have different advantages over GPs for particular scenarios. For example, computational cost, that may be critical in some BO scenarios. Hence, a brief description of these models is also covered by this chapter.

An advantage of using GPs for BO is that the GP predictive distribution $p(f(\mathbf{x})|\mathcal{D})$ for the objective function $f(\mathbf{x})$ has an analytic closed-form expression that is easy to compute (Rasmussen, 2003). Hence, it is easy to perform exact Bayesian inference. A consequence of using GPs for BO is that we assume that that the black-box function $f(\mathbf{x})$ is a sample from a GP (Snoek et al., 2012). The GP generates a Gaussian predictive distribution $p(f(\mathbf{x})|\mathcal{D})$ of the objective function $f(\mathbf{x})$. The predictive distribution $p(f(\mathbf{x})|\mathcal{D})$ is used by BO to guide the search (Frazier, 2018). BO focuses on evaluating the objective only on regions of the input space $\mathcal{X}$ that are expected to deliver the most information of the optimum $\mathbf{x}^\star$. These regions can be identified by analyzing the mean and the standard deviation of the predictive distribution of the GP, $p(f(\mathbf{x})|\mathcal{D})$. GPs are also useful for BO for other reasons. GPs are non-parametric models, which are basically models whose number of parameters grows with data (Rasmussen, 2003). Non-parametric models do not assume that the structure of a model is fixed, like in the case of a neural network, parametrized by its weights. In non-parametric models, the model parameters grow in size to accommodate the data complexity. This property makes GPs flexible. Moreover, it is also possible to compute the marginal likelihood of the data $p(\mathcal{D})$ to make Bayesian model selection (Rasmussen, 2003). Although they have a set of hyper-parameters, it is easy to estimate them using type-II maximum-likelihood or a sampling procedure as we will further describe in this chapter (Snoek et al., 2012). One advantage of sampling the hyper-parameters from their posterior distribution is that we do not incur in overfitting with a small amount of data. By sampling the hyper-parameters from their posterior distribution, we are considering various hypotheses about the data, pondered by the hyper-parameter posterior distribution. Type-II maximum-likelihood only considers one hypothesis about the data, which is a limited use of the information available. Given that GPs contain various hyper-parameters and that multiple explanations about data are plausible, considering various hypotheses is a more robust procedure that only just one. This is specially relevant in the presence of a low number of data, where the hyper-parameter posterior distribution may have several modes, that can be interpreted as different hypotheses that explain the data (Rasmussen, 2003). As BO is specifically designed to work in scenarios where the budget of evaluations is small, not incurring in overfitting with a small number of observations make GPs ideal for BO.

Some problems require non-Gaussian likelihoods and the computation of the predictive distribution of a GP $p(f(\mathbf{x})|\mathcal{D})$. In this case, the GP predictive distribution is non-Gaussian, it has no analytic closed-form solution and is infeasible to compute. In this case, a common approach is to approximate the non-Gaussian likelihood factors of the joint distribution of the observed and latent variables of the model by Gaussian distributions. Non-Gaussian likelihood factors can also appear in BO. For example, consider the problem where the only query that we can perform is whether an user has preferred the value of a new input space point $\mathbf{x}$ with respect to an old one $\mathbf{x}'$. Let $g(\mathbf{x})$ be the usual evaluation of the objective function, in this problem, we do not have access to it. This problem is called preferential BO (González et al., 2017). In this problem, we obtain a binary return $y \in \{0, 1\}$ representing which of the two locations $\mathbf{x}, \mathbf{x}'$ is preferred. The reward $f(\mathbf{x}, \mathbf{x}') = g(\mathbf{x}') - g(\mathbf{x})$ cannot be observed directly. These preferences are modelled through a Bernoulli likelihood $p(y = 1|[\mathbf{x}, \mathbf{x}']) = \pi_f([\mathbf{x}, \mathbf{x}'])$ and $p(y = 0|[\mathbf{x}, \mathbf{x}']) = 1 - \pi_f([\mathbf{x}, \mathbf{x}'])$ where $\pi : \mathbb{R} \times \mathbb{R} \to [0, 1]$ can be the logistic function:

$$\pi_f([\mathbf{x}, \mathbf{x}']) = \sigma(f([\mathbf{x}, \mathbf{x}'])) = \frac{1}{1 + \mathrm{e}^{-\mathrm{f}([\mathbf{x},\mathbf{x}'])}}, \tag{2.1}$$

as it has the property that $\pi_f([\mathbf{x}, \mathbf{x}']) = 1 - \pi_f([\mathbf{x}', \mathbf{x}])$. Another common example where non-Gaussian factors are combined with GPs is binary Gaussian process classification (Nickisch and Rasmussen, 2008). In this particular problem, class labels $y_i \in \{-1, 1\}$ are assigned to each data point $\mathbf{x}$. We can assume that these labels are generated through a sign function applied to a function sampled from the GP, $y_i = \mathrm{sign}(f(x_i) + e_i)$, where $e_i \sim \mathcal{N}(0, \sigma^2)$. If we have a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., N\}$ the objective would be to infer $f(\mathbf{x})$. In order to do so, each data point $\mathbf{x}_i$ is assigned one indicator factor $\mathbb{I}(y_i(f_i))$, giving rise to a factorial likelihood $p(\boldsymbol{y}|\boldsymbol{f}) = \prod_{i=1}^{N} \mathbb{I}(y_i f_i)$, where $\mathbf{f} = (f(x_1), ..., f(x_n))^T$ and we assume that there is no noise. The indicator factors $\mathbb{I}(y_i f_i)$ are not Gaussian distributions. If we incorporate these indicator factors into the likelihood, the exact posterior $p(\mathbf{f}|\mathbf{y})$ can no longer be computed analytically. Although, the posterior distribution $p(\mathbf{f}|\mathbf{y})$ can be approximated.

In the previous cases, we can still apply several approaches to approximate the posterior distribution of the GP, which is required for making predictions. Some examples of these approaches are the Laplace approximation (Williams and Barber, 1998), numerical quadrature (Lubinsky and Rabinowitz, 1984), Monte Carlo methods (Hammersley, 2013) or the expectation propagation (EP) algorithm (Minka, 2001b). EP outperforms the Laplace approximation technique in a wide range of problems (Minka, 2001b; Nickisch and Rasmussen, 2008). The Laplace approximation produces a Gaussian distribution that is placed on the mode of the target distribution. This approximation underestimates the variance of the posterior distribution. Moreover, it is not necessarily the best strategy, as the target distribution may have most of its probability mass on a region where its mode does not belong. Numerical quadrature is the perfect match to approximate unidimensional posteriors. However, the problems where we are going to apply approximate inference have a high number of latent variables. If there are a high number of latent variables, numerical quadrature is costly. In particular, the computational cost of numerical quadrature is exponential on the number of latent variables, *i.e.*, the dimension of the integral. In high-dimensional scenarios, numerical quadrature becomes intractable. Monte Carlo methods are also computationally very expensive. They simulate a Markov chain whose stationary distribution coincides with the target distribution. This is a costly procedure. For example, Monte Carlo algorithms need to run the chain for a long time to generate a few independent samples and require a good starting point. Finally,

in another possible technique that could be applied, variational Bayes (Jordan et al., 1999), the exact posterior is lower bounded everywhere, producing approximations that could be innacurate. Also, variational Bayes has to evaluate the expected logarithm of the likelihood factors. If these factors are not smooth, as in the case of indicators or step functions, computing the logarithm of the factors is problematic. EP does not need to compute these logarithms, being more suitable to approximate factors that are not smooth.

The techniques described in the following chapters make use of EP to compute an approximate posterior distribution. Hence, we additionally include a section describing EP in this chapter. We choose EP since we have strong empirical evidence of the good performance that it delivers producing good predictive distributions and marginal likelihood approximations in similar scenarios as the ones described in the following chapters when compared to the other mentioned techniques (Minka, 2001b; Nickisch and Rasmussen, 2008). In particular, EP has been applied with success in the context of BO (Hennig and Schuler, 2012; Hernández-Lobato et al., 2014). It has also shown great performance on approximating posterior distributions in Gaussian process classification problems (Kuss and Rasmussen, 2005; Nickisch and Rasmussen, 2008).

2.2 Gaussian Processes

A Gaussian Process (GP) is a collection of random variables (of potentially infinite size), any finite number of which have (consistent) joint Gaussian distributions. Equivalently, it describes a stochastic process whose values, at any finite number of input locations, have joint Gaussian distributions. We can also think of GPs as defining a distribution over functions where inference takes place directly in the space of functions (Rasmussen, 2003). These processes can be used for regression or classification. In both of these problems, we have a matrix of training data or covariates $\mathbf{X} = (\mathbf{x}_1, ...\mathbf{x}_N)^T$ and a vector of labels, a dependent variable or the variable to predict $\mathbf{y} = (y_1, ..., y_N)^T$. Each row of the $\mathbf{X}$ matrix, $\mathbf{x}_i$, where i is the index of a vector in the matrix, are the characteristics of the i-th data instance that are associated with an observed target value y_i, which can be noisy. We define as a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., N\}$ the set of labeled instances. In regression problems, the variable to predict is real-valued $y_i \in \mathbb{R}$ and in classification problems is categorical. In this section of the thesis we will focus on regression problems, as they are the most common in BO.

A GP can also be defined as a distribution over functions, *i.e.*, we can sample functions from it. So, if we assume that the objective function is a sample of the GP, it is reasonable to use a GP to model the black-box in BO given that it is a non-parametric flexible model, it is easy to estimate its hyper-parameters and it is a robust model that does not overfit with small amounts of data. We illustrate on Figure 2.1 some functions sampled from a GP. These functions have been sampled from a GP model that has not been conditioned on any observation, *i.e.*, these are samples from the a priori distribution. Hence, we can use the GP as a prior when we do not have any observations. Even in this setting, every possible function sampled from the GP prior is a possible hypothesis of the problem. It is easy to generate smooth functions using a GP as prior, even if there are no observations. In this setting, the GP prior specifies a mean (often equal to zero) and a standard deviation for the potential values of $f(\mathbf{x})$. These are displayed, for a one-dimensional objective function $f(x)$, in Figure 2.1, alongside with some samples of $f(\mathbf{x})$.

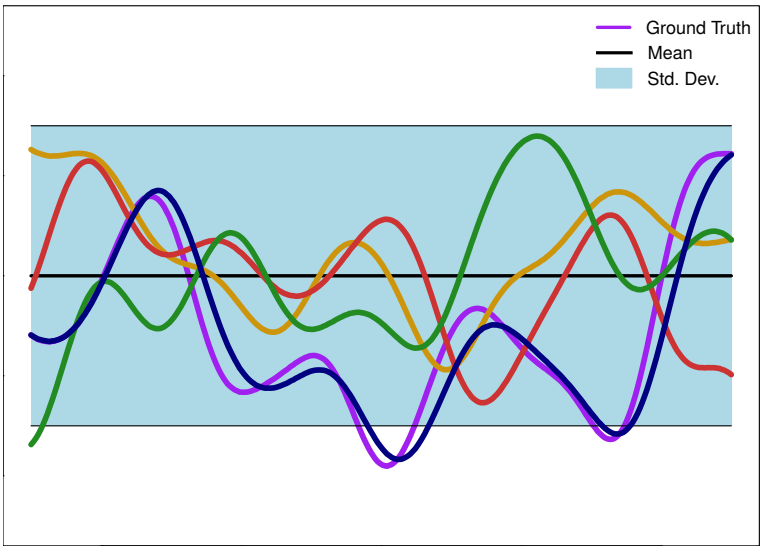

FIGURE 2.1: GP prior, defined by a flat prediction and uniform uncertainty. GPs provide a mean and standard deviation for the ground truth function, drawn in purple. The samples drawn from a GP model belong, with high probability, to that space. We can see that a particular sample, drawn in blue, resembles the ground truth function.

More formally, a GP is fully characterized by a zero mean and a covariance function or kernel $k(\mathbf{x}, \mathbf{x}')$, that is, $f(\mathbf{x}) \sim \mathcal{GP}(\mathbf{0}, k(\mathbf{x}, \mathbf{x}'))$. The covariance function of the GP receives two points as an input, $\mathbf{x}$ and $\mathbf{x}'$. We define the prior mean function $m(\mathbf{x})$ and the covariance function $k(\mathbf{x}, \mathbf{x}')$ that computes the covariance between $f(\mathbf{x})$ and $f(\mathbf{x}')$ has to be evaluated as:

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})]\,, \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]\,. \end{aligned} \tag{2.2}$$

Given a set of observed data $\mathcal{D} = \{(\mathbf{x}_i, y_i) | i = 1, ..., N\}$, where $y_i = f(\mathbf{x}_i) + \epsilon_i$ with ϵ_i some additive Gaussian noise, a GP builds a predictive distribution $p(f(\mathbf{x}^\star)|\mathcal{D})$ for the potential values of $f(\mathbf{x}^\star)$ at a new input point $\mathbf{x}^\star$. This distribution is Gaussian. Namely, $p(f(\mathbf{x}^\star)|\mathcal{D}) = \mathcal{N}(f(\mathbf{x}^\star)|\mu(\mathbf{x}^\star), v(\mathbf{x}^\star))$. We can set a GP prior mean $m(\mathbf{x})$ given our prior knowledge of the problem. Typically, the GP prior mean, $m(\mathbf{x})$, is set to 0. The mean $\mu(\mathbf{x}^\star)$ and variance $v(\mathbf{x}^\star)$ of the predictive distribution $p(f(\mathbf{x}^\star)|\mathcal{D})$ are respectively given by:

$$\mu(\mathbf{x}^\star) = \mathbf{k}_\star^T (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y}\,, \tag{2.3}$$

$$v(\mathbf{x}^\star) = k(\mathbf{x}_\star, \mathbf{x}_\star) - \mathbf{k}_\star^T (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{k}_\star\,, \tag{2.4}$$

where $\mathbf{y} = (y_1, \ldots, y_N)^{\mathrm{T}}$ is a vector with the observations collected so far; σ^2 is the variance of the additive Gaussian noise ϵ_i; $\mathbf{k}_\star = \mathbf{k}(\mathbf{x}_*)$ is a N-dimensional vector with the prior covariances between the test point $f(\mathbf{x}^\star)$ and each of the training points $f(\mathbf{x}_i)$; and $\mathbf{K}$ is a $N \times N$ matrix with the prior covariances among each $f(\mathbf{x}_i)$, for $i = 1, \ldots, N$. Each element $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ of the matrix $\mathbf{K}$ is given by the covariance function between each of the training points $\mathbf{x}_i$ and $\mathbf{x}_j$ where $i, j = 1, ..., N$ and N is the total number of training points. If the GP prior mean, $m(\mathbf{x})$, is not set to 0, the predictive variance

remains unchanged and the mean $\mu(\mathbf{x}^\star)$ is given by:

$$\mu(\mathbf{x}^\star) = m(\mathbf{x}^\star) + \mathbf{k}_\star^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}(\mathbf{y} - m(\mathbf{X}))\,. \tag{2.5}$$

We can see how a GP outputs a predictive distribution $p(f(\mathbf{x})|\mathcal{D})$ of a ground true function conditioned on previously evaluated observations in Figure 2.2. In this figure, we observe how, by conditioning the GP to the observed data, the standard deviation is smaller as more and more data is observed, better predicting the objective function. At the end of the process, the GP conditioned on the observed points will have lower uncertainty about the objective function that the unconditioned GP prior. Hence, on average, samples drawn from the conditioned GP will be more similar to the ground true function than samples drawn from the GP prior. This behavior of the conditioned GP is useful for regression problems, as we can predict the objective function. The

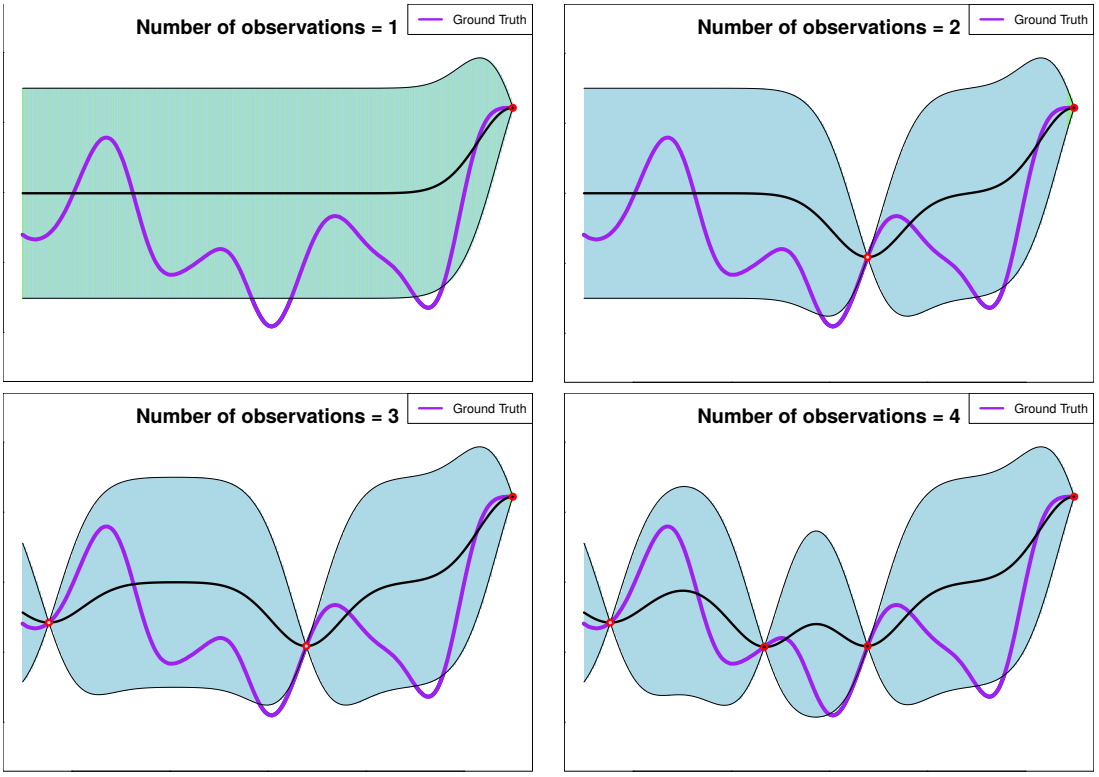

FIGURE 2.2: A GP model fitting a ground truth function (purple). As observations condition the GP model, the mean (black), or expected prediction, of the GP model becomes more similar to the ground truth and the standard deviation, or uncertainty, area of the GP becomes smaller (blue).

particular characteristics assumed for $f(\cdot)$ (*e.g.*, level of smoothness, additive noise, etc.) are specified by the covariance function $k(\mathbf{x}, \mathbf{x}')$ of the GP. The specification of the covariance function implies a distribution over functions. Depending on the chosen covariance function, the GP will be a good fit for the objective function.

2.3 Covariance Functions

Covariance functions encode our assumptions about the function that we wish to learn (Rasmussen, 2003). They can be seen as a notion of similarity between data points

$\mathbf{X} = (\mathbf{x}_1, ...\mathbf{x}_N)^T$. The Gram matrix $\mathbf{K}$ whose entries are $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is the covariance matrix that contains all the notions of similarity of a set of points $\mathbf{X} = (\mathbf{x}_1, ...\mathbf{x}_N)^T$. Not all functions are valid covariance functions. For a covariance function to be valid, its corresponding Gram matrix must be positive semidefinite (PSD). A symmetric matrix is PSD if and only if all of its eigenvalues are non-negative.

Depending on the analytical expression of the covariance function, it has different properties. A stationary covariance function is a function of $\mathbf{x} - \mathbf{x}'$. Such a function is invariant to translations in the input space. If, also, the covariance function is a function of $|\mathbf{x} - \mathbf{x}'|$, it is called isotropic. In isotropic covariance functions, the direction of the deviation is of no importance, it only depends on the distance of the covariance function arguments. Lastly, if a covariance function depends only on the product of the input space points $\mathbf{x} \cdot \mathbf{x}'$ it is a dot product covariance function. Dot product covariance functions are invariant to a rotation of the coordinates about the origin, but not translations.

Common examples of covariance functions used by GPs are the squared exponential or the Matérn function. Deciding which covariance function to use is a design choice that can be critical for the result of the GP application. Every covariance function makes different assumptions about the target function. Understanding the basic properties and assumptions made by the most common GP covariance functions is, hence, important. We provide a review of these covariance functions:

- **Squared exponential covariance function**: The most popular covariance function for GPs is known as the squared exponential covariance function. This function $k(\cdot, \cdot)$ is formulated as:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{r^2}{2\ell^2}\right) + \sigma_n^2 \delta_{pq}\,, \tag{2.6}$$

  where r is the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}'$. ℓ is a hyper-parameter known as length-scale, which controls the smoothness of the functions generated from the GP. Most of the times a different length scale ℓ_j is used for each dimension j. σ_f^2 is the amplitude parameter or signal variance, which controls the range of variability of the GP samples. Finally, $\sigma_n^2 \delta_{pq}$ is the noise variance that applies when the covariance function is computed for the same point $k(\mathbf{x}, \mathbf{x})$. This condition is modelled by the delta function δ_{pq}. If there is noise in the covariance function, we can eliminate it from the likelihood function. In this case, we would observe $y_i(\mathbf{x}_i) = f_i(\mathbf{x}_i)$, and the noise would be included in $f_i(\mathbf{x}_i)$ via $\sigma_n^2 \delta_{pq}$. Note that $k(\cdot, \cdot)$ only depends on r. This particular covariance function and others that share this property are known as *radial basis functions* (RBFs). This covariance function is infinitely differentiable, which means that the GP is very smooth, as we can see in Figure 2.3. We can observe in Figure 2.3 different shaped functions sampled from a GP with the squared exponential covariance function with various length-scale values ℓ, for a single dimension. We can see in Figure 2.3 how, by augmenting the lengthscale ℓ, the samples obtained from a GP with a squared exponential covariance function are smoother. However, when the lengthscale ℓ is reduced, these samples become rough. As we do not know the actual level of smoothnes of the target function, estimating this hyper-parameter is critical for the GP model to provide an accurate predictive distribution. We later explain, in this section, different methods for estimating the hyper-parameters of the covariance function.

- **Matérn covariance function**: The squared exponential covariance function is critized for its strong smoothness assumptions, which are unrealistic for modelling

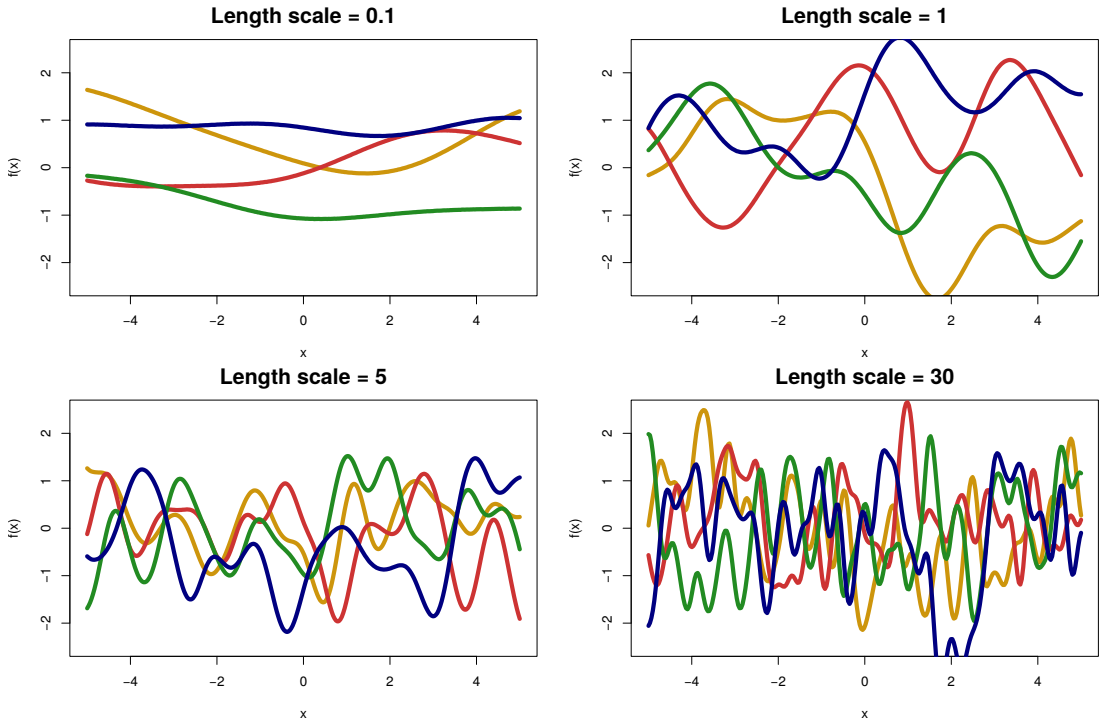

FIGURE 2.3: Different functions drawn from GP priors with different value of the length scale hyper-parameter. We can see how their shape is dependant on the value of the length scale hyper-parameter. Higher values of the length scale hyper-parameter imply smoother functions sampled by the GP model.

many physical processes (Stein, 2012). An alternative to the squared exponential covariance function, which does not make such strong smoothness assumptions, is the Matérn class of covariance functions. This covariance function is usually employed in the context of BO. The Matérn covariance function analytical expression contains, among others, a parameter defined as ν. In contrast with the hyperparameters of the squared exponential covariance function, like the amplitude parameter σ_f^2, this parameter is fixed and cannot be learnt from data. The Matérn class of covariance functions is given by the following expression:

$$k_{\text{Matern}}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu r}}{\ell}\right)^{\nu} \mathrm{K}_{\nu}\left(\frac{\sqrt{2\nu r}}{\ell}\right), \tag{2.7}$$

where $\Gamma(\cdot)$ is the Gamma function (Artin, 2015), r is the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}'$ and K_ν is a modified Bessel function (Abramowitz and Stegun, 1965). Depending on the value of the parameter ν, the analytical expression of the Matérn covariance function varies. The most popular values for the machine learning community for the ν parameter are $\nu = 3/2$ and $\nu = 5/2$. The following expressions correspond to the analytical expression of the Matérn covariance function with $\nu = 3/2$ and $\nu = 5/2$:

$$k_{\nu=3/2}(r) = \sigma_f^2\left(1 + \frac{\sqrt{3}r}{\ell}\right)\exp\left(-\frac{\sqrt{3}r}{\ell}\right), \tag{2.8}$$

$$k_{\nu=5/2}(r) = \sigma_f^2\left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right)\exp\left(-\frac{\sqrt{5}r}{\ell}\right), \tag{2.9}$$

where σ_f^2 is the signal variance parameter, also present in the squared exponential covariance function. The ν parameter is related to the number of times that the GP samples can be differentiated. Hence, higher values of the ν parameter make the GP sample smoother functions. A related result is that, as $\nu \to \infty$, the Mátern covariance function converges to the squared exponential covariance function, which is infinitely differentiable. We can observe this effect on the Matérn covariance function for different values of its hyper-parameters in Figure 2.4. Higher values of ν make the GP sample smoother functions and viceversa.

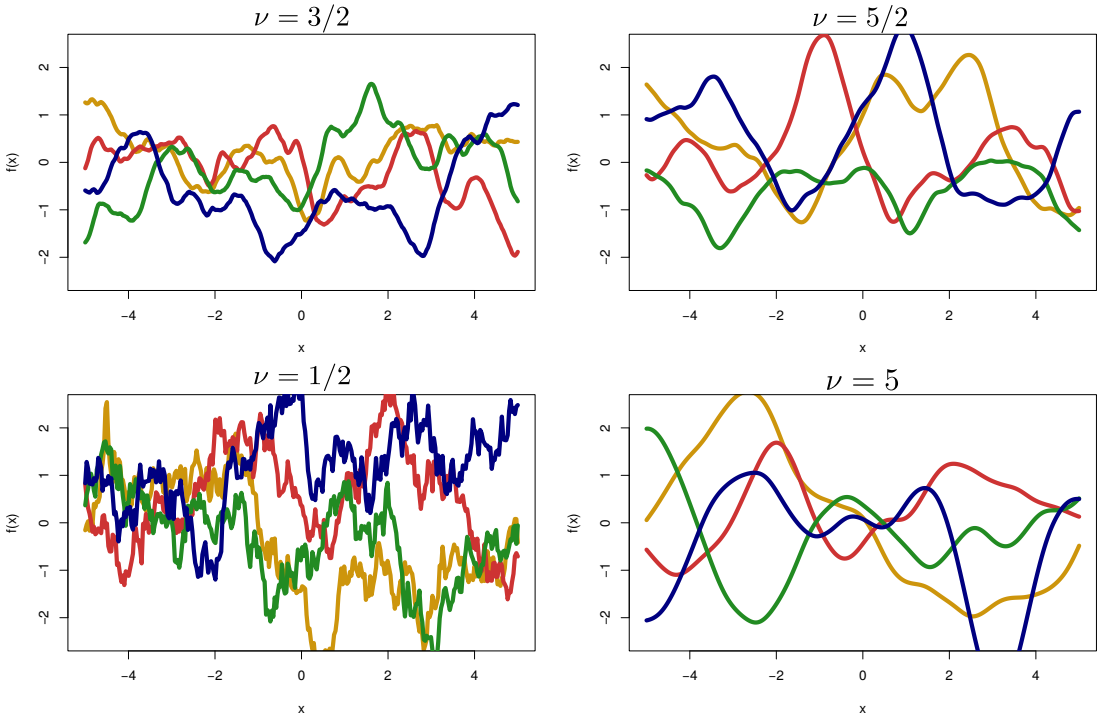

FIGURE 2.4: Different functions drawn from GP priors with different value of the ν hyper-parameter. We can see how their shape is dependant on the value of the ν hyper-parameter. Higher values of ν imply smoother functions sampled from the GP model.

Covariance functions can also be combined in different ways. We can perform addition, multiplication or exponentiation of covariance functions (Rasmussen, 2003). The only requisite to test whether a covariance function is valid is that the covariance matrix $\mathbf{K}$ must be positive semidefinite. We can even generate a covariance function from different spaces $\mathcal{X}_1$ and $\mathcal{X}_2$ by the tensor product $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$. As we have seen, we can sample very different shaped functions by using GPs just by varying the hyper-parameters. Hence, it is critical to estimate accurate hyper-parameter values to fit an unknown function with a GP.

2.4 Hyper-Parameter Estimation

A GP model has a set of hyper-parameters $\boldsymbol{\theta}$ that can be adjusted to better fit the data $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., N\}$. These include the variance of the additive Gaussian noise σ^2 and any potential hyper-parameter of the covariance function $k(\cdot, \cdot)$. These can be, *e.g.*,

the amplitude and the length-scales. Two popular approaches to estimate the values for these hyper-parameters w.r.t the data are: maximizing the log marginal likelihood and approximately computing a posterior distribution for the hyper-parameters.

2.4.1 Maximizing the Log Marginal Likelihood

We can find point estimates for the hyper-parameters of the GP through optimizing the log marginal likelihood. The marginal likelihood, or evidence, is related to the marginalization over the latent function values $\mathbf{f} = (f_1, ..., f_N)^T$ of the data points. These latent function values vary according to the chosen values of the GP hyper-parameters. The marginal likelihood is also called evidence as it can be seen as the probability of the observed data given the model. Intuitively, if this quantity is bigger for one model, it means that it explains the data better than another model. Comparing models through this quantity is what Bayesian model comparison performs. Let $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., N\}$ be the observed data and $\mathcal{M}$ a set of models whose performance we want to compare and m a model belonging to that set. Let $\boldsymbol{\theta}$ be the set of hyper-parameters of a model m. If we use a uniform prior over models $p(m) \approx 1$, then, the model that better fits the data according to Bayesian model selection is the one that maximizes the likelihood of the data given that model (Murphy, 2012):

$$p(\mathcal{D}|m) = \int p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}|m)d\boldsymbol{\theta}\,. \tag{2.10}$$

Section 5.3.2 of Murphy (2012) gives details about the computation of $p(\mathcal{D}|\boldsymbol{\theta})$ and $p(\boldsymbol{\theta}|m)$. The trick to make this computation easy is to choose a conjugate prior for the hyper-parameters $p(\boldsymbol{\theta}|m)$. A conjugate prior $p(\boldsymbol{\theta}|m)$ for the likelihood function $p(\mathcal{D}|\boldsymbol{\theta})$ is a probability distribution that makes the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ be in the same probability distribution family after applying Bayes theorem. For example, the exponential family is conjugate to itself. Models with a low number of parameters, that incur in underfitting, will not explain the data well and will have a low marginal likelihood value. As this quantity integrates out the parameters $\boldsymbol{\theta}$ of the models, it also avoids overfitting. Having a higher number of parameters does not maximize the marginal likelihood, as bad values for these parameters penalize the evidence. This is known as Bayesian Occam's razor (MacKay, 1995), in an analogy to the Occam's razor. Occam's razor states that given a set of models that accurately explain the data we should pick the simplest one of them.

Now that we have explained what the evidence is, let us define how to compute the log marginal likelihood of a GP to estimate its hyper-parameters. The log marginal likelihood can also be seen as the integral of the likelihood, $p(\mathbf{y}|\mathbf{f}, \mathbf{X})$, of the observed data $\mathbf{y}$ given the input data $\mathbf{X} = (\mathbf{x}_1, ...\mathbf{x}_N)^T$ and the latent function values of a GP $\mathbf{f} = (f_1, ..., f_N)^T$ times the prior of the latent function values given the data $p(\mathbf{f}|\mathbf{X})$. Recall that the latent function values of the GP $\mathbf{f}$ vary with every different value of the hyper-parameters $\boldsymbol{\theta}$ of the GP. Hence, marginalizing the latent function values of the GP, $p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{f}, \mathbf{X})p(\mathbf{f}|\mathbf{X})d\mathbf{f}$, is equivalent to marginalizing the parameters of the GP, that is what Eq. (2.10) does. Both the likelihood $p(\mathbf{y}|\mathbf{f}, \mathbf{X})$ and the prior $p(\mathbf{f}|\mathbf{X})$ are Gaussian distributions. As the product of two Gaussian distributions gives another un-normalized Gaussian distribution, we can compute a closed-form analytical expression for the marginal likelihood of the GP. The normalizing constant can also be computed analytically, see Eq. (A.8) of Rasmussen (2003) for details. The integration yields the

following log marginal likelihood:

$$\log p(\mathbf{y}|\boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^T(\mathbf{K}+\sigma^2\mathbf{I})^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}+\sigma^2\mathbf{I}| - \frac{N}{2}\log 2\pi\,. \tag{2.11}$$

The previous analytical expression can be optimized to obtain a point estimate $\boldsymbol{\theta}^\star$ for the hyper-parameters $\boldsymbol{\theta}$. The three terms of the marginal likelihood can be interpreted. The first term, that includes the observed targets $\mathbf{y}$, is the data-fit. The data-fit is bigger if the GP performs an accurate prediction of the objective function values and viceversa. The second is the complexity penalty on the covariance function. This log determinant term gives preference to Gaussian distributions with high variability, as we will see through an example. The third term is a normalization constant. There is a tradeoff between the first two terms. Consider varying the lengthscale of a squared exponential covariance function with fixed amplitude parameter in a one dimensional problem. Good fits produced by a small lengthscale make $\mathbf{y}^T(\mathbf{K}+\sigma^2\mathbf{I})^{-1}\mathbf{y}$ small but produce a high model complexity. Recall that small lengthscales make most points not near to any others. This implies that the Gram matrix $\mathbf{K}$ will be almost diagonal. Hence, the log determinant $|\mathbf{K}+\sigma^2\mathbf{I}|$ will be large. For long lengthscales we will have the opposite effect, poor fits but low model complexity as all the entries of the Gram matrix $\mathbf{K}$ will be almost 1, making the log determinant $|\mathbf{K}+\sigma^2\mathbf{I}|$ smaller.

If we have enough data, maximizing the log marginal likelihood with respect to the hyper-parameters of the covariance function will be a good idea. Although, when we do not have enough data, we may incur in overfitting if we use this approach. For example, we may underestimate the level of noise and have length-scales with high value. As the number of observations N increases, estimating the hyper-parameters $\boldsymbol{\theta}$ of a GP through the optimization of the log marginal likelihood becomes less prone to overfitting.

In order to optimize the log marginal likelihood expression, we can use a gradient-based optimizer like L-BFGS (Zhu et al., 1997). The L-BFGS quasi-Newton method is an approximation of the BFGS algorithm that uses a limited amount of memory. It uses an estimate of the inverse Hessian matrix of the objective function to optimize. Instead of storing the full inverse Hessian as BFGS does, L-BFGS only stores some updates of the positions visited in the optimization process and the gradient of the objective function in that positions. As the mentioned approximations produce a linear memory complexity, this technique is well suited for optimization problems with many variables. If we extract the gradient from the log marginal likelihood, L-BFGS can compute a local optimum for the value of the hyper-parameters. The gradient $\nabla_{\boldsymbol{\theta}}\log(\mathbf{y}|\mathbf{X},\theta)$ has an analytical closed-form expression, computed through matrix derivatives rules. Every partial derivative $\partial/\partial\theta_j\log(\mathbf{y}|\mathbf{X},\theta)$ of the gradient is given by the following expression (Rasmussen, 2003):

$$\begin{aligned}\frac{\partial}{\partial\theta_j}\log(\mathbf{y}|\mathbf{X},\theta) &= \frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\theta_j}\mathbf{K}^{-1}\mathbf{y} - \frac{1}{2}\mathrm{tr}(\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\theta_j}) \\ &= \frac{1}{2}\mathrm{tr}((\boldsymbol{\alpha}\boldsymbol{\alpha}^T - \mathbf{K}^{-1})\frac{\partial\mathbf{K}}{\partial\theta_j})\,,\end{aligned} \tag{2.12}$$

where $\boldsymbol{\alpha} = K^{-1}\mathbf{y}$. The complexity of the previous expression is cubic in the number of observations N as it involves the inverse of the covariance matrix K. The marginal likelihood can suffer for multiple local optima. This does not need to be a problem, as the different optima can be seen as different interpretations of the data.

2.4.2 Slice Sampling from the Posterior Distribution

As we have seen in the previous section, we can incur in over-fitting by finding a point estimate $\boldsymbol{\theta}^\star$ of the hyper-parameters $\boldsymbol{\theta}$ through the maximization of the log marginal likelihood. Consider the trivial example of a GP conditioned to one point $(\mathbf{x}, y)$. One point is not enough to give accurate information about the shape of the objective function $f(\mathbf{x})$ and much less to discriminate which hyper-parameter values generate accurate predictive distributions $p(f(\mathbf{x})|\boldsymbol{\theta}, \mathcal{D})$ by only having that information. If we estimate the hyper-parameters of the GP by maximizing the marginal likelihood, giving a point estimate of the hyper-parameters $\boldsymbol{\theta}$, we will be commiting overfitting. When we do not have much data, a more reasonable approach is to consider the hyper-parameters posterior distribution $p(\boldsymbol{\theta}|\mathbf{y})$ instead of using a single hyper-parameter value $\boldsymbol{\theta}^\star$. This distribution is intractable because we cannot compute the normalization constant $p(\mathbf{y})$ of it. Let us explain it more in detail. If we compute the posterior distribution using Bayes theorem $p(\boldsymbol{\theta}|\mathbf{y}) = p(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathbf{y})$, the normalization constant of this distribution, $p(\mathbf{y}) = \int p(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$, involves integrating the marginal likelihood of the observed data $p(\mathbf{y}|\boldsymbol{\theta})$ given all the values of the hyper-parameters pondered by the prior distribution for them $p(\boldsymbol{\theta})$. This operation is intractable, as the space of hyper-parameters is high-dimensional. Given the impossibility of fully computing the posterior distribution of the hyper-parameters, an alternative that has shown good empirical results is to compute an approximate posterior distribution of the hyper-parameters $\boldsymbol{\theta}$. We can perform this computation using slice sampling (Neal, 2003) (Snoek et al., 2012). Slice sampling performs a fully-Bayesian treatment of the hyper-parameters $\boldsymbol{\theta}$. Under slice sampling, we generate the predictive distribution $p(f(\mathbf{x}^\star)|\boldsymbol{\theta}, \mathcal{D})$ for a new point $(\mathbf{x}^\star, f(\mathbf{x}^\star))$ by averaging across S samples of the hyper-parameters $\boldsymbol{\theta}$:

$$p(f(\mathbf{x}^\star)|\mathcal{D}, \boldsymbol{\theta}) = \frac{1}{S}\sum_{s=1}^{S} p(f(\mathbf{x}^\star)|\mathcal{D}, \boldsymbol{\theta}_s)\,, \tag{2.13}$$

where $\boldsymbol{\theta}_s$ is the s sample of the hyper-parameters. Having approximated this posterior distribution $p(f^\star|\mathbf{x}^\star, \mathcal{D})$, the computation of the predictive distribution $p(f(\mathbf{x}^\star)|\mathcal{D})$ of an observation $y^\star$ that corresponds to a new data point $\mathbf{x}^\star$ is:

$$p(y^\star|\mathbf{x}^\star, \mathcal{D}) = \int p(y^\star|f(\mathbf{x}^\star))p(f(\mathbf{x}^\star)|\mathcal{D})df(\mathbf{x}^\star)\,, \tag{2.14}$$

where $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., N\}$ is the dataset of observed data points and $p(f(\mathbf{x}^\star)|\mathcal{D})$ is the posterior distribution of the latent function value $f(\mathbf{x}^\star)$ of the new point $\mathbf{x}^\star$ given the observed data $\mathcal{D}$. For a low number of samples, $S = 10$ for example, the process of generating these samples and computing the final predictive distribution $p(y^\star|\mathbf{x}^\star, \mathcal{D})$ takes only a few seconds at most. This time can be considered negligible compared to the cost of evaluating the actual black-box function $f(\cdot)$.

As we have seen, we approximately generate the S hyper-parameter samples $\boldsymbol{\theta}_s$ for the predictive distribution $p(f(\mathbf{x}^\star)|\mathcal{D})$ using slice sampling (Neal, 2003). Slice sampling generates the S samples by executing a Markov chain a finite number of steps whose stationary distribution coincides with the target distribution $p(f(\mathbf{x}^\star)|\mathcal{D})$ (Bishop, 2006). We define a stationary distribution as the distribution where the chain is located after a sufficiently long number of steps, not changing any longer. A Markov chain is a stochastic model that describes a sequence of possible events $\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_n$ in which the probability of an event $p(\boldsymbol{\theta}_t)$ depends only on the previous event $p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})$, where t is the index

of the chain. If the sampling technique makes the stationary distribution computed by the Markov chain match the target distribution, then, we can use the samples from the Markov chain to approximate the target distribution.

Sampling methods that use Markov chains are called Markov chain Monte Carlo algorithms. These techniques use a proposal distribution $q(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})$ to sample a new state from a previous one combined with a criterion that states whether the new sample $\boldsymbol{\theta}_t$ should be rejected (Bishop, 2006). This criterion is added to the Markov chain Monte Carlo method to guarantee that the stationary distribution of the Markov chain matches the objective distribution. One example of a Markov chain Monte Carlo algorithm is Metropolis Hastings. Metropolis Hastings has a step-size parameter (Chib and Greenberg, 1995), that controls the size of the proposal distribution, *i.e.*, the average size between samples. A small step-size makes difficult to sample from all the area of the target distribution. It can even make the stationary distribution to match only a part of the target distribution. This can happen if, for example, the target distribution is multi modal and the size of the steps generate a the stationary distribution that only focuses in one mode. This issue happens as the steps are not long enough to explore the other mode. However, a large step-size makes the algorithm slow as it incurs in a sample high rejection rate.

Markov chain Monte Carlo algorithms usually depend on a set of parameters. Depending on the values of that parameters, the sampling method may compute a stationary distribution that matches the target distribution or fail to do so. Having to parametrize the sampling algorithm is an undesirable feature. Slice sampling was proposed driven by the motivation of having a parameter-free sampling algorithm (Neal, 2003). Slice sampling, in contrast to other algorithms, such as Metropolis Hastings, does not have any parameter, avoiding the issue of the Metropolis Hastings step-size parameter described in the previous paragraph. In slice sampling, the step-size is automatically configured according to the target distribution.

Let $p(x)$ be an univariate distribution and $\hat{p}(x)$ the target un-normalized distribution. The goal in slice sampling is to generate approximate samples from its normalized version $p(x)$. In order to do so, the first step of slice sampling is to choose at random an initial point x_1 such that $p(x_1) > 0$. Let u be an additional variable. The algorithm samples u uniformly such that $0 \leq u \leq \hat{p}(x_1)$. The trick of slice sampling lies in augmenting the variable x with a variable u and drawing samples from the joint (x, u) space. This joint probability distribution $\hat{p}(x, u)$ has uniform probability in the areas lying in the interval $0 \geq u \geq \hat{p}(x)$, *i.e.*, every value has probability equal to $\frac{1}{Z_p}$, where $Z_p = \int \hat{p}(x)dx$ is the normalization constant of the un-normalized target distribution. By considering that the marginal distribution over x satisfies $p(x) = \int p(x, u)du$, we can just sample from $p(x)$ by sampling from the joint distribution $p(x, u)$ and then ignoring the u values. This process is carried out by alternately sampling x and u. That is the reason why, in the first place, this algorithm samples u uniformly such that $0 \leq u \leq \hat{p}(x_t)$, where x_t is the sampled point at iteration t. The value (x_t, u) defines a slice to the distribution, an horizontal line that cuts the un-normalized target distribution at location x_t and height u of the joint (x, u) space. This slice is defined in the region of the input space where $\hat{p}(x) > u$. Figure 2.5 illustrates this process. We have sliced the target un-normalized distribution $\hat{p}(x)$ plotted on blue with a horizontal line plotted on pink at the height given by u, which is the slice. The size of the slice is delimited by two values: $x_{\min}$ and $x_{\max}$, such that x_t is contained in the interval defined by these points $x_{\min} < x_t < x_{\max}$. These values are used to sample a new point x from the un-normalized distribution. The new point will hence be contained in the region defined by: $x_{\min} \leq x \leq x_{\max}$. The purpose of this region is to

allow large moves in the x space, but without getting to far from the slice. Getting too far of the slice would make the sample inefficient. The slice is useful as it tells us that the area above it, $\hat{p}(x) > u$, is a high probability mass area of the target un-normalized distribution. Since we want to approximate the target distribution through samples, we are interested in sampling with more probability the areas in the x where the slices leave probability above. It is critical hence to accurately compute $x_{\min}$ and $x_{\max}$ since new points are uniformly sampled from the interval that these values form. In order to compute these values, slice sampling chooses a width w. Then, starting from x_t, it adds and substracts w, testing whether $(x_t - w, u)$ and $(x_t + w, u)$ belong to the slice, *i.e.*, are locations y where $u < p(y)$. If these positions leave probability mass above it, the width is applied again. If not, the position is either $x_{\min}$ or $x_{\max}$. By doing this process, the interval $[x_{\min}, x_{\max}]$ covers a high probability mass area of the un-normalized target distribution. We represent in Figure 2.5 how a new point is sampled from a previous one. Then, it samples x_{t+1} from $x_t - w, x_t + w$, checking if x_{t+1} belongs to the slice and adapting if it does not.

Slice sampling can be applied to multivariate distributions by repeatedly sampling each variable in turn. In order to retrieve an accurate representation of the target distribution by generating samples, it is critical to perform a big number of iterations. As we have seen by how slice sampling works, the samples are not independent, *i.e.*, are correlated. Hence, the effective number of generated samples is going to be lower than the number of iterations. If the distribution is complex and multi-modal, representing it through slice sampling will demand a big number of iterations. A summary of the operations performed on slice sampling in shown in Algorithm 1.

Input: Number of samples S, target distribution $p(\cdot)$, un-normalized target distribution $\hat{p}(\cdot)$.
1: Choose at random an initial point x_1 such that $p(x_1) > 0$.
for $t = 1, 2, 3, \ldots, S$ **do**
  2: Sample $u \in [0, \hat{p}(x_t)]$.
  3: Compute an slice of the distribution $[(x_t - w, u), (x_t + w, u)]$ through the width w.
  4: Uniformly sample a new point x_{t+1} from the slice $[(x_t - w, u), (x_t + w, u)]$ computed in Step 3.
end
Result: S Samples from the distribution.

Algorithm 1: Slice sampling steps to sample from a target distribution $p(\cdot)$.

We visually compare the described methods of estimating the GP hyper-parameters $\boldsymbol{\theta}$. We approximate the marginal posterior distribution over the hyper-parameters $p(f(\mathbf{x}^\star)|\mathcal{D}) = (1/S)\sum_{s=1}^{S} p(f(\mathbf{x}^\star)|\mathcal{D}, \boldsymbol{\theta}_s)$ by drawing S samples from the posterior distribution of the hyper-parameters $p(\boldsymbol{\theta}|\mathbf{y})$ using slice sampling. As we can see from $p(\boldsymbol{\theta}|\mathcal{D})$, the probability for the values of the hyper-parameters is a function of the observed data $\mathcal{D}$. Depending on the sampled values of the hyper-parameters $\boldsymbol{\theta}$, predictions $p(y^\star|\mathbf{x}^\star, \mathcal{D}) = \int p(y^\star|f(\mathbf{x}^\star))p(f(\mathbf{x}^\star)|\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{y})df(\mathbf{x}^\star)d\boldsymbol{\theta}$ change. The predictive distribution is an average of the predictions done by GPs whose hyper-parameters $\boldsymbol{\theta}$ are drawn by slice sampling of the posterior distribution of the hyper-parameters $p(\boldsymbol{\theta}|\mathbf{y})$. It approximates the potentially infinite number of predictions of GPs hyper-parametrized by all the values of the hyper-parameter space $\boldsymbol{\Theta}$. The inmediate consequence of working

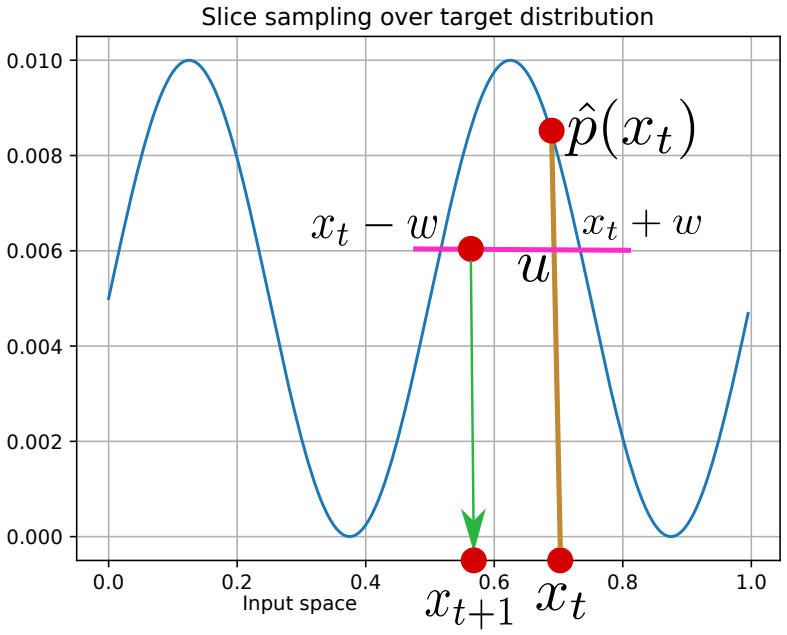

FIGURE 2.5: Sampling of a new point from a previous one through a slice. Points are represented by red dots. The un-normalized distribution is plotted in blue color. The defined slice that we use to sample the next point is plotted in pink color.

with the posterior distribution of the hyper-parameters instead of working with a point estimate of them is that the uncertainty given by that distribution is added to the predictive distribution of the GP $p(f(\mathbf{x})|\mathcal{D})$. This uncertainty makes the GP more robust, so we expect the GP to provide better results by considering the posterior distribution of the hyper-parameters rather than just one point estimate. We visually compare the described methods of estimating the GP hyper-parameters $\boldsymbol{\theta}$ in Figure 2.6.

2.5 Other Surrogate Models

Due to its commented properties and flexibility, GPs have been the most popular probabilistic surrogate models used in BO (Frazier, 2018; Shahriari et al., 2015). Nevertheless, GPs cannot provide accurate posterior and predictive distributions of all the objective functions that we can face in different situations. In some cases, GPs may not be the best option to choose. Other surrogate models may provide better predictions and uncertainty estimates for those objective functions. In practice, GPs provide great results, but cases where GPs have difficulties modelling objective functions have also been shown. An example is the class of non-stationary functions (Snoek et al., 2014). If a function has different variability across its range, we say that it is not-stationary. The concept of stationarity comes from time series. We say that a time series is stationary if its statistical properties do not change over time. The analogous concept in GP-BO is that the objective function variability does not change in its range. Let us provide an example: the sine one dimensional function is stationary but the exponential function is not. Another drawback of GPs is its cubic complexity over the number of observations $\mathcal{O}(N^3)$, where N is the number of observations, that involves computing an inverse matrix. Recall from previous

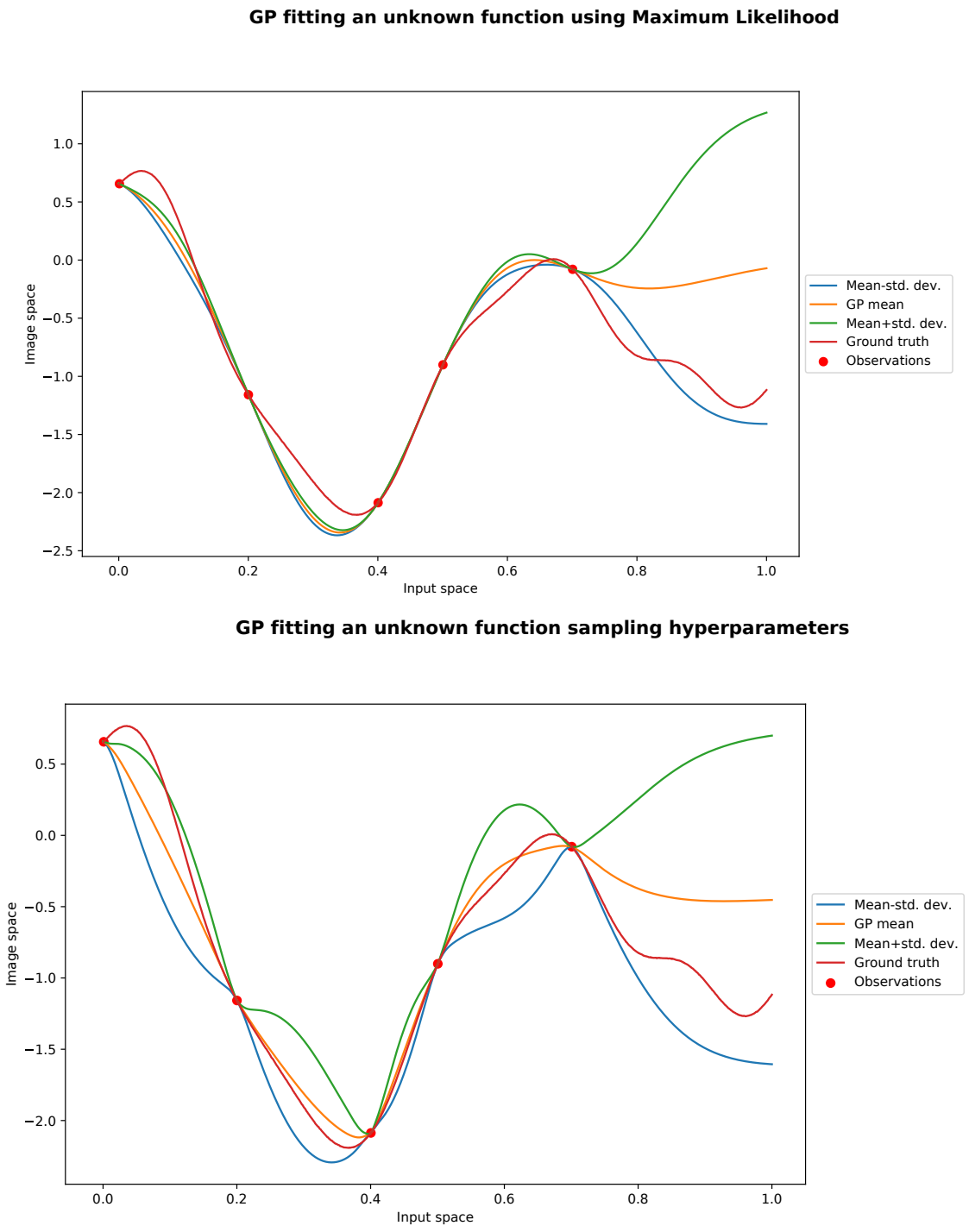

FIGURE 2.6: GP posterior distributions optimizing the log marginal likelihood (top) and sampling from the posterior distribution of the hyper-parameters. (down)

equations that we need to compute the following expression for the mean of the GP: $\mu(\mathbf{x}^\star) = \mathbf{k}_\star^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}$. As we can observe, this expression includes the computation of the inverse of the sum of the Gram matrix on training data plus an identity matrix multiplied by a constant $(\mathbf{K} + \sigma^2\mathbf{I})^{-1}$. This matrix also needs to be inverted for the predictive distribution variance computation $k(\mathbf{x}, \mathbf{x}) - \mathbf{k}_*^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}_*$. In practice, inverting this matrix with more than 1000 observations is problematic. In the described scenarios, an alternative choice to fit the data is to choose another probabilistic model. This section exposes a list with model alternatives to GPs.

- **Random forests**: These models are a combination, or ensemble, of decision trees (Breiman, 2001). A decision tree predicts the value of an instance through a sequential decision making process corresponding to the traversal of a binary tree. In this process, each node represents a classification of the instance to be predicted into two categories. Depending on the category of the instance classified by the node, the tree sends this instance to other nodes, that performs other classifications until the instance reaches a leaf node, that represents the label, for classification problems, or value, for regression problems, predicted for the instance. The structure of the tree and the questions performed by the nodes can be learnt via the CART algorithm (Breiman et al., 1984). Each tree classifier is trained on a bootstrap sample of the data.

  Random forests (RFs) are both used for classification and regression purposes. The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them. It depends on the correlation between them because when considering different predictors, if they have variability on their predictions and the errors occur in different space areas they are cancelled because each one of them predicts a different result in each area. This fact happens as each tree is trained on a different bootstrap sample of the training set. The errors are cancelled as the prediction of the ensemble is computed by an average of the predictions of the trees. Because of this variability in the predictions, using different predictors make ensembles report better results than decision trees. But an ensemble of trees is useless if all the classifiers produce similar predictions, *i.e.*, the ensemble does not have variability in its predictions and the errors are dependent on each other. In order to introduce variability on the predictions, randomization is introduced in the tree growing algorithm. The randomization can be introduced using different techniques. For example, we can randomly choose the best split among the s best of them where s is usually fixed to be the square root of the number of attributes (Breiman, 2001). Random forests have other hyper-parameters that condition its predictions. Some examples include the number of trees, the number of variables to randomly choose in each split or the maximum depth of each tree, among others. More trees and number of variables considered in each split often lead to a better generalization error, as different criteria is added into the prediction. Random forests usually consider the square root of the number of attributes as the number of variables to randomly choose in each split. That value usually work well and it is not fixed. Considering the maximum depth of the trees, a high value can incur in overfitting if the forest does not have variability in its predictions and a low value may produce underfitting. Also, a high value of this hyper-parameter implies a higher computational cost in the prediction of a new instance, as we will further explain.

Let us define the prediction of a tree as $f_i(\mathbf{x})$. The prediction of RFs is done by the average of the outputs. RFs are computationally cheaper than GPs. The computational cost at test time for a RF is $\mathcal{O}(TD)$, where T is the ensemble size, *i.e.*, the number of trees considered in the random forest, and D is the maximum depth of a tree, *i.e.*, the maximum number of splits that a tree can have in the RF (Solé et al., 2014). One can obtain an approximate Gaussian predictive distribution with mean given by $\hat{f}(\mathbf{x}) = 1/T \sum_{i=1}^{T} f_i(\mathbf{x})$ and variance given by $\sigma^2 = (\sum_{i=1}^{T}(f_i(\mathbf{x}') - \hat{f})^2)/(T-1)$. We can see a graphical representation of random forests on Figure 2.7

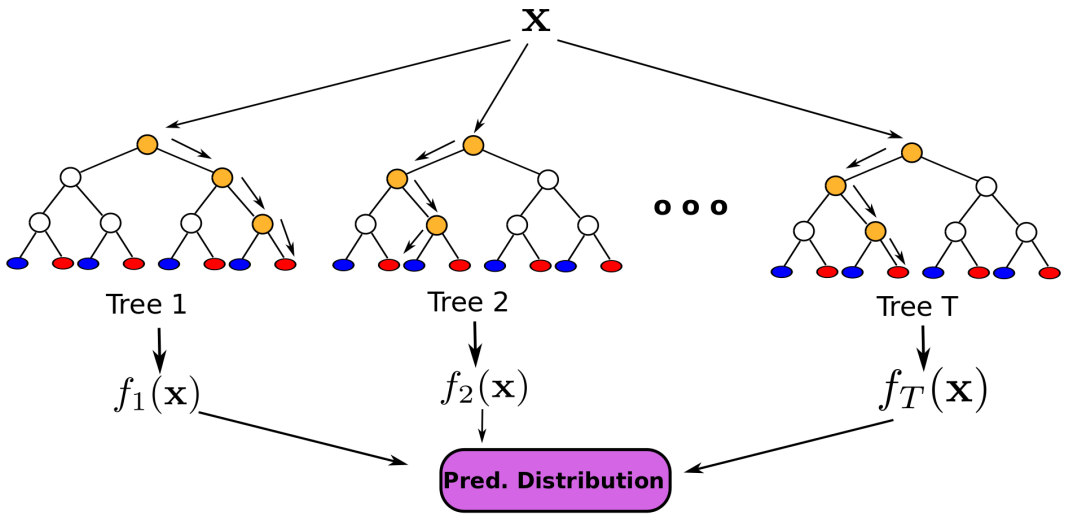

FIGURE 2.7: Random forest graphical representation. Trees are enforced to be diverse. Every tree outputs a different result. An approximate Gaussian predictive distribution can be computed using the empirical mean prediction and the variance of the predictions.

An advantage of RFs with respect to GPs is that they can generate conditional spaces, letting BO to easily optimize a hierarchical space. Conditional spaces are the ones involving variables that only arise when other variables are given certain values. For example, in deep neural networks, the number of neurons of layer 5 is only taken into account if the variable number of layers has value greater or equal to 5. Consider the joint hyper-parameter space given by two machine learning algorithms. We know that the hyper-parameters of one algorithm do not affect the performance of the other algorithm. If a search process considers the two spaces jointly without incorporating this knowledge, it will have to search in a space with as many dimensions as the sum of hyper-parameters of the two algorithms. If instead, the search process consider to model this space as a conditional space, it will do a hierarchical search. First, it will decide which of the two algorithms is more suited to the problem and then it will decide which value of the hyper-parameters of the chosen algorithm is the best. These conditional search spaces are modelled naturally by the splits of RF trees. RFs also deal with a high number of attributes in the instances to predict due to the high number of predictors that a RF contains. By considering conditional search spaces and a high number of attributes, RFs arise as a great alternative for automatic machine learning tools (Hutter et al., 2019). Bayesian optimization is a class of methods than can be employed for automatic machine learning (Hutter et al., 2019). An automatic machine

learning tool recommends a hyper-parametrized machine learning algorithm from a conditional search space of machine learning algorithms and their hyper-parameters to be trained for a given dataset in a budget of time. The conditional search space is traversed through a BO process with the RF as surrogate model, as it effectively deal with a high number of attributes and conditional regions of the search space. Examples of automatic machine learning tools are Auto-Sklearn and Auto-Weka (Feurer et al., 2019; Kotthoff et al., 2017). The main disadvantage of the RFs is that they tend to underestimate the variance of the predictive distribution, as we can see in Figure 2.8. We compare the predictive distributions of a GP with a squared exponential covariance function and a RF with 10 estimators and a maximum depth of 20 splits. The objective function is a one dimensional function with the following analytical expression $f(x) = \exp(\sin(25x) + \cos(15x))$. We extract 10 random samples from the objective function in the $[0, 1]$ range and fit the GP and RF to those data. Then we compute the predictive distribution of both models on a grid in $[0, 1]$. We can see how the GP variance of the predictive distribution is wider than the RF variance in unexplored areas. We can also see that the predictive distribution of the RF is not smooth. This makes the RF a worst candidate for modelling this particular smooth objective function.

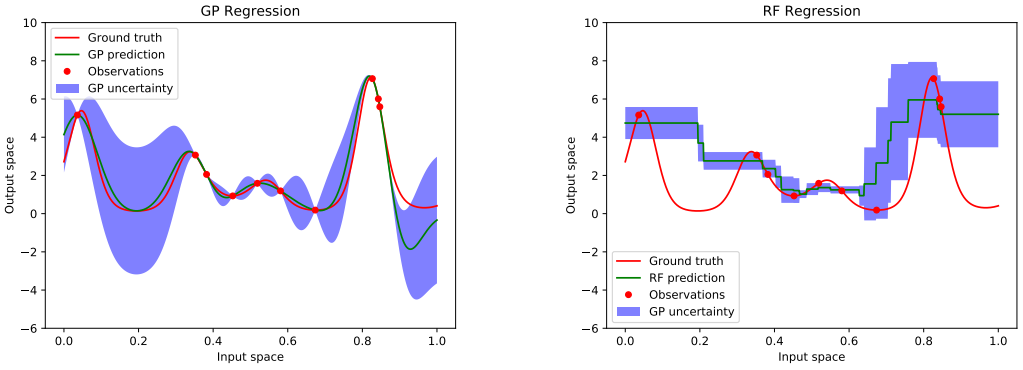

FIGURE 2.8: Gaussian process (left) and random forest (right) predictive distributions (mean: green, std dev: blue) of an objective function (red) after being trained on observations (red dots). Random forests are not as good as GPs modelling smooth functions.

- **T-Student processes**: A T-Student process is a GP with an inverse Wishart process prior on the covariance function (Shah et al., 2014). The Wishart distribution is a probability distribution over the set of real valued, symmetric, positive definite matrices. T-Student processes are an alternative to GPs as non-parameteric priors over functions. The T-Student process is a generalization of the GP that contains, as the T-Student distribution, an additional parameter to control how heavy are the tails of the process, the degrees of freedom ν. If $\nu \to \infty$, the multivariate T-Student distribution converges to a multivariate Gaussian distribution. Hence, if this parameter gets larger, the tails converge to a GP. It is consistent under marginalization and has analytical closed-form expressions for the marginal likelihood, the predictive and conditional distributions. Interestingly, the predictive mean form is the same as the one of the GP. The key difference with respect to GPs lies in the posterior variance expression, which in the case of T-Student

processes explicitly depends on the observations $\mathbf{y}$. Concretely, the GP has variance $v(\mathbf{x}^\star) = k(\mathbf{x}_\star, \mathbf{x}_\star) - \mathbf{k}_\star^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}_\star$. Therefore, the GP posterior variance does not depend on the observations $\mathbf{y}$. However, the T-Student process posterior variance is given by the following expression:

$$v(\mathbf{x}^\star) = \frac{\nu - 2}{\nu}\frac{\nu + \mathbf{y}^T\mathbf{K}^{-1}\mathbf{y} - 2}{\nu + |\mathcal{D}| - 2}(k(\mathbf{x}_\star, \mathbf{x}_\star) - \mathbf{k}_\star^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}_\star)\,, \tag{2.15}$$

where $|\mathcal{D}|$ is the number of observations that we have collected so far and we can see its dependence with the observations $\mathbf{y}$ (Tracey and Wolpert, 2018). It is claimed that the GP is an special case of T-Student process and that the T-Student process has many, if not all of the benefits of GP (Shah et al., 2014). T-Student processes increase flexibility at no extra cost, which suggests that T-Student processes can replace GPs in almost any application. In practice, however, GPs are easier to use, more interpretable and have less hyper-parameters, providing good results for the majority of the problems. This fact may be an explanation regarding why T-Student processes are not used as often as GPs for BO. Another argument is that we can just see the T-Student process as a GP with a Bayesian treatment of the scale. Although being more flexible, the scale becomes well determined very quickly, so the advantage given by T-Student processes vanishes as observations are fitted on the model. Regarding the use of T-Student processes in BO, it seems that the T-Student process shows improved performance over GPs with no additional computational costs in synthetic and benchmark functions such as Branin-Hoo or Hartmann (Shah et al., 2014, 2013). Although, in our opinion, there is no empirical evidence of the superiority of T-Student processes over GPs in real experiments or in an exhaustive benchmark.

- **Sparse Gaussian Processes**: A sparse Gaussian process is a GP whose covariance is parameterized by the the locations of M pseudo-input points, that are learned by a gradient based optimization procedure (Snelson and Ghahramani, 2005; Titsias, 2009). GPs lead to a cost of $\mathcal{O}(N^3)$ where N is the number of observations. If we have $N = 1000$ observations or more, the cost of fitting a GP is prohibitive in most settings, and can be even higher than evaluating an objective function in BO. Sparse Gaussian processes tackle this issue with the introduction of M pseudo-inputs or inducing points $\mathbf{Z}$ such that $M \ll N$. These inducing points lie in the same space as the rest of the data $\mathbf{X}$, summarizing it. The sparse Gaussian process model reduces the training cost from $\mathcal{O}(N^3)$ to $\mathcal{O}(M^2N)$. Therefore, in Sparse Gaussian processes, the training cost goes from being cubic on the number of observations to linear on the number of observations. Associated to the inducing points $\mathbf{Z} = (\mathbf{z}_1, ..., \mathbf{z}_M)^T$, we define the process values associated to the inducing points as $\mathbf{u} = (u_1, ..., u_M)^T$, *i.e.*, $u_i = f(\mathbf{z}_i)$. The process value at each point $\mathbf{x}_i$, is fully determined by $\mathbf{u}$ and given by:

$$p(\mathbf{f}|\mathbf{u}) = \mathcal{N}\left(\mathbf{f}|\mathbf{K}_{\mathbf{X},\mathbf{Z}}\mathbf{K}_{\mathbf{Z},\mathbf{Z}}^{-1}\mathbf{u}, \mathbf{K}_{\mathbf{X},\mathbf{X}} - \mathbf{K}_{\mathbf{X},\mathbf{Z}}\mathbf{K}_{\mathbf{Z},\mathbf{Z}}^{-1}\mathbf{K}_{\mathbf{Z},\tilde{\mathbf{X}}}\right)\,, \tag{2.16}$$

where $\mathbf{K}_{\mathbf{X},\mathbf{Z}}$ is the matrix of covariances at between the process values at the observed data points $\mathbf{X}$ and the inducing points $\mathbf{Z}$. Similarly, $\mathbf{K}_{\mathbf{Z},\mathbf{Z}}$ is the matrix of covariances between the process values at the inducing points $\mathbf{Z}$. The prior for

each $\mathbf{u}$ is simply the GP prior. Namely,

$$p(\mathbf{u}) = \mathcal{N}(\mathbf{u}|\mathbf{0}, \mathbf{K}_{\mathbf{Z},\mathbf{Z}}) \,. \tag{2.17}$$

The inducing point locations $\mathbf{Z}$ and their values $\mathbf{u}$ are unknown in practice. In order to compute them, we can maximize the marginal likelihood with respect to the inducing points $\mathbf{Z}$ and the hyper-parameters of the GP (Snelson and Ghahramani, 2005). This method approximates $p(\mathbf{f}|\mathbf{u})$ by $\prod_{i=1}^{N} p(f_i|\mathbf{u})$. This approximation makes the computational cost smaller. In particular, we obtain an approximate prior for the GP with a covariance matrix structure such that it can be inverted with cost $\mathcal{O}(NM^2)$. The assumption regarding the factorization of $p(\mathbf{f}|\mathbf{u})$ that leads to the described approximate prior considers the inducing point locations as hyper-parameters of the GP. Hence, these locations are included in the marginal likelihood. Maximizing such a marginal likelihood is a costly operation and can lead to overfitting. To avoid this problem, the inducing inputs can also be treated as variational parameters which are selected by minimizing the Kullback-Leibler divergence between the variational distribution $q(\mathbf{f}, \mathbf{u}) = p(\mathbf{f}|\mathbf{u})q(\mathbf{u})$ and the exact posterior distribution over the latent function values $p(\mathbf{f}, \mathbf{u}|\mathcal{D})$ given the data (Titsias, 2009). This method does not assume that $p(\mathbf{f}|\mathbf{u})$ factorizes but restricts the shape of $q(\mathbf{f}, \mathbf{u})$. In order to perform the minimization of the Kullback-Leibler divergence, first, we specify the initial variational Gaussian distribution for the inducing points: $q(\mathbf{u}) = \mathcal{N}(\mathbf{u}|\mathbf{m}, \mathbf{S})$. The parameters of this Gaussian distribution can be learnt through variational inference. Variational inference minimizes the KL divergence between the variational distribution of the inducing points $q(\mathbf{u})$ and the true posterior of the inducing points given the data $p(\mathbf{u}|\mathcal{D})$ with an optimization procedure. When we learn a distribution for the inducing points $q(\mathbf{u})$ via variational inference, the uncertainty about the inducing points $\mathbf{u}$ is introduced in Eq. (2.16) by marginalizing them:

$$p(\mathbf{f}|\mathbf{y}) \approx \int p(\mathbf{f}|\mathbf{u})q(\mathbf{u})d\mathbf{u} = \mathcal{N}(\mathbf{f}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \,. \tag{2.18}$$

Recall that we can perform this operation as the inducing point locations are treated as variables $q(\mathbf{u})$. We can marginalize them to obtain an averaged predictive distribution $p(\mathbf{f}|\mathbf{y})$ with respect to the probability of the location of the inducing points $q(\mathbf{u})$. We can expect this approach to work better since it does not assume that $p(\mathbf{f}|\mathbf{u})$ factorizes (Titsias, 2009). The parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ of the predictive Gaussian distribution $p(\mathbf{f}|\mathbf{y})$, are given by:

$$\begin{aligned} \boldsymbol{\mu} &= \mathbf{K}_{\mathbf{X},\mathbf{Z}}\mathbf{K}_{\mathbf{Z},\mathbf{Z}}^{-1}\mathbf{m} \,, \\ \boldsymbol{\Sigma} &= \mathbf{K}_{\mathbf{X},\mathbf{X}} - \mathbf{K}_{\mathbf{X},\mathbf{Z}}\mathbf{K}_{\mathbf{Z},\mathbf{Z}}^{-1}(\mathbf{K}_{\mathbf{Z},\mathbf{Z}} + \mathbf{S})\mathbf{K}_{\mathbf{Z},\mathbf{Z}}^{-1}\mathbf{K}_{\mathbf{Z},\tilde{\mathbf{X}}} \,, \end{aligned} \tag{2.19}$$

where $\mathbf{m}$ and $\mathbf{S}$ are the parameters of $q(\mathbf{u})$ that are computed with variational inference. The cost of computing Eq. (2.19) is $\mathcal{O}(NM^2)$.

- **Deep Gaussian Processes**: Deep GPs (DGPs) are a deep belief network based on GP mappings (Damianou and Lawrence, 2013). A graphical representation of this model explains the concept and intuition of a deep belief network based on GP mappings. Figure 2.9 illustrates the structure and components of a DGPs. This figure shows a DGP with two layers, where each of the layers contains 3 nodes and each node is a GP. The first hidden layer of this DGP consists of 3 GPs that

recieve as an input the data attributes given by x_1, x_2, x_3. The output of each of these GPs is a predictive distribution. Each of these predictive distributions, that are the outputs of the first layer are the inputs of the last layer of the DGP. This operation is defined as a GP mapping. Finally, the observed variables from the DGP are given by the output of the last layer, that also consists of 3 GP, in this particular example. DGPs are not restricted to two layers but can contain any positive number of layers L and nodes, or GPs, in each layer.

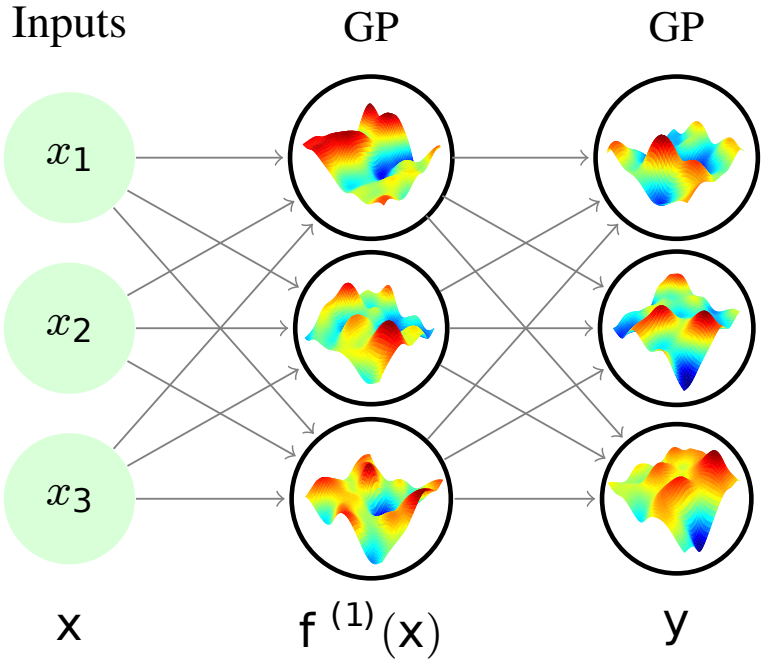

FIGURE 2.9: Graphical representation of the GP mappings of a DGP. A first layer of GPs fit the data, the last layer of GPs fit the output of the previous GPs.

GPs are flexible priors over functions but have difficulties modelling certain types of functions. For example, non-stationary functions. Recall that if a function has different variability across its range, we say that it is not-stationary. Standard covariance functions of the GP are stationary, *i.e.*, assume that the model hyper-parameters $\boldsymbol{\theta}$ stay constant over the input space $\mathcal{X}$. For example, a standard GP with a stationary covariance function assumes that the signal amplitude or range of variability of the function, given by the signal variance σ_f^2, is constant over the input space $\mathcal{X}$. It also assumes that the smoothness of the function is constant over the input space $\mathcal{X}$, property that is modelled by the lengthscales ℓ. To circumvent this issue, we can employ a different probabilistic surrogate model that accounts for that kind of complex functions. DGPs represent an alternative that may be competitive in this case (Damianou and Lawrence, 2013). As we have seen, deep GPs are a composition of GPs. By using a GP composition, we can obtain functions with different variability across the input space $\mathcal{X}$. That is, we can obtain non-stationary functions. Recall that these functions are cannot be obtained with a standard GP with a RBF covariance function. There is empirical evidence that deep models, such as DGPs or deep neural networks, seem to have structural advantages that can improve the learning of complicated data sets, complex and

non-stationary functions (Bengio, 2009). Hence, the motivation of using DGPs to model complex and non-stationary functions seems to be justified.

DGPs are based on a hierarchy of hidden layers of latent variables where each node, which is a GP, gives an output for each node of the layer above and gets the inputs from the nodes of the layer below. The observed outputs, $\mathbf{Y}$, are placed in the leaves of the hierarchy. For example, let us consider a stacking of two hidden layers of GPs. The hidden layer of GPs takes the N input space points $\mathbf{x}_n$ and generates a latent space given by a GP mapping of the input space $z_{nd} = f_d(\mathbf{x}_n)$ such that $f_d \sim \mathcal{GP}(0, k_d(\mathbf{X}, \mathbf{X}))$ and a GP is placed for every dimension of the input space $d = 1, ..., D$, generating a latent intermediate representation of every point $\mathbf{z}_n$. The output layer gets the information generated by the hidden layer and performs a similar mapping $y_{nq} = f_q(\mathbf{z}_n)$, such that $f_q \sim \mathcal{GP}(0, k_q(\mathbf{Z}, \mathbf{Z}))$ and $q = 1, ..., Q$, generating a Q dimensional observed output $\mathbf{Y}$. The proposed architecture can be extended vertically, deeper hierarchies, or horizontally, segmentation of each layer into different partitions of the output space.

The obvious problem of this hierarchy is that each layer l adds a significant number of parameters $\boldsymbol{\theta}_l$ to the model that need to be estimated. To do so, we place a Gaussian prior in the parent latent nodes of the hierarchy $p(\mathbf{Z}) = \mathcal{N}(\mathbf{Z}|\mathbf{0}, \mathbf{I})$ and propagate it through the hierarchy. Let $\mathbf{X}_i$ be the latent space of values where a hidden layer l of the DGP maps its inputs. Let $\mathbf{X}_1$ be the input space. In order to fully marginalize the latent spaces $\mathbb{X} = (\mathbf{X}_2, ..., \mathbf{X}_L)^T$ in a hierarchy with L hidden layers, we would have to compute the optimization of the marginal likelihood of all the intermediate latent spaces:

$$\log p(\mathbf{Y}) = \int p(\mathbf{Y}|\mathbf{X}_L) \prod_{l=1}^{L} p(\mathbf{X}_l|\mathbf{X}_{l-1}) p(\mathbf{X}_1|\mathbf{Z}) p(\mathbf{Z}) d\mathbf{X}_L, ..., d\mathbf{X}_2, d\mathbf{Z}\,. \tag{2.20}$$

This equation is intractable, as every layer is a generative GP mapping with complexity $\mathcal{O}(N^3)$ where N is the number of observations. In practice, fixing a single GP becomes infeasible when we have more than 1000 observations, due to the bottleneck of inverting the Gram matrix $\mathbf{K}$. This issue is worse in DGPs, where we can have several GP mappings. In order to circumvent this issue, DGPs substituted GPs from its layers to introduce Sparse GPs (Snelson and Ghahramani, 2006; Titsias, 2009). Via variational inference, any number of GP models can be stacked to build deep hierarchies of sparse GPs. The lower bound on the marginal likelihood of the model is an objective measure that can be used for model selection, *i.e.*, to select the number of layers L and number of nodes, or GPs, per layer (Damianou and Lawrence, 2013).

Different approaches have been proposed in the literature to perform the approximate inference of the posterior distribution of the latent variables of the DGP. We briefly give references about them, as the particular details of the approximate inference of the posterior distribution of the DGPs are out of the scope of this thesis and giving an explanation about them would require a lot of space. A joint variational distribution q can be defined in the place of the posterior of the latent variables of the Sparse GPs of the hidden layers given the outputs to obtain a tractable variational bound (Damianou and Lawrence, 2013). In this approach, different approximate variational Gaussian distributions $q(\mathbf{X})$ and $q(\mathbf{Z})$ that are independent between them model the inputs and outputs of each layer l. This

approach solves the problem but limits the application of DGPs to a range of small to medium scale regression problems. An approximation to the expectation propagation algorithm along with an efficient extension of the probabilistic backpropagation algorithm is proposed to enable DGPs for medium to large scale regression problems (Bui et al., 2016). We also find in the literature a doubly stochastic variational inference algorithm that uses samples to approximate the predictive distribution of each layer, not being restricted to approximate them via Gaussian distributions (Salimbeni and Deisenroth, 2017). Doubly stochastic variational inference optimizes the lower bound of variational inference using two sources of stochasticity, *i.e.*, using Monte Carlo techniques to solve expectations and through sub-sampling of the data. The authors of this approach use sparse variational inference to simplify the correlations within layers, but maintaining the correlations between layers. This approach enables the successfull application of DGPs in problems of billions of points (Salimbeni and Deisenroth, 2017). Empirical evidence shows that the posterior distribution of the latent variables of the DGP is not necessarily unimodal, but generally multimodal (Havasi et al., 2018). Previous approximations forced the posterior distribution of the inducing points in each GP to be Gaussian. As this situation does not necesarilly happen in practice, forcing this distribution for the inducing points of each GP can lead to poor results. The stochastic gradient Hamiltonian Monte Carlo sampling algorithm does not assume the target distribution to be Gaussian and it is suited for large amounts of data (Chen et al., 2014). It is hence a good candidate to generate samples of the posterior distribution of the DGPs and its application is proposed to solve the issue of the Gaussian assumption of the posterior obtaining state-of-the-art results in DGPs inference (Havasi et al., 2018).

If we apply DGPs as probabilistic surrogate models for BO, they need to be adapted for scenarios where the budget of evaluations N can be very low. As we have seen, each node of the DGP is a sparse GP. Recall that sparse GPs use a set of M inducing points whose locations where determined by maximizing the marginal likelihood or as a result of modelling them as latent variables and optimizing their positions through variational inference. In scenarios where the number of observations N is very low, as BO, the computational time spent by inverting the Gram matrix $\mathbf{K}$ of a GP is not worth the cost of approximating the set of N observations by a lower set of M inducing points. The locations of the inducing points could suffer from overfitting and losing accuracy in order to save computational time is not necessary for BO scenarios where the budget of evaluations is very low. Hence, if we want to use DGP for BO, substituting the Sparse GPs with standard GPs in each node is a task that is necessary to be done. An initial approach for adapting DGP to BO slightly modifies BO and DGPs (Hebbal et al., 2019). It uses the natural gradients of all the variational parameters of the DGP in training to enable a better convergence of the ELBO and a better uncertainty quantification on the prediction. To speed up the training of the DGP, this approach takes advantage of the fact that BO is an iterative algorithm and uses the optimal parameters of the model fitted in a previous iteration as the initial ones for the model of the following iteration. The same authors have used this approach for an application concerning aerospace system design (Hebbal et al., 2020).

- **Bayesian Neural Networks**: Traditional neural networks based on backpropagation are prone to overfitting, do not provide predictive distributions or uncertainties

over their parameters and require costly procedures to obtain optimal values of their hyper-parameters (Goodfellow et al., 2016; Hern'andez-Lobato and Adams, 2015). A Bayesian approach to neural networks overcomes these issues and makes them suitable as an alternative to GPs for BO (Springenberg et al., 2016). A Bayesian neural network (BNN) is a neural network with a prior distribution on its weights (MacKay, 1992; Neal, 2012). BNNs have been used in BO with success in scenarios with a high number of dimensions or where the scalability and flexibility provided by a GP is not enough to model an objective function (Springenberg et al., 2016). BNNs combine the flexibility and scalability of deep neural networks with well-calibrated uncertainty estimates.

Let us describe the necessary steps to perform the computation of the BNN predictive distribution $p(y^\star|\mathbf{x}^\star, \mathcal{D})$ for the evaluation $y^\star$ of a point $\mathbf{x}^\star$ given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., N\}$. First, we need to describe the different parameters of a common deep neural network that we want to incorporate in the BNN. A neural network can have lots of parameters that can be treated in a Bayesian framework but, in this exposition, we are only going to cover the weights of the neural network and, as the evaluations of the neural network are corruputed by additive noise variables ϵ_n, the precision γ of the Gaussian distribution that accounts for this noise $\epsilon_n \sim \mathcal{N}(0, \gamma^{-1})$. A deep neural network has L layers and V_l hidden units in every layer l. Let us define as $\mathcal{W} = \{\mathbf{W}_l\}_{l=1}^{L}$ the collection of $V_l x(V_{l-1} + 1)$ weight matrices between the fully connected layers. Bayesian inference of the posterior distribution of the weights given the data $p(\mathcal{W}|\mathcal{D})$ in BNNs is a required task to obtain predictive distributions of unknown outputs. It is also useful to perform model selection through the maximization of the marginal likelihood $p(\mathcal{D})$. The first thing to define for the computation of the posterior distribution is to choose the likelihood distribution for the network weights $\mathcal{W}$ and the noise precision γ given the data $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., N\}$. This likelihood is given by the following expression:

$$p(\mathbf{y}|\mathcal{W}, \mathbf{X}, \gamma) = \prod_{n=1}^{N} \mathcal{N}(y_n|f(\mathbf{x}_n; \mathcal{W}), \gamma^{-1}) \,, \tag{2.21}$$

where $f(\mathbf{x}_n; \mathcal{W})$ is the output of the deep neural network parametrized by the weights $\mathcal{W}$. Then, we need to define a prior for every weight of the network. We will assume independence between the weights, defining a joint prior distribution of the weights that factorizes. In this scenario, an acceptable choice for the prior of these weights are Gaussian distributions, as they are easy to work with in a Bayesian framework. This prior distribution for the weights can be modelled by the following joint distribution:

$$p(\mathcal{W}|\lambda) = \prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{ij,l}|0, \lambda^{-1}) \,, \tag{2.22}$$

where a weight $w_{ij,l}$ is indexed by the entry in the i-th row and j-th column of the weight matrix $\mathbf{W}_l$ of layer l and λ is a precision parameter. The hyper-priors for the precisions λ and γ can be modelled as Gamma distributions. The posterior distribution for the weights of the network $\mathcal{W}$, the noise precision of the evaluations of the γ and the precision for the prior of the weights λ can be obtained through

Bayes rule:

$$p(\mathcal{W}, \gamma, \lambda|\mathcal{D}) = \frac{p(\mathbf{y}|\mathcal{W}, \mathbf{X}, \gamma)p(\mathcal{W}|\lambda)p(\lambda)p(\gamma)}{p(\mathbf{y}|\mathbf{X})}\,, \tag{2.23}$$

where $p(\mathbf{y}|\mathbf{X})$ is a normalization constant, and the hyper-prior for the noise precision γ and the weight precision is Gamma: $p(\gamma) = p(\lambda) = \text{Gam}(\gamma|\alpha_0^\gamma, \beta_0^\gamma)$. Given all the previous expressions, we can now compute the posterior distribution $p(\mathcal{W}, \gamma, \lambda|\mathcal{D})$ and define a predictive distribution $p(y^\star|\mathbf{x}^\star, \mathcal{D})$ to predict the value $y^\star$ of a new input vector $\mathbf{x}^\star$. The predictive distribution for a new input vector $\mathbf{x}^\star$ is computed by marginalizing the precision random variables γ and λ and the weights $\mathcal{W}$ of the neural network. The predictive distribution is obtained by the following expression:

$$p(y_\star|\mathbf{x}_\star, \mathcal{D}) = \int\int\int p(y^\star|\mathbf{x}^\star, \mathcal{W}, \gamma)p(\mathcal{W}, \gamma, \lambda|\mathcal{D})\, d\gamma\, d\lambda\, d\mathcal{W}\,, \tag{2.24}$$

where $p(y^\star|\mathbf{x}^\star\mathcal{W}, \gamma)$ can be modelled as $\mathcal{N}(y^\star|f(\mathbf{x}^\star), \gamma)$. Unfortunately, the exact computation of the posterior distribution $p(\mathcal{W}, \gamma, \lambda|\mathcal{D})$ and, hence, the predictive distribution $p(f(\mathbf{x}^\star)|\mathbf{x}, \mathcal{D})$ is intractable as the number of weights of a common deep neural network is very high. Hence, as in the case of DGPs, we need to perform approximate inference methods to obtain uncertainty estimates for these weights.

We will not include the full details of the computations performed by the approximations of these uncertainty estimates as they get out of the scope of this thesis but give references to the articles that contain the full description of these computations. In order to get the uncertainty estimates for the weights $\mathcal{W}$, various approximate inference techniques can be used, that generally obtain a Gaussian approximation of the weights, or any other parameter, given the data $p(\mathcal{W}|\mathcal{D})$. The first approach to obtain these estimates was using the Laplace approximation to approximate the posterior distribution (MacKay, 1992). The Laplace approximation produces a Gaussian distribution that is placed on the mode of the target distribution and produces approximations that are too local, in the sense that although they match the mode of the exact posterior distribution, the approximation underestimates the variance of the posterior distribution. Other approximate inference techniques have been used to approximate the posterior distribution. Expectation propagation has shown better empirical results than the Laplace approximation for a wide variety of problems (Minka, 2001b; Nickisch and Rasmussen, 2008). Expectation propagation has been successfully applied for the approximation of the posterior distribution of the weights of a Bayesian neural network (Hern'andez-Lobato and Adams, 2015; Jylänki et al., 2014). Variational inference has also been used to learn a probability distribution of the weights of a neural network in the Bayes by Backprop algorithm (Blundell et al., 2015). As in the case of DGPs, the mentioned techniques assume that the posterior distributions of the weights is unimodal, which is a very strong assumption that can hurt the performance of the approximate predictive distributions of new outputs $\mathbf{y}^\star$. In order to circumvent this issue, Stochastic Hamiltonian Monte Carlo (HMC) methods has been successfully applied for obtaining samples of the approximate posterior distribution of the weights (Chen et al., 2014). Although BNNs provide high flexibility and scalability, the computations needed to approximately obtain the uncertainty estimates of the weights produce a higher computational cost than the one of GPs. Although, BNNs

are an alternative that has proved empirical success on BO (Springenberg et al., 2016).

2.6 Approximate Inference

Let us consider the case of GPs using Gaussian likelihoods $p(\mathbf{y}|\mathbf{f})$. In this scenario, an analytical exact solution for the GP predictive distribution can be computed given by Eq. (2.5). The problem becomes more difficult to solve when we have GPs with non-Gaussian likelihoods $p(\mathbf{y}|\mathbf{f})$. In this case, we do not have analytical closed-form expressions for the GP posterior $p(\boldsymbol{\theta}|\mathcal{D})$, predictive $p(f(\mathbf{x})|\mathcal{D})$ or marginal distribution $p(\mathcal{D})$. Unfortunately, in this scenario, the computation of these distributions cannot be done in closed-form and is generally intractable. A possible solution to approximate these distributions, in the case of having non-Gaussian likelihoods, is to approximate each non-Gaussian likelihood with a Gaussian factor. Approximate inference algorithms, and concretely the expectation propagation algorithm that is going to be described in this section, can perform that operation. From now on, in this section, let us denote with $\boldsymbol{\theta}$ the latent variables, or parameters, of the model we are interested in making inference of. For example, in the particular case of GPs, latent variables $\mathbf{f}$ will be represented by $\boldsymbol{\theta}$. Approximate inference algorithms approximate intractable target distributions $p(\boldsymbol{\theta})$ by simpler distributions $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$, where $\boldsymbol{\phi}$ represents the set of parameters of the approximate distribution $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$ (Bishop, 2006; Blei et al., 2017; Murphy, 2012). For clarity, we will abbreviate the representation of the approximate distribution $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$ of parameters $\boldsymbol{\phi}$ as $q(\boldsymbol{\theta})$, omiting the dependence of $\boldsymbol{\phi}$.

Approximate inference techniques obtain parametric approximations $q(\boldsymbol{\theta})$ via the optimization of a divergence D between the target distribution $p(\boldsymbol{\theta})$ and the proposed parametric distribution $q(\boldsymbol{\theta})$. Intuitively, by carrying out this process, we are computing the most similar distribution $q(\boldsymbol{\theta})$ of a parametric family $\mathcal{F}$ to the target distribution $p(\boldsymbol{\theta})$ in terms of the divergence D. Figure 2.10 illustrates this intuition. The result obtained by the approximate inference technique is a set of parameter values

$$\boldsymbol{\phi}^{\star} = \arg\min_{\boldsymbol{\phi}\in\boldsymbol{\Phi}} D(q_{\phi}(\boldsymbol{\theta})||p(\boldsymbol{\theta}))\,, \tag{2.25}$$

that minimize the divergence D between the proposed distribution $q(\boldsymbol{\theta})$ and the target distribution $p(\boldsymbol{\theta})$. In other words, it finds the most similar member $q_{\boldsymbol{\phi}^{\star}}(\boldsymbol{\theta})$ of a parametric family of distributions $\mathcal{F}$ to the target distribution $p(\boldsymbol{\theta})$. One example of a divergence measure D between probability distributions is the Kullback-Leibler (KL) divergence. The KL divergence between two probability distributions with densities $p(\boldsymbol{\theta})$ and $q(\boldsymbol{\theta})$ over continuous variables is given by the following expression:

$$\mathrm{KL}(p(\boldsymbol{\theta})||q(\boldsymbol{\theta})) = \int_{-\infty}^{\infty} p(\boldsymbol{\theta}) \log\left(\frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right) d\boldsymbol{\theta}\,. \tag{2.26}$$

The KL divergence between two distributions is always greater or equal than zero, *i.e.*, $\mathrm{KL}(p(\boldsymbol{\theta})||q(\boldsymbol{\theta})) \geq 0$. It is important to remark that the KL divergence is not symmetric, *i.e.*, $\mathrm{KL}(q(\boldsymbol{\theta})||p(\boldsymbol{\theta})) \neq \mathrm{KL}(p(\boldsymbol{\theta})||q(\boldsymbol{\theta}))$. If two probability distributions with densities $p(\boldsymbol{\theta})$ and $q(\boldsymbol{\theta})$ are equal, $p(\boldsymbol{\theta}) = q(\boldsymbol{\theta})$, then we have that $\mathrm{KL}(p(\boldsymbol{\theta})||q(\boldsymbol{\theta})) = 0$.

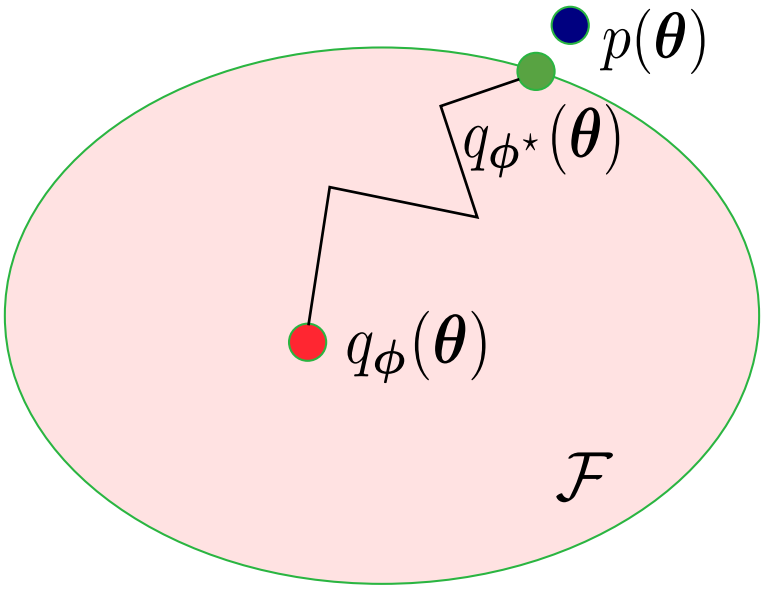

FIGURE 2.10: Visualization of the behavior of an approximate inference technique. The parametric distribution $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$ approximates the target distribution $p(\boldsymbol{\theta})$ as closely as possible. The distribution $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$ belongs to the exponential family $\mathcal{F}$, whose parametric space is plotted as a pink ellipse. At the end of the process, we obtain the set of parameter values $\boldsymbol{\phi}^\star$ that minimizes the KL divergence of $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$ with respect to $p(\boldsymbol{\theta})$. In other words, the approximate inference technique finds the most similar exponential family member $q_{\boldsymbol{\phi}^\star}(\boldsymbol{\theta})$ to the target distribution $p(\boldsymbol{\theta})$ in terms of the divergence.

2.6.1 Exponential Family

The exponential family of distributions $\mathcal{F}$ is a parametric set of probability distributions that, for example, includes the Gaussian, Beta or Gamma distributions (Duda et al., 1973). The probability distributions that belong to the exponential family have properties in common. In particular, if the distribution $q(\boldsymbol{\theta})$ belongs to the exponential family $\mathcal{F}$ of probability distributions, it can be written as:

$$q(\boldsymbol{\theta}) = \exp(\boldsymbol{\eta}^{\mathrm{T}}\boldsymbol{u}(\boldsymbol{\theta}) - g(\boldsymbol{\eta}))\,, \tag{2.27}$$

where T means transpose, $\boldsymbol{\eta}$ is a vector of natural parameters, $\boldsymbol{u}(\boldsymbol{\theta})$ is a vector function called sufficient statistics and $g(\boldsymbol{\eta}) = \log \int \exp(\boldsymbol{\eta}^{\mathrm{T}} u(\boldsymbol{\theta}))d\boldsymbol{\theta}$ is a log partition normalization function (Seeger, 2005). For the particular case of the Gaussian distribution, it can be written as:

$$\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \exp(\boldsymbol{\eta}^{T}\mathbf{u}(\boldsymbol{\theta}) - g(\boldsymbol{\eta}))\,, \tag{2.28}$$

where

$$\begin{aligned}
\mathbf{u}(\boldsymbol{\theta}) = (&\theta_1, \ldots, \theta_d, \\
&\theta_1^2, \theta_1\theta_2, \ldots, \theta_1\theta_d, \\
&\theta_1\theta_2, \theta_2^2, \ldots, \theta_2\theta_d, \\
&\vdots \\
&\theta_d\theta_1, \theta_d\theta_2, \ldots, \theta_d^2)^T\,,
\end{aligned}
\qquad
\begin{aligned}
\boldsymbol{\eta} = -\frac{1}{2}\big(&-2\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\,, \\
&\Sigma_{11}^{-1}, \Sigma_{12}^{-1}, \ldots, \Sigma_{1d}^{-1}, \\
&\Sigma_{21}^{-1}, \Sigma_{22}^{-1}, \ldots, \Sigma_{1d}^{-1}, \\
&\vdots \\
&\Sigma_{d1}^{-1}, \Sigma_{d2}^{-1}, \ldots, \Sigma_{dd}^{-1}\big)^T
\end{aligned}
\tag{2.29}$$

and $g(\boldsymbol{\eta}) = (1/2)\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + (d/2)\log(2\pi) + (1/2)\log(|\boldsymbol{\Sigma}|)$. One property of exponential family $\mathcal{F}$ members is that the family is closed under product and division. That is, performing a product or division of members of the exponential family gives another exponential family member that can be exactly computed through an analytical expression. The parameters of the exponential family member change as a result of these operations but not the vector of sufficient statistics $\boldsymbol{u}(\boldsymbol{\theta})$. In the particular case of the Gaussian distribution, we substract the values of the natural parameters of a Gaussian distribution when we divide a Gaussian distribution by with another Gaussian distribution. Similarly, we sum the values of the natural parameters when we multiply a Gaussian distribution by another Gaussian distribution. More information about these operations can be seen in Appendix A. We can compute the KL divergence between a target distribution $p(\boldsymbol{\theta})$ and an exponential family member distribution $q(\boldsymbol{\theta})$. The KL divergence is given by:

$$\mathrm{KL}(p(\boldsymbol{\theta})||q(\boldsymbol{\theta})) = g(\boldsymbol{\eta}) - \boldsymbol{\eta}^T\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{u}(\boldsymbol{\theta})] + \mathcal{C}\,, \tag{2.30}$$

where $\mathcal{C}$ is a constant value that is independent of the natural parameters $\boldsymbol{\eta}$. $\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{u}(\boldsymbol{\theta})]$ is the expectation of the sufficient statistics of the parametric approximate distribution $q(\boldsymbol{\theta})$ with respect to the target distribution $p(\boldsymbol{\theta})$. If two probability distributions are similar, then, their KL divergence is zero. Hence, if we want to make two probability distributions more similar, we can minimize their KL divergence.

Let $q(\boldsymbol{\theta})$ be a probability distribution that belongs to the exponential family $\mathcal{F}$. We want to make it similar to the target distribution $p(\boldsymbol{\theta})$. Minimizing $\mathrm{KL}(p(\boldsymbol{\theta})||q(\boldsymbol{\theta}))$ with respect to the parameters of $q(\boldsymbol{\theta})$ is equivalent to making the gradients of Eq. (2.30) with respect to $\boldsymbol{\eta}$ equal to zero, as we show with the following expression:

$$\frac{\mathrm{KL}(p(\boldsymbol{\theta})||q(\boldsymbol{\theta}))}{\partial\boldsymbol{\eta}} = 0 \iff \frac{\partial g(\boldsymbol{\eta})}{\partial\boldsymbol{\eta}} = \mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{u}(\boldsymbol{\theta})]\,. \tag{2.31}$$

As a corollary, an interesting result that we obtain is that the gradient of $g(\boldsymbol{\eta})$ is given by the expectation of $\boldsymbol{u}(\boldsymbol{\theta})$ under the exponential family distribution member $q(\boldsymbol{\theta})$. The gradient of $g(\boldsymbol{\eta})$ is called the expected sufficient statistics. It is easy to show that in the case of $\mathrm{KL}(p(\boldsymbol{\theta})||q(\boldsymbol{\theta}))/\partial\boldsymbol{\eta} = 0$ we also have $\partial g(\boldsymbol{\eta})/\partial\boldsymbol{\eta} = \mathbb{E}_{q(\boldsymbol{\theta})}[\boldsymbol{u}(\boldsymbol{\theta})]$, therefore $\mathbb{E}_{q(\boldsymbol{\theta})}[\mathbf{u}(\boldsymbol{\theta})] = \mathbb{E}_{p(\boldsymbol{\theta})}[\mathbf{u}(\boldsymbol{\theta})]$. This is a great result, as $q(\boldsymbol{\theta})$ is a parametric distribution with analytic closed-form expressions and a distribution from which we can sample from. Another advantage is that now, the optimal distribution $q_{\boldsymbol{\phi}^\star}(\boldsymbol{\theta})$ can be obtained simply by matching the expected sufficient statistics $\mathbb{E}_{q(\boldsymbol{\theta})}[\mathbf{u}(\boldsymbol{\theta})]$ and $\mathbb{E}_{p(\boldsymbol{\theta})}[\mathbf{u}(\boldsymbol{\theta})]$. This process is called moment matching.

Let us suppose that the exponential family distribution member $q(\boldsymbol{\theta})$ is a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. We want to minimize the KL divergence of $q(\boldsymbol{\theta})$ with respect to a target distribution $p(\boldsymbol{\theta})$. To do so, as we have seen in the previous paragraph, we just need to match the moments or expected sufficient statistics, of the distributions $p(\boldsymbol{\theta})$ and $q(\boldsymbol{\theta})$. In this particular example, this is equal to set the mean $\boldsymbol{\mu}$ of $q(\boldsymbol{\theta})$ equal to the mean of $p(\boldsymbol{\theta})$ and to set the covariance matrix $\boldsymbol{\Sigma}$ of $q(\boldsymbol{\theta})$ equal to the covariances of $p(\boldsymbol{\theta})$. Figure 2.11 illustrates this process. In this Figure, we want to approximate a mixture of Gaussians represented by $p(\boldsymbol{\theta})$ with a simpler parametric distribution $q(\boldsymbol{\theta})$ that belongs to the exponential family $\mathcal{F}$. By matching the moments of both distributions, we obtain the optimal distribution $q_{\boldsymbol{\phi}^\star}(\boldsymbol{\theta})$.

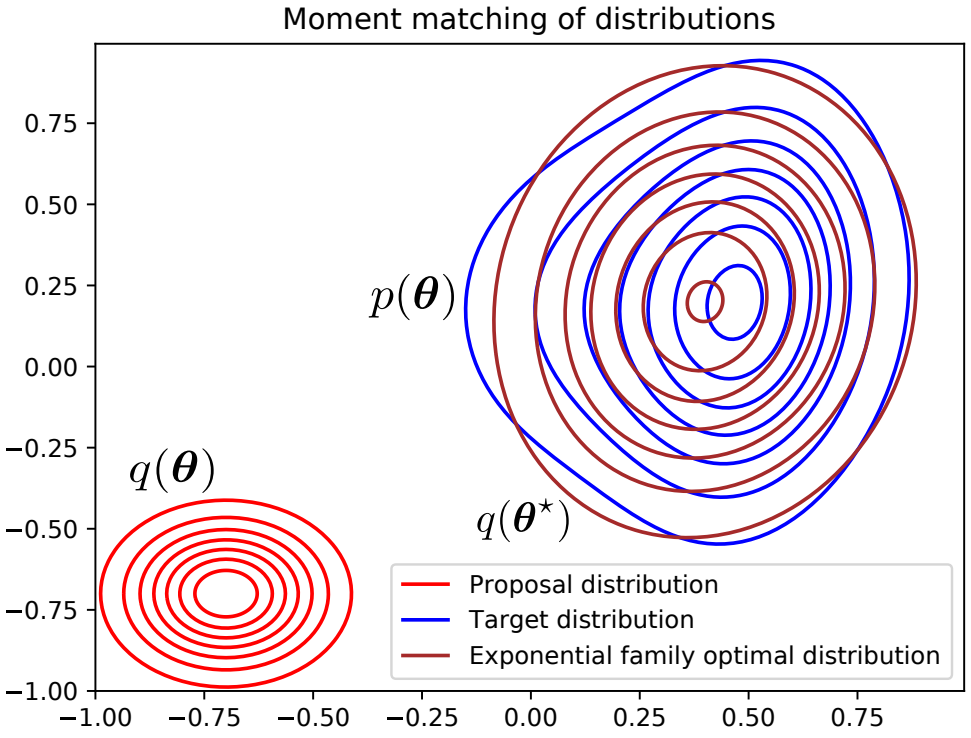

FIGURE 2.11: Moment matching of distributions. The exponential family distribution member $q(\boldsymbol{\theta})$ matches the moments of the target mixture of Gaussians $p(\boldsymbol{\theta})$. The resultant distribution $q_{\boldsymbol{\phi}^\star}(\boldsymbol{\theta})$ is the most similar exponential family member with respect to $p(\boldsymbol{\theta})$.

2.6.2 Expectation Propagation

In this subsection, the *Expectation Propagation* (EP) algorithm is described in detail (Minka, 2001a). EP is an approximate inference algorithm. It minimizes the KL divergence in an approximate way between a target distribution $p(\boldsymbol{\theta})$ and an exponential family $\mathcal{F}$ probability distribution member $q(\boldsymbol{\theta})$. As we have seen in the previous subsection, this is equivalent to matching the expected sufficient statistics under $q(\boldsymbol{\theta})$ and $p(\boldsymbol{\theta})$: $\mathbb{E}_{q(\boldsymbol{\theta})}[\mathbf{u}(\boldsymbol{\theta})] = \mathbb{E}_{p(\boldsymbol{\theta})}[\mathbf{u}(\boldsymbol{\theta})]$.

EP assumes that the intractable distribution $p(\boldsymbol{\theta})$ factorizes in a product of a set of factors of the form

$$p(\boldsymbol{\theta}) = \frac{1}{Z}\prod_{i=1}^{N} f_i(\boldsymbol{\theta})\,, \tag{2.32}$$

where Z is the normalization constant. EP approximately minimizes the KL of the previous distribution $p(\boldsymbol{\theta})$ with a parametric proposal distribution $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$, that is, $\mathrm{KL}(p(\boldsymbol{\theta})||q(\boldsymbol{\theta}))$ by exploiting the fact that the target distribution $p(\boldsymbol{\theta})$ can be written as a product of factors $f_i(\boldsymbol{\theta})$. These factors do not need to be normalized. In particular, EP can be applied for the inference of the parameters of probabilistic machine learning models, $\boldsymbol{\theta}$. Let the collection of points $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., N\}$ be independent and identically distributed (i.i.d. assumption). Therefore, the joint distribution $p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$ factorizes into the product of likelihood factors of each data instance $p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$. That is, factor $f_i(\boldsymbol{\theta})$, corresponding to every predictive distribution $p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$, will be a function of each datum $(\mathbf{x}_i, y_i)$ that has the same shape as the others and all are mutually

independent. Therefore, in these models, the joint distribution of data $\mathcal{D}$ and parameters, or hidden variables, $\boldsymbol{\theta}$ factorizes. That is, $p(\mathcal{D}, \boldsymbol{\theta}) = \prod_{i=1}^{N+1} f_i(\boldsymbol{\theta})$ where N is the size of the dataset. In this case, we have a factor for each data point $f_i(\boldsymbol{\theta})$ and a factor for the prior (Minka, 2001a,b). EP approximates each of the non-Gaussian likelihood factors, $p(\mathcal{D}, \boldsymbol{\theta}) = \prod_{i=1}^{N+1} f_i(\boldsymbol{\theta})$, using approximate factors $\tilde{f}_i(\boldsymbol{\theta})$ that belong to the exponential family $\mathcal{F}$. If we have a non-Gaussian prior, $f_0(\boldsymbol{\theta})$, EP can also approximate it by using an exponential family factor. Recall that the product and division are closed for the exponential family distributions, as we have seen in previous sections. If all the factors of the exact distribution $p(\boldsymbol{\theta})$ were members of the exponential family, we could apply Bayes theorem to obtain a posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ using the product and division analytical expressions for exponential family members (see Appendix A for more details):

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{\prod_{i=0}^{N} f_i(\boldsymbol{\theta})}{\int \prod_{i=0}^{N} f_i(\boldsymbol{\theta}) d\boldsymbol{\theta}} = \frac{\prod_{i=0}^{N} f_i(\boldsymbol{\theta})}{p(\mathcal{D})} \,. \tag{2.33}$$

The previous expression is, however, intractable, as it involves the computation of the normalization constant of the posterior or marginal likelihood $p(\mathcal{D})$. This is an integral over the high dimensional space of parameters $\boldsymbol{\theta}$. To compute the posterior distribution of the parameters given the data $p(\boldsymbol{\theta}|\mathcal{D})$, an approximation needs to be done. This is what EP precisely does, in an efficient way. EP approximates the target distribution $p(\boldsymbol{\theta})$ by another distribution $q(\boldsymbol{\theta})$, also generated by a product of factors $\prod_{i=0}^{N} \tilde{f}_i(\boldsymbol{\theta})$. Recall that the target distribution $p(\boldsymbol{\theta})$ can be expressed as a product of factors, see (2.33). EP approximates the previous distribution $p(\boldsymbol{\theta})$ with another product of approximate factors, given by the following expression:

$$q(\boldsymbol{\theta}) = \frac{1}{Z} \prod_{i=1}^{N+1} \tilde{f}_i(\boldsymbol{\theta}) \,, \tag{2.34}$$

where Z is a normalization constant. Each factor $\tilde{f}_i(\boldsymbol{\theta})$ approximates the respective exact factor $f_i(\boldsymbol{\theta})$ of the true posterior. Approximate factors $f_i(\boldsymbol{\theta})$ do not need to be normalized. By assuming that the intractable distribution can be expressed as a product of factors $\prod_i \tilde{f}(\boldsymbol{\theta})$, EP approximates the product of intractable exact factors $\prod_i f_i(\boldsymbol{\theta})$ with exponential family approximate factors $\prod_i \tilde{f}_i(\boldsymbol{\theta})$ (where each factor $\tilde{f}_i(\boldsymbol{\theta})$ is a parametric known unormalized distribution). As the product is closed for the exponential family and $q(\boldsymbol{\theta})$ is the product of the exponential family factors $\tilde{f}_i(\boldsymbol{\theta})$, then, $q(\boldsymbol{\theta})$ is also an exponential family member. For example, let $\tilde{f}_i(\boldsymbol{\theta})$ be an univariate Gaussian distribution $\mathcal{N}(\theta_i|\mu_i, \sigma_i)$. A Gaussian distribution is an exponential family member. Hence, the product of Gaussian distributions $\prod_{i=0}^{N} \mathcal{N}(\theta_i|\mu_i, \sigma_i)$ has an analytical-closed form expression, and is Gaussian, as we have seen before. An additional advantage is that $q(\boldsymbol{\theta})$ is easy to normalize, as we will later explain in this section.

If we want to make $p(\boldsymbol{\theta})$ and $q(\boldsymbol{\theta})$ similar, we need to minimize the KL divergence of the product of the exact factors $\prod_i f_i(\boldsymbol{\theta})$ and the approximate factors $\prod_i \tilde{f}_i(\boldsymbol{\theta})$

$$\mathrm{KL}(p(\boldsymbol{\theta})||q(\boldsymbol{\theta})) = \mathrm{KL}(\frac{1}{p(\mathcal{D})} \prod_i f_i(\boldsymbol{\theta}) || \frac{1}{Z} \prod_i \tilde{f}_i(\boldsymbol{\theta})) \,. \tag{2.35}$$

Minimizing Eq. (2.35) is generally intractable, as it involves the computation of the normalization constant $p(\mathcal{D})$. In order to circumvent this issue, EP does an approximation where it minimizes the KL divergence of each pair of correspondant factors $f_i(\boldsymbol{\theta})$ and

$\tilde{f}_i(\boldsymbol{\theta})$ at once. It is expected that this process will lead in something similar to the global minimization of the KL divergence. To perform these minimizations, EP performs the following computations. Each approximate factor $\tilde{f}_j(\boldsymbol{\theta})$ has parameters. First, EP uniformly initializes the parameters of the approximate factors $\tilde{f}_j(\boldsymbol{\theta})$ to an initial value. For example, let us suppose that we place an univariate Gaussian distribution $\mathcal{N}(\theta_j|\mu_j, \sigma_j)$ for each approximate factor $\tilde{f}_j(\theta_j)$. In practice, we can set $\mu_j = 0$ and $\sigma_j = \infty \; \forall j \in [0, N]$. Factors do not need to be univariate, they can be multivariate. After the initialization of each factor $\tilde{f}_j(\theta_j)$, we initialize the posterior approximation $q(\boldsymbol{\theta})$ to be the product of the factors $\prod_j \tilde{f}_j(\theta_j)$. In the case of Gaussian distributions, this is simply done by converting the parameters of every Gaussian distribution to natural parameters and then summing their values (see Appendix A for more information). We now enter a double loop. The outer loop iterates until every parameter of the approximate factors $\tilde{f}_j(\theta_j)$ have converged. The inner loop iterates over all the $N + 1$ approximate factors $\tilde{f}_j(\theta_j)$. We perform the following operations inside the described double loop. First, we select the approximate factor $\tilde{f}_j(\theta_j)$. EP will refine this factor $\tilde{f}_j(\theta_j)$, and the other ones, by the following process. To refine a factor, $\tilde{f}_j(\theta_j)$, the first step is to remove it from the product of approximate factors $\prod_{j=0}^{N} \tilde{f}_j(\theta_j)$. Let the approximate distribution of all the approximate factors except $\tilde{f}_j(\theta_j)$ be called the cavity distribution $q^{\setminus j}(\boldsymbol{\theta})$, which is given by:

$$q^{\setminus j}(\boldsymbol{\theta}) = \frac{1}{Z^{\setminus j}} \prod_{i \neq j} \tilde{f}_i(\theta_i) \,, \tag{2.36}$$

where $Z^{\setminus j}$ is the normalization constant $Z^{\setminus j} = \int \prod_{i \neq j} \tilde{f}_i(\theta_i) d\boldsymbol{\theta}$. It is important to observe that not multiplying the approximate factor $\tilde{f}_j(\theta_j)$ is equivalent to divide the joint distribution $q(\boldsymbol{\theta})$ by $\tilde{f}_j(\theta_j)$:

$$q^{\setminus j}(\boldsymbol{\theta}) = \frac{Z}{Z^{\setminus j}} \frac{q(\boldsymbol{\theta})}{\tilde{f}_j(\theta_j)} \,. \tag{2.37}$$

Suppose that all the approximate factors $\tilde{f}_j(\theta_j)$ are Gaussian distributions $\mathcal{N}(\theta_j|\mu_j, \sigma_j)$. The next step of the algorithm is to multiply the exact factor $f_j(\theta_j)$, corresponding to the removed approximate factor $\tilde{f}_j(\theta_j)$, by the cavity distribution $q^{\setminus j}(\boldsymbol{\theta})$ computed before. This operation is done through the next expression:

$$\hat{p}_j(\boldsymbol{\theta}) = \frac{1}{Z_j} f_j(\theta_j) \tilde{q}^{\setminus j}(\boldsymbol{\theta}) \,, \tag{2.38}$$

where the resultant distribution $\hat{p}_j(\boldsymbol{\theta})$ is called the tilted distribution. Z_j is the normalization constant, given by $Z_j = \int f_j(\theta_j) q^{\setminus j}(\boldsymbol{\theta}) d\boldsymbol{\theta}$. We now need to define an additional distribution, the approximate new distribution $q^{\text{new}}(\boldsymbol{\theta})$. This distribution is simply defined by the product of all the approximate factors:

$$q^{\text{new}}(\boldsymbol{\theta}) = \frac{1}{Z} \tilde{f}_j(\theta_j) \prod_{i \neq j} \tilde{f}_i(\theta_i) \,, \tag{2.39}$$

where Z is the normalization constant. We will use the approximate new distribution $q^{\text{new}}(\boldsymbol{\theta})$ to refine the approximate factor $\tilde{f}_j(\theta_j)$. In order to do so, we minimize the KL divergence of the new approximate distribution $q^{\text{new}}(\boldsymbol{\theta})$ with the tilted distribution $\hat{p}_j(\boldsymbol{\theta})$. This minimization modifies the parameters of the approximate factors $\tilde{f}_j(\theta_j)$ of the new approximate distribution $q^{\text{new}}(\boldsymbol{\theta})$. In other words, the result of the minimization is the

most similar new approximate distribution $q^{\text{new}}(\boldsymbol{\theta})$ to the tilted distribution $\hat{p}_j(\boldsymbol{\theta})$ in terms of the KL divergence.

$$\text{KL}(\hat{p}_j(\boldsymbol{\theta})||q^{\text{new}}(\boldsymbol{\theta}))\,. \tag{2.40}$$

After minimizing this divergence, $q^{\text{new}}(\boldsymbol{\theta})$ contains the information of the refined approximate factor $\tilde{f}_j(\theta_j)$. We can think that $q^{\text{new}}(\boldsymbol{\theta})$ as a placeholder that has shifted from an initial position $\boldsymbol{\phi}$ in the exponential family parameter space $\boldsymbol{\Phi}$ to the closest position of the space $\boldsymbol{\phi}^\star$ to the tilted distribution $\hat{p}_j(\boldsymbol{\theta})$. We can observe a graphical representation of this idea in Figure 2.10. The movement has been done via minimizing the KL divergence of the new approximate distribution $q^{\text{new}}(\boldsymbol{\theta})$ with the tilted distribution $\hat{p}_j(\boldsymbol{\theta})$. The intuition is that we iteratively learn about each of the exact factors $f_j(\theta_j)$ by minimizing the distance of the approximate factor $\tilde{f}_j(\theta_j)$ to the exact factor $f_j(\theta_j)$ in the context of the cavity distribution $q^{\backslash j}(\boldsymbol{\theta})$. By performing this process iteratively, we do not lose the information of any exact factor $f_j(\theta_j)$ while retaining the most important information of the target distribution $p(\boldsymbol{\theta})$.

The minimization of the KL divergence is carried out by moment matching of the distributions $\hat{p}_j(\boldsymbol{\theta})$ and $q^{\text{new}}(\boldsymbol{\theta})$. Let us consider an example. Let $q^{\text{new}}(\boldsymbol{\theta})$ be a Gaussian distribution $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. In this case $\boldsymbol{\mu}$ would be equal to the mean of the un-normalized distribution $f_j(\theta_j)q^{\backslash j}(\boldsymbol{\theta})$ and $\boldsymbol{\Sigma}$ to its covariance. More generally, for any member of the exponential family, that could be normalized, we can obtain the required expected sufficient statistics. In the Gaussian case, these are the expectations $\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}]$ and $\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}\boldsymbol{\theta}^T] - \mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}]\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}]^T$. These expected sufficient statistics are used for matching the moments, since the expected statistics $u(\boldsymbol{\theta})$ are related to the derivatives of the normalization coefficient Z_j. In the case of Gaussian distributions, let $t(\boldsymbol{\theta})$ be an arbitrary function of $\boldsymbol{\theta}$, that can be an intractable factor of the true posterior that we are trying to approximate, and let

$$Z = \int t(\boldsymbol{\theta})\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})d\boldsymbol{\theta}\,, \qquad p(\boldsymbol{\theta}) = \frac{1}{Z}t(\boldsymbol{\theta})\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\,. \tag{2.41}$$

$p(\boldsymbol{\theta})$ is a probability distribution, as it has been normalized by the constant Z. By considering the expression of the natural moments shown before, EP matches the moments of distributions by using the following expressions:

$$\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}] = \boldsymbol{\mu} + \boldsymbol{\Sigma}\frac{\partial \log(Z)}{\partial \boldsymbol{\mu}}\,, \tag{2.42}$$

$$\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}\boldsymbol{\theta}^T] - \mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}]\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}]^T = \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\left(\frac{\partial \log(Z)}{\partial \boldsymbol{\mu}}\left(\frac{\partial \log(Z)}{\partial \boldsymbol{\mu}}\right)^T - 2\frac{\partial \log(Z)}{\partial \boldsymbol{\Sigma}}\right)\boldsymbol{\Sigma}\,. \tag{2.43}$$

In some practical situations, $\partial \log(Z)/\partial \boldsymbol{\Sigma}$ is not robust. In that cases, we can use the second derivative of the mean, $\partial^2 \log(Z)/\partial(\boldsymbol{\mu})^2$, rather than $\partial \log(Z)/\partial \boldsymbol{\Sigma}$. By doing it so, Eq. (2.43) is now given by the following expression:

$$\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}\boldsymbol{\theta}^T] - \mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}]\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}]^T = \boldsymbol{\Sigma}\frac{\partial^2 \log(Z)}{\partial(\boldsymbol{\mu})^2}\boldsymbol{\Sigma} + \boldsymbol{\Sigma}\,. \tag{2.44}$$

We can hence use this expression to compute $\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}]\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}]^T$ in Eq. (2.43), see Appendix A for the derivation of Eq. (2.43). According to different posteriors and the used approximate distributions, the particular expressions for Z, $\partial \log(Z)/\partial \boldsymbol{\mu}$, $\partial \log(Z)/\partial \boldsymbol{\Sigma}$

and $\partial^2 \log(Z)/\partial(\boldsymbol{\mu})^2$ vary. EP performs the described operations for all the j factors $\tilde{f}_j(\theta_j)$.

After computing the new approximate distribution $q^{\text{new}}(\boldsymbol{\theta})$, we now have to update the approximated factor $\tilde{f}_j(\theta_j)$. Recall that the cavity distribution $q^{\setminus j}(\theta_j)$ is the joint distribution of the approximate factors except the approximated factor $\tilde{f}_j(\theta_j)$. In order to obtain the updated approximated factor $\tilde{f}_j(\theta_j)$ we just need to divide the new approximate distribution $q^{\text{new}}(\boldsymbol{\theta})$ by the cavity distribution $q^{\setminus j}(\boldsymbol{\theta})$:

$$\tilde{f}_j(\theta_j) = Z_j q^{new}(\boldsymbol{\theta})/\tilde{q}^{\setminus j}(\boldsymbol{\theta})\,. \tag{2.45}$$

Eq. (2.45) guarantees that the new approximate distribution $q^{\text{new}}(\boldsymbol{\theta})$ is proportional to the product of the approximate updated factor $\tilde{f}_j(\theta_j)$ and the cavity distribution $q^{\setminus j}(\boldsymbol{\theta})$, that is, $q^{\text{new}}(\boldsymbol{\theta}) \propto \tilde{f}_j(\theta_j) q^{\setminus j}(\boldsymbol{\theta})$. Another interesting result is that we have ensured that both the approximate $\tilde{f}_j(\theta_j)$ and the exact factor $f_j(\theta_j)$ integrate the same quantity with respect to the cavity distribution $q^{\setminus j}(\boldsymbol{\theta})$. These are all the steps that the inner EP loop performs to refine the approximate factors $\tilde{f}_j(\theta_j)$. Once these operations are done, in the outer loop, EP repeats the steps of the inner loop to refine the parameters of the approximate factors $\tilde{f}_j(\theta_j)$ iteratively until convergence. It is expected that the EP algorithm minimizes the KL divergence $\text{KL}(p(\boldsymbol{\theta})||q(\boldsymbol{\theta}))$ of the target $p(\boldsymbol{\theta})$ and proposed distribution $q(\boldsymbol{\theta})$. The normalization constant of $q(\boldsymbol{\theta})$, Z_q, approximates the normalization constant of $p(\boldsymbol{\theta})$, that can be seen as the marginal likelihood or model evidence. We summarize the steps performed by EP in Algorithm 2.

Input: Approximate factors $\tilde{f}_j(\theta_j)$, exact factors $f_j(\theta_j)$.
1: Uniformly initialize the parameters of the approximate factors $\tilde{f}_j(\theta_j)$.
2: Initialize the posterior approximation $\tilde{q}(\boldsymbol{\theta}) \approx \prod_j \tilde{f}_j(\theta_j)$.
while *Approximate factors* $\tilde{f}_j(\theta_j)$ *have not converged* **do**
  for $j = 1, 2, 3, \ldots, N$ **do**
    3: Compute the cavity distribution $\tilde{q}^{\setminus j}(\boldsymbol{\theta})$.
    4: Compute the tilted distribution $\hat{p}_j(\boldsymbol{\theta})$.
    5: Compute the approximate new distribution $q^{\text{new}}(\boldsymbol{\theta})$.
    6: Minimize the KL divergence between the tilted distribution $\hat{p}_j(\boldsymbol{\theta})$ and the approximate new distribution $q^{new}(\boldsymbol{\theta})$.
    7: Update the approximated factor as $\tilde{f}_j(\theta_j) = Z_j q^{new}(\boldsymbol{\theta})/\tilde{q}^{\setminus j}(\boldsymbol{\theta})$.
  end
end
8: Compute the approximate distribution $\tilde{q}(\boldsymbol{\theta}) = (1/Z)\prod_{j=1}^{N} \tilde{f}_j(\theta_j)$ and the model evidence $p(\mathcal{D}) \approx \int \tilde{q}(\boldsymbol{\theta}) d\boldsymbol{\theta}$.
Result: Approximate distribution $\tilde{q}(\boldsymbol{\theta})$ and model evidence $p(\mathcal{D})$.

Algorithm 2: Expectation propagation algorithm

2.6.3 Expectation Propagation in Practice

We now consider a particular case of EP application to clarify how EP can be used in practice. Let us suppose that you want to approximate the following probability distribution $p(\boldsymbol{\theta})$. This distribution can be expressed as a product of three exact factors.

These factors are $\mathbb{I}(\theta_1 > 0)$, $\mathbb{I}(\theta_2 > 0)$ and $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The distribution is hence given by:

$$p(\boldsymbol{\theta}) = \mathbb{I}(\theta_1 > 0)\mathbb{I}(\theta_2 > 0)\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \,, \tag{2.46}$$

where $\mathbb{I}(\theta_1 > 0)$ is an indicator function that takes value 1 when the condition $\theta_1 > 0$ is true and 0 otherwise. $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is a bivariate Gaussian distribution. The product of these factors generates a truncated Gaussian distribution. This truncated Gaussian distribution is shown in Figure 2.12. In this particular example, EP will approximate the indicator functions $\mathbb{I}(\theta_1 > 0)$ and $\mathbb{I}(\theta_2 > 0)$ with un-normalized univariate Gaussian distributions $\tilde{\mathcal{N}}(\theta_1|\mu_1, \sigma_1)$ and $\tilde{\mathcal{N}}(\theta_2|\mu_2, \sigma_2)$. Indicator functions $\mathbb{I}(\theta_1 > 0)$ and $\mathbb{I}(\theta_2 > 0)$ do not belong to the exponential family $\mathcal{F}$. Hence, we cannot analytically compute a posterior distribution of the parameters given some collected data $p(\boldsymbol{\theta}|\mathcal{D})$. By performing EP, we approximate the indicator function $\mathbb{I}(\theta_1 > 0)$ and $\mathbb{I}(\theta_2 > 0)$ by univariate un-normalized Gaussian distributions $\tilde{\mathcal{N}}(\theta_1|\mu_1, \sigma_1)$ and $\tilde{\mathcal{N}}(\theta_2|\mu_2, \sigma_2)$. Let us describe now the steps performed EP to approximate the indicator functions $\mathbb{I}(\theta_1 > 0)$ and $\mathbb{I}(\theta_2 > 0)$ by univariate un-normalized Gaussian distributions $\tilde{\mathcal{N}}(\theta_1|\mu_1, \sigma_1)$ and $\tilde{\mathcal{N}}(\theta_2|\mu_2, \sigma_2)$. Recall that the steps that we are describing in this example are the same that the ones of Algorithm 2.

Let $f_1(\theta_1) = \mathbb{I}(\theta_1 > 0)$ and $f_2(\theta_2) = \mathbb{I}(\theta_2 > 0)$ be the exact factors that we want to approximate. Let $\tilde{f}_1(\theta_1) = \tilde{\mathcal{N}}(\theta_1|\mu_1, \sigma_1)$ and $\tilde{f}_2(\theta_2) = \tilde{\mathcal{N}}(\theta_2|\mu_2, \sigma_2)$ be the approximate factors corresponding to the exact factors $f_1(\theta_1) = \mathbb{I}(\theta_1 > 0)$ and $f_2(\theta_2) = \mathbb{I}(\theta_2 > 0)$. The first step that EP does is to initialize the factors μ_1, σ_1, μ_2 and σ_2 uniformly. For instance, we can set $\mu_1 = 0\,, \sigma_1 = \infty\,, \mu_2 = 0$ and $\sigma_2 = \infty$. The second step of Algorithm 2 is to compute the initial posterior approximation $q(\boldsymbol{\theta})$. The prior, in this case, is Gaussian, $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. As the Gaussian distribution belongs to the exponential family, there is no need to approximate it. As we can observe in step 2 of Algorithm 2, this distribution $q(\boldsymbol{\theta})$ is simply given by the product of factors $\tilde{q}(\boldsymbol{\theta}) \approx \prod_j \tilde{f}_j(\theta_j)$ and the prior $p(\boldsymbol{\theta})$, that is Gaussian. In this example, we can compute $q(\boldsymbol{\theta})$ as:

$$q(\boldsymbol{\theta}) = (1/Z)\tilde{f}_1(\theta_1)\tilde{f}_2(\theta_2)p(\boldsymbol{\theta}) \,, \tag{2.47}$$

where Z is the normalization constant. We can use the expression for the product of Gaussian distributions to analytically compute $q(\boldsymbol{\theta})$. See Appendix A for details. We can alternatively convert the Gaussian parameters to natural parameters and simply sum the parameters of the distributions $\tilde{f}_1(\theta_1)$ and $\tilde{f}_2(\theta_2)$. After performing the initial EP steps, we now enter the double loop of the algorithm. The outer loop will continue until the factors $\tilde{f}_1(\theta_1)$ and $\tilde{f}_2(\theta_2)$ meet a convergence criterion. In the inner loop, we iterate over the $\tilde{f}_1(\theta_1)$ and $\tilde{f}_2(\theta_2)$ factors, refining them. In this case, we first refine $\tilde{f}_1(\theta_1)$ and then $\tilde{f}_2(\theta_2)$. The third step of Algorithm 2 is to compute the cavity distribution of the first factor $q^{\setminus 1}(\boldsymbol{\theta})$. This distribution is given by:

$$q^{\setminus 1}(\boldsymbol{\theta}) = \frac{1}{Z^{\setminus 1}} \frac{q(\boldsymbol{\theta})}{\tilde{f}_1(\theta_1)} \,, \tag{2.48}$$

where $Z^{\setminus 1}$ is the normalization constant. We can analytically compute $q^{\setminus 1}(\boldsymbol{\theta})$ by using the division of Gaussians analytical expression. See Appendix A for details. We can alternatively convert the Gaussian parameters to natural parameters and just substract the parameters of the distributions $\tilde{f}_1(\theta_1)$ to the ones of the distribution $\tilde{f}_2(\theta_2)$. Let $\boldsymbol{\mu}^{\setminus 1}$ and $\boldsymbol{\Sigma}^{\setminus 1}$ be the Gaussian parameters of the unnormalized cavity distribution $q^{\setminus 1}(\boldsymbol{\theta})$ and

μ_1, σ_1, μ_2 and σ_2 be the Gaussian parameters of $\tilde{f}_1(\theta_1)$ and $\tilde{f}_2(\theta_2)$. Then:

$$\boldsymbol{\Sigma}^{\backslash 1} = (\sigma_1^{-1} - \sigma_2^{-1})^{-1}\,, \tag{2.49}$$

$$\boldsymbol{\mu}^{\backslash 1} = \boldsymbol{\Sigma}^{\backslash 1}(\sigma_1^{-1}\mu_1 - \sigma_2^{-1}\mu_2)\,. \tag{2.50}$$

The normalization constant $Z^{\backslash 1}$ of the cavity distribution is equal to:

$$Z^{\backslash 1} = (2\pi)\frac{|\boldsymbol{\Sigma}^{\backslash 1}|^{0.5}|\sigma_2|^{0.5}}{|\sigma_1|^{0.5}}\exp\{-0.5(\mu_1\sigma_1^{-1}\mu_1 - \mu_2\sigma_2^{-1}\mu_2 + \boldsymbol{\mu}^{\backslash 1T}\boldsymbol{\Sigma}^{\backslash 1-1}\boldsymbol{\mu}^{\backslash 1})\}\,. \tag{2.51}$$

The fourth step of Algorithm 2 corresponds to the computation of the tilted distribution $\hat{p}_1(\boldsymbol{\theta})$. This distribution is given by the product of the exact factor $f_1(\theta_1)$, that corresponds to the factor $\tilde{f}_1(\theta_1)$ that we are refining, multiplied by the cavity distribution $q^{\backslash 1}(\boldsymbol{\theta})$. The tilted distribution is computed as:

$$\hat{p}_1(\boldsymbol{\theta}) = \frac{1}{Z_1} f_1(\theta_1) q^{\backslash 1}(\boldsymbol{\theta})\,, \tag{2.52}$$

where the normalization constant Z_1 would be given by the following expression, where we use Eq. (2.41), setting $t(\theta_1) = \mathbb{I}(\theta_1 > 0)$ and $\Phi(\cdot)$ represents the standard Gaussian cumulative distribution function:

$$\begin{aligned} Z_1 &= \int f_1(\theta_1) q^{\backslash 1}(\boldsymbol{\theta}) d\theta_1 d\theta_2 \\ &= \int \mathbb{I}(\theta_1 > 0)\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}^{\backslash 1}, \boldsymbol{\Sigma}^{\backslash 1}) d\theta_1 d\theta_2 = \Phi\left(\frac{\boldsymbol{\mu}^{\backslash 1}}{\sqrt{\boldsymbol{\Sigma}^{\backslash 1}}}\right)\,. \end{aligned} \tag{2.53}$$

We now compute the new approximate distribution $q^{\text{new}}(\boldsymbol{\theta})$, as the fifth step of Algorithm 2 states. Recall that as the prior is Gaussian, no extra operation involving the prior is needed. This distribution is given by the product of the approximate factors:

$$q^{\text{new}}(\boldsymbol{\theta}) = (1/Z)\tilde{f}_1(\theta_1)\tilde{f}_2(\theta_2)p(\boldsymbol{\theta})\,. \tag{2.54}$$

$q^{\text{new}}(\boldsymbol{\theta})$ can be analytically computed by the product of Gaussian distributions analytical expression. Having computed all the necessary distributions we can move forward to the sixth step of Algorithm 2. In this particular example, this step requires us to minimize the KL divergence between the tilted distribution $\hat{p}_1(\boldsymbol{\theta})$ and the new approximate distribution $q^{\text{new}}(\boldsymbol{\theta})$. We will extract the updated parameters of the factor that we are refining, $\tilde{f}_1(\theta_1)$, from the ones of the approximate distribution $q^{\text{new}}(\boldsymbol{\theta})$ once it minimizes the KL divergence with respect to the tilted distribution $\hat{p}_1(\boldsymbol{\theta})$. Minimizing the KL divergence of $\hat{p}_1(\boldsymbol{\theta})$ and $q^{\text{new}}(\boldsymbol{\theta})$ is equivalent to matching their moments. To match the moments between the tilted $\hat{p}_1(\boldsymbol{\theta})$ and the new approximate distribution $q^{\text{new}}(\boldsymbol{\theta})$, we need the partial derivatives of the tilted distribution normalization constant Z_1 with respect to the parameters μ_2 and σ_2 as we can see in Eq. (A.47). As we have an analytical closed-form solution for the normalization constant Z_1, we can analytically compute the gradient of the normalization constant $Z^{\backslash 1}$ given by the partial derivatives of $Z^{\backslash 1}$ with respect to the parameters μ_2 and σ_2:

$$\mathbb{E}_{\hat{p}_1}[\theta_1] = \mu_1^{\backslash 1} + \sigma_1^{\backslash 1}\frac{\partial \log Z_1}{\partial \mu_1^{\backslash 1}} = \mu_1^{\backslash 1} + \sigma_1^{\backslash 1}\frac{\mathcal{N}(\theta_1|\frac{\mu_1^{\backslash 1}}{\sqrt{\sigma_1^{\backslash 1}}})[0,1]}{\Phi(\theta_1|\frac{\mu_1^{\backslash 1}}{\sqrt{\sigma_1^{\backslash 1}}})}\,, \tag{2.55}$$

$$\mathbb{E}_{\hat{p}_1}[\theta_1^2] - \mathbb{E}_{\hat{p}_1}[\theta_1]^2 = \sigma_1^{\backslash 1} - \sigma_1^{\backslash 1}[(\frac{\partial \log Z_1}{\partial \mu_1^{\backslash 1}})^2 - 2\frac{\partial \log Z_1}{\partial \sigma_1^{\backslash 1}}]\sigma_1^{\backslash 1} =$$

$$\sigma_1^{\backslash 1} - \sigma_1^{\backslash 1}[(\frac{\mathcal{N}(\theta_1|\frac{\mu_1^{\backslash 1}}{\sqrt{\sigma_1^{\backslash 1}}}[0,1])}{\Phi(x_1|\frac{\mu_1^{\backslash 1}}{\sqrt{\sigma_1^{\backslash 1}}})}\frac{1}{\sqrt{\sigma_1^{\backslash 1}}})^2 - 2\frac{\mathcal{N}(\theta_1|\frac{\mu_1^{\backslash 1}}{\sqrt{\sigma_1^{\backslash 1}}}[0,1])}{\Phi(\theta_1|\frac{\mu_1^{\backslash 1}}{\sqrt{\sigma_1^{\backslash 1}}})}\frac{-\mu_1^{\backslash 1}}{2(\sigma_1^{\backslash 1})^{\frac{3}{2}}}]\sigma_1^{\backslash 1}\,. \tag{2.56}$$

Via Eqs. (2.55) and (2.56) we analytically compute the new parameters of the new

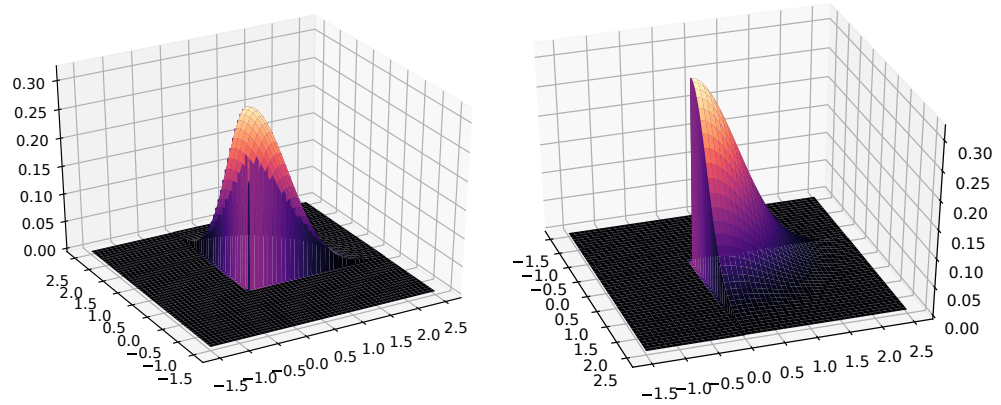

FIGURE 2.12: Truncated bivariate Gaussian distribution that is going to be approximated by a product of factors given by EP.

approximate distribution $q_{new}(\boldsymbol{\theta})$. We can now update the parameters of the approximate $\tilde{f}_1(\theta_1)$ factor by extracting them from the new approximate distribution $q_{new}(\boldsymbol{\theta})$, which is the step 7 of Algorithm 2. In order to do so, we extract it from the new approximate distribution $q_{new}(\boldsymbol{\theta})$. This can easily be done by the following expression, using again the analytical expressions of the division of Gaussian distributions.

$$\tilde{f}_1(\theta_1) = \frac{1}{Z_1}\frac{q^{new}(\boldsymbol{\theta})}{q^{\backslash 1}(\boldsymbol{\theta})}\,. \tag{2.57}$$

Now that we have refined the factor $\tilde{f}_1(\theta_1)$, EP iterates through the other factors of the approximate distribution $q(\boldsymbol{\theta})$ in its inner loop, repeating steps 3 to 7 of Algorithm 2. In this particular example, EP will refine $\tilde{f}_2(\theta_2)$ after having refined $\tilde{f}_1(\theta_1)$. We can analogously use the same equations as the ones that we have presented for the refinement of factor $\tilde{f}_2(\theta_2)$. EP continues refining the factors $\tilde{f}_1(\theta_1)$ and $\tilde{f}_2(\theta_2)$ of the approximate distribution $q(\boldsymbol{\theta})$ iteratively in its outer loop until it meets a convergence criterion.

2.7 Conclusions

In this chapter, we have covered the fundamentals of GPs. We have studied the posterior and predictive distribution of GPs, the most popular covariance functions and how to estimate the hyper-parameters of these covariances functions. All this information is valuable to better understand BO, which will be described in the following chapter. We have also shown the details of other alternative probabilistic surrogate models that

can be used on the BO framework. Concretely, we have reviewed random forests, T-Student processes, sparse Gaussian processes, deep Gaussian processes and Bayesian neural networks. For each probabilistic surrogate model, we have covered its main advantages and disadvantages with respect to the GPs. No model is the best for all the potential problems where BO can be applied. Hence, it is important to know the basics of them to choose the more adequate one in each situation. Finally, we have illustrated the EP approximate inference algorithm. This algorithm can be used to approximate non-Gaussian factors with exponential family members such as un-normalized Gaussian distributions. By approximating non-Gaussian likelihoods by these approximate factors, we can analytically compute approximate posterior and predictive distributions that otherwise would be intractable to compute. EP will be used in the following chapters to perform the mentioned approximation.

Chapter 3

Fundamentals Of Bayesian Optimization

A black-box is a function that satisfies three properties. First, its analytical expression is unknown. Hence, its gradients are not accesible. Second, it is very expensive to evaluate. Finally, the function evaluations are potentially noisy. For example, the estimation of the generalization error of machine learning algorithms is considered to be a black-box function. Optimizing such a function with respect to the hyper-parameters is a challenging problem. It requires an algorithm that is able to optimize a black-box without using gradients, in a small number of steps, and dealing with noise in the evaluations. This chapter introduces the fundamental concepts of Bayesian optimization (BO). BO is a class of methods that successfully optimize black-box functions. In order to do so, BO uses a probabilistic surrogate model, typically a Gaussian process (GP), of the objective function. Using the GP, we can compute a predictive distribution of the objective in regions of the space where it has not been evaluated yet. Based on the information given by the GP predictive distribution, BO computes, at each iteration, an acquisition function. The acquisition function estimates, for every input space point, the expected utility of evaluating the objective there. The point whose value maximizes the acquisition function is suggested for evaluation at each iteration. That point maximizes the trade-off between exploration of unknown areas and exploitation of promising solutions. Therefore, BO uses the acquisition function as an oracle to guide the search for the optimum. The GP is updated with the evaluation result of the suggested point. After conditioning the GP to the new evaluation performed at the chosen location, the BO method repeats the described operations iteratively, until a budget of evaluations is consumed. At the end of the process, it gives the final recommendation. This point can be the one whose evaluation has the best observed value or the point that optimizes the GP predictive mean. This chapter describes BO in detail. It also introduces the most popular acquisition functions, with special emphasis on information theory acquisition functions.

3.1 Introduction

The purpose of this section is to settle the foundations of BO. As it was described in Chapter 1, the purpose of BO is to retrieve the extremum $\mathbf{x}^\star$ of a black-box function $f(\mathbf{x})$ where $\mathbf{x} \in \mathcal{X}$ and $\mathcal{X}$ is the input space where $f(\mathbf{x})$ can be evaluated. Formally, we

seek to obtain $\mathbf{x}^\star$ such that,

$$\mathbf{x}^\star = \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})\,, \tag{3.1}$$

assuming minimization. We can define a BO method by the following tuple

$$\mathcal{A} = (\mathcal{M}, \alpha(\cdot), p(f(\mathbf{x})|\,\mathcal{D}))\,, \tag{3.2}$$

where $f(\mathbf{x})$ is the black-box or objective function that we want to optimize, $\mathcal{M}$ is the chosen surrogate model, $\alpha(\cdot)$ is an acquisition function, $p(f(\mathbf{x})|\mathcal{D})$ is a predictive distribution of the evaluation of $\mathbf{x}$ and $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., t\}$ is the dataset of previous observations at iteration t. The probabilistic surrogate model is typically a GP. $p(f(\mathbf{x})|\mathcal{D})$ is the predictive distribution of the evaluation $f(\mathbf{x})$ of a point $\mathbf{x}$ given that the model $\mathcal{M}$ (typically a GP) has been conditioned on $\mathcal{D}$. The acquisition function receives as an input the GP predictive distribution $p(f(\mathbf{x})|\mathcal{D})$ at a point $\mathbf{x}$. Hence, we could represent the acquisition function as $\alpha(\mathbf{x}; p(f(\mathbf{x})|\mathcal{D}))$. Since this notation may be confusing, we will abbreviate the notation of the acquisition function $\alpha(\mathbf{x}; p(f(\mathbf{x})|\mathcal{D}))$ by $\alpha(\mathbf{x})$. The acquisition function $\alpha(\mathbf{x})$ represents the expected utility of evaluating the objective at each candidate point $\mathbf{x} \in \mathcal{X}$, with the goal of solving the optimization problem. $\mathcal{M}$ and $\alpha(\mathbf{x})$ are used to search the extremum $\mathbf{x}^\star$ of a black-box function $f(\mathbf{x})$ in $\mathcal{X}$. First, the surrogate model $\mathcal{M}$ encodes assumptions about the properties of the black-box function $f(\mathbf{x})$, such as smoothness or stationarity. In particular, GPs encode those assumptions via the hyper-parameters of the covariance functions $k(\mathbf{x}, \mathbf{x}')$. Depending on the choice of surrogate model $\mathcal{M}$, we can encode a different set of assumptions about the black-box function $f(\mathbf{x})$. See section 2.3 of Chapter 2 for more details. On the other hand, different choices for the acquistion function $\alpha(\mathbf{x})$ encode a different exploration and exploitation trade-off. This trade-off basically consists in deciding whether to explore new unknown areas with high uncertainty value or to exploit promising known areas with low prediction values according to the GP predictive distribution $p(f(\mathbf{x})|\mathcal{D})$. Both actions are interesting and required for the search of the optimum of the objective function $f(\mathbf{x})$. We will describe this trade-off in detail in Section 3.3.1 of this chapter. In general, some acquisition functions $\alpha(\mathbf{x})$ favour more exploration and viceversa. No single criteria is the best for optimizing every possible objective function $f(\mathbf{x})$ (Ho and Pepyne, 2002). Hence, depending on our prior knowledge about $f(\mathbf{x})$, we may choose an appropiate acquisition function $\alpha(\mathbf{x})$ and surrogate model $\mathcal{M}$. These ideas are developed in more detail in a comprehensive introduction to BO given by the tutorial of Brochu et al. (2010).

There are several real applications where black-boxes need to be optimized. Hence, BO has been a very active research area. The most popular application of BO in the machine learning community is the tuning of the hyper-parameters of machine learning algorithms (Snoek et al., 2012). BO can also be used for improved learning of the structure of proabilistic graphical models (Córdoba et al., 2018). The potential of BO has been shown by the enhancement of the AlphaGo system (Chen et al., 2018; Wang et al., 2016). AlphaGo is an automatic system to play Go that has defeated the Go world champion. AlphaGo played the Go game via a Monte Carlo tree search (Chaslot, 2010). BO tuned the Monte Carlo tree search hyper-parameters (Chen et al., 2018). By doing so, AlphaGo improved its win-rate from 50 % to 66.5 % in self-play games. Eventually, the version of AlphaGo enhanced by BO was deployed in the final match against Lee Sedol, a prestigious Go player. Not only can BO be used in complex problems such as Go, but in subjective tasks such as elaborating better recipes for cookies (Garrido-Merchán and Albarca-Molina, 2018; Kochanski et al., 2017). BO has also been applied on other fields such as renewable energies, finance, real-time control systems or chemical design

(Baheri et al., 2017; Cornejo-Bueno et al., 2018; Gonzalvez et al., 2019; Griffiths and Hernández-Lobato, 2020). In particular, regarding renewable energies, BO has been applied to optimize the hyper-parameters of a genetic algorithm that tuned a extreme learning machine that predicts the significant wave height and the wave energy flux at a goal marine structure facility (Cornejo-Bueno et al., 2018). In particular, this application will be described in detail in Chapter 7. In the finance sector, BO has been applied to the portfolio optimization problem in the context of quantitative asset management. In particular, it has been applied to obtain the optimal hyper-parameters of a trend-following strategy that balances risk and return of investing in a set of assets (Gonzalvez et al., 2019). BO has also been used to the real-time altitude optimization of an Airborne Wind Energy system in order to maximize net energy production (Baheri et al., 2017). In particular, BO uses the GP predictive distribution of the estimation of the wind speed to determine the best subsequent operating altitude of the Airborne Wind Energy system in real-time. By doing so, net energy production of the Airborne Wind Energy system is maximized. Finally, BO has been applied to automatically generate novel molecules with optimized properties (Griffiths and Hernández-Lobato, 2020). In particular, BO is used over the latent space of a variational autoencoder in order to search for regions that generate novel molecules (Gómez-Bombarelli et al., 2018). However, BO tends to search in latent space areas that lie far away from the data on which the variational autoencoder has been trained, producing invalid molecular structures. To circumvent this pathology, Griffiths and Hernández-Lobato (2020) reframe the problem as a constrained BO problem, mitigating the described pathology. We describe in detail constrained and multi-objective BO in Section 3.4 of this chapter.

The literature provides vast empirical evidence of the better performance of BO compared with random search for the tuning of the hyper-parameters of machine learning algorithms (Fernández-Sánchez et al., 2020; Garrido-Merchán and Hernández-Lobato, 2019b, 2020; Snoek et al., 2012). Random search is a procedure that places a uniform distribution over all the input space $\mathcal{X}$ and samples randomly a point $\mathbf{x}$ from it iteratively. Hence, random search makes a hypothesis about the objective function $f(\mathbf{x})$. Either all the points that belong to the input space $\mathcal{X}$ have the same probability of being the optimum or the objective function $f(\mathbf{x})$ is completely random and there is no pattern to be learnt. In other words, that the image space $\mathcal{Y}$ that results from the evaluation of the input space $\mathcal{X}$ by the objective function $f(\mathbf{x})$ is not smooth, so the values that it takes in the evaluated points are not useful to predict unknown values, or completely random, *i.e.*, there is nothing to learn. As there is nothing to learn, there is no need to place a surrogate model on the objective function $f(\mathbf{x})$. Notwithstanding, real applications such as the ones cited in the previous paragraph show that we can make some assumptions about the objective function $f(\mathbf{x})$ that GPs also assume. Hence, we can learn some features about the objective function $f(\mathbf{x})$ that the GP can capture via its covariance function and that can help us to guide the search more efficiently that just by random search. Several reasons explain why BO outperforms random search. First, random search does not make any assumption about the objective function $f(\mathbf{x})$. It simply explores the input space $\mathcal{X}$ randomly. On the other hand, the surrogate model $\mathcal{M}$ encodes assumptions about the objective function $f(\mathbf{x})$. For example, the objective function $f(\mathbf{x})$ being smooth. That information is used by the surrogate model $\mathcal{M}$ for the computation of the predictive distribution $p(f(\mathbf{x})|\mathcal{D})$. Then, the acquisition function $\alpha(\mathbf{x})$ uses the information of the predictive distribution $p(f(\mathbf{x})|\mathcal{D})$ to suggest a new point $\mathbf{x}$ to evaluate the objective function $f(\mathbf{x})$. The acquisition function $\alpha(\mathbf{x})$ uses the hypotheses made about the objective function $f(\mathbf{x})$ to avoid the evaluation of unpromising areas

according to those hypotheses. For example, if the function takes high values in a space region, at various points, the minimum is unlikely to be there. Therefore, those areas will not be evaluated. If the hypotheses are correct, useless evaluations will be saved. Hence, if our assumptions about the objective function $f(\mathbf{x})$ are correct, BO has more information to determine whether a given point $\mathbf{x}$ is useful to be evaluated than random search, that basically does not use any information to decide what point should be evaluated at each iteration. In practice, one of these assumptions is the smoothness of the objective function $f(\mathbf{x})$. Intuitively, let the distance of an evaluated point $\mathbf{x}$ with respect to other point $\mathbf{x}'$ be very small and the evaluation of that point $\mathbf{x}$ be a value far from the optimal value. We would not evaluate $\mathbf{x}'$, since we know that it is very likely that its evaluation $f(\mathbf{x}')$ is going to be more or less as bad as $f(\mathbf{x})$. BO will behave in a similar way. In particular, in the case of a GP the covariance function $k(\mathbf{x}, \mathbf{x}')$ and its hyper-parameters encode that assumption. On the other hand, random search will evaluate that point $\mathbf{x}'$ with the same probability as any other point. Moreover, the acquisition function $\alpha(\mathbf{x})$ of BO can enforce the exploitation of areas where we have retrieved good evaluations. Given the information encoded in the model $\mathcal{M}$ and the preferences encoded in the acquisition function $\alpha(\mathbf{x})$, BO determines which regions of the input space $\mathcal{X}$ are interesting to evaluate and which are worthless. Computing the predictive distribution $p(f(\mathbf{x})|\mathcal{D})$ and optimizing the acquisition function $\alpha(\mathbf{x})$ is costly. Nevertheless, BO assumes that the cost of evaluating the objective function $f(\mathbf{x})$ is much more expensive than the cost of computing the predictive distribution $p(f(\mathbf{x})|\mathcal{D})$ and optimizing the acquisition function $\alpha(\mathbf{x})$. In other words, the cost of suggesting a point $\mathbf{x}$ with BO to evaluate the objective function $f(\mathbf{x})$ is negligible compared to the cost of evaluating the objective function $f(\mathbf{x})$. Therefore, spending a little time carefully thinking about the following evaluation $f(\mathbf{x})$ is worth in the case that the objective function $f(\mathbf{x})$ is very expensive.

This chapter describes how BO works in detail, its main components and the most popular software libraries that implement BO. BO uses a probabilistic surrogate model, typically a GP, as a source of information to make decisions. If the reader has skipped Chapter 2 and it is not familiarized with the GP probabilistic model, the reader is encouraged to return to Chapter 2. BO also computes acquisition functions. Acquisition functions use the information given by the GP surrogate model to guide the search. Several acquisition functions have been proposed in the literature. We will describe the most relevant ones from our point of view. Some acquisition functions use information theory concepts. Therefore, we also include a subsection that explains the basics of information theory. Finally, we also cover the most popular BO free software libraries in our opinion.

3.2 Bayesian Optimization

This section describes BO in depth. BO is a class of methods used to retrieve the extremum $\mathbf{x}^\star$ of a black-box function $f(\mathbf{x})$: $\mathbf{x}^\star = \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$, assuming minimization. In order to perform this task, it depends on a probabilistic surrogate model of the black-box function $f(\mathbf{x})$, which is typically a GP. BO uses the GP predictive distribution $p(f(\mathbf{x})|\mathcal{D})$ to generate a response surface that measures the expected utility of evaluating every point $\mathbf{x} \in \mathcal{X}$. This response surface is called an acquisition function, $\alpha(\mathbf{x})$. BO uses the acquisition function $\alpha(\mathbf{x})$ to guide the search of the minimizer $\mathbf{x}^\star$ of the black-box function $f(\mathbf{x})$. In particular, the next point $\mathbf{x}$ to be evaluated is the one that maximizes the acquisition function $\alpha(\mathbf{x})$, that can be optimized inexpensively. An acquisition function

$\alpha(\mathbf{x})$ represents an exploration-exploitation trade-off. Specifically, the acquisition function $\alpha(\mathbf{x})$ favors exploration in the sense that it is high in areas where no point $\mathbf{x}$ has been evaluated before. These areas may contain values near the optimum value. It also needs to perform exploitation, *i.e.*, the acquisition function $\alpha(\mathbf{x})$ favors the evaluation of points that are near of good evaluations. Concretely, we expect that the new evaluations are near the optimum as we assume that the function is smooth. We will justify and develop this idea in Section 3.3.1 of this chapter.

BO is an iterative methodology. In each iteration, it performs a sequence of steps. The number of iterations is delimited by the budget that an user can afford to evaluate the black-box, which is assumed to be costly. For example, the user may only be able to test 50 configurations of a machine learning algorithm. In this case, the number of BO iterations would be 50. At each iteration t, a point $\mathbf{x}_t$ is suggested. In this particular example, the point $\mathbf{x}_t$ represents a configuration of the machine learning algorithm, *i.e.*, a set of values of the machine learning algorithm hyper-parameters. This point, $\mathbf{x}_t$, is the maximizer of the acquisition function $\alpha(\mathbf{x})$:

$$\mathbf{x}_t = \arg\max_{\mathbf{x}\in\mathcal{X}} \alpha_t(\mathbf{x})\,, \tag{3.3}$$

where $\alpha_t(\mathbf{x})$ represents the acquisition function built at iteration t. Therefore, this point $\mathbf{x}_t$ is used to evaluate the black-box function $f(\mathbf{x})$. Specifically, we obtain an observation $y_t = f(\mathbf{x}_t) + \epsilon$ by evaluating the black-box function $f(\mathbf{x})$ at $\mathbf{x}_t$. Note from the previous expression that we do not have access to the latent value $f(\mathbf{x})$ of the black-box function, as we only observe y which is corrupted by noise ϵ. In particular, this noise is typically modelled as additive Gaussian noise over the latent function value: $\epsilon \sim \mathcal{N}(0, \sigma^2)$, although, it can be other type of noise. By doing this process iteratively, we obtain a set $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., t\}$ of observations $(\mathbf{x}_t, y_t)$ at iteration t that we define as dataset $\mathcal{D}$. At every iteration t, a new tuple $(\mathbf{x}_t, y_t)$ is added to the dataset $\mathcal{D}$. The GP is then fitted to the augmented dataset $\mathcal{D}$, acquiring new knowledge about the potential values of the objective function $f(\mathbf{x})$. The process is repeated iteratively until the budget of evaluations T is consumed. Finally, BO recommends the point $\mathbf{x}$ whose observation value y has been the best as the solution of the problem. That is, the point $\mathbf{x}$ whose observation value y minimizes the evaluation of the black-box function $f(\mathbf{x})$. It can also recommend as a solution the point $\mathbf{x}$ that minimizes the GP prediction of the objective function $f(\mathbf{x})$. Ideally, we would like to have as many iterations T as possible, to better explore the input space $\mathcal{X}$. In practice, BO is launched for 10 to 500 iterations because, generally, in practice we cannot afford to evaluate the objective function $f(\mathbf{x})$ more times. Recall that the evaluation of the objective function $f(\mathbf{x})$ is a very costly process. Therefore, the budget of evaluations of the objective function $f(\mathbf{x})$ is limited and the number of evaluations are usually limited to the mentioned interval, 10 to 500. Moreover, the complexity of the GP is cubic on the number of observations $\mathcal{O}(t^3)$, where t is the number of observations.

The acquisition function $\alpha(\mathbf{x})$ is generally not difficult to maximize. In particular, we can compute the gradient $\nabla_{\mathbf{x}}\alpha(\mathbf{x})$ of the acquisition function and use it for its optimization. We can compute the gradient $\nabla_{\mathbf{x}}\alpha(\mathbf{x})$ because the acquisition function $\alpha(\mathbf{x})$ is cheap to evaluate, as it is only based on the GP predictive distribution $p(f(\mathbf{x})|\mathcal{D}))$. It is important to observe that we do not need to evaluate the black-box $f(\mathbf{x})$ to compute the acquisition function $\alpha(\mathbf{x})$. Recall that the bottleneck of the optimization process is the evaluation of the objective function $f(\mathbf{x})$. Therefore, as it has been said before, the evaluation of the acquisition function $\alpha(\mathbf{x})$ is a cheap process. Hence, the gradient of the

acquisition function $\nabla_{\mathbf{x}}\alpha(\mathbf{x})$ is also cheap to evaluate, which makes the optimization of the acquisition function a feasible process that is significantly cheaper than evaluating the black-box. Therefore, the computational time of finding the next point to evaluate can be considered negligible. Under these circumstances, when the objective function $f(\mathbf{x})$ is costly to evaluate, it is worth to spend a few seconds thinking carefully which point to evaluate next.

The acquisition function $\alpha(\mathbf{x})$ will only have as many dimensions D as dimensions has the black-box being optimized $f(\mathbf{x})$. In particular, the number of dimensions is typically small in practice. We expect that, for high-dimensionality, BO has problems. In particular, if the dimensionality of the input space $\mathcal{X}$ is high, it is difficult to extrapolate and make predictions of the objective function $f(\mathbf{x})$ since the number of evaluations is small and the input space is huge. Therefore, the model is not expected to be very useful and BO does not significantly improve the results given by a random search. Although, there exist several approaches for high-dimensional BO (Li et al., 2018; Rana et al., 2017; Wang et al., 2013).

There are several acquisition functions $\alpha(\mathbf{x})$ that have been proposed. Depending on the choice of acquisition function $\alpha(\mathbf{x})$, we can also obtain its gradient $\nabla_{\mathbf{x}}\alpha(\mathbf{x})$. If an analytical expression for the gradient $\nabla_{\mathbf{x}}\alpha(\mathbf{x})$ cannot be computed, we can obtain the gradient $\nabla_{\mathbf{x}}\alpha(\mathbf{x})$ via automatic differentiation (Paszke et al., 2017) or we can approximate it by using finite differences. Although, it is generally more costly to use automatic differentiation than computing the analytical expression of the gradient $\nabla_{\mathbf{x}}\alpha(\mathbf{x})$. In most problems we can solve the optimization of the acquisition function $\alpha(\mathbf{x})$ by combining a grid search with a local optimizer such as L-BFGS-B (Zhu et al., 1997). A local optimizer uses an initial point $\mathbf{x} \in \mathcal{X}$, the bounds of the input space $\mathcal{X}$ and the gradient $\nabla_{\mathbf{x}}f(\mathbf{x})$ of a function $f(\mathbf{x})$ to obtain a local optima. Often, L-BFGS-B is initialized at the best point given by the grid search. Then, L-BFGS is used to refine the result and to find a local optimum of the acquisition function.

We can gain an intuition of why BO is useful to optimize black-boxes through a graphical example. In particular, we can see in Figure 3.1 an example of the steps performed by BO. In this figure, the acquisition function $\alpha(\mathbf{x})$ is plotted on green, the GP posterior distribution on blue, observations are represented as black dots and the latest observation is represented as a red dot. The ground truth is plotted as a dashed black line, the GP posterior mean as a continuous black line and the GP posterior uncertainty of its prediction as a blue area, surrounding the mean. The GP posterior uncertainty represents one standard deviation of the predictive distribution around the mean. We can see how the GP predictive distribution $p(f(\mathbf{x})|\mathcal{D})$ and the acquisition function $\alpha(\mathbf{x})$ shapes change at each iteration. In the first figure, we can see the posterior distribution of a GP fitted to 3 observations. The acquisition function $\alpha(\mathbf{x})$ will take higher values in areas with high uncertainty and promising predictions. From a practical point of view, the only interesting point of the acquisition function $\alpha(\mathbf{x})$ is its maximizer $\mathbf{x}_t$, represented by an inverted red triangle. In particular, in the next iteration, BO evaluates the black-box function at that point $y_t = f(\mathbf{x}_t) + \epsilon$. Recall that the noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$ can be modelled as additive Gaussian noive over the latent function $f(\mathbf{x}_t)$. After this, BO augments the dataset $\mathcal{D}$ of previous observations with the new evaluation $(\mathbf{x}_t, y_t)$. Then, the GP is fitted to the augmented dataset $\mathcal{D}$, as it is shown in the figure of the middle. We can see how, in the point $\mathbf{x}_t$ and in its neighbourhood, the acquisition function $\alpha(\mathbf{x})$ now has a low value. This happens because there is now low uncertainty in that region. Points $\mathbf{x}$ surrounding an evaluation $\mathbf{x}_t$ will have a high covariance function value $k(\mathbf{x}, \mathbf{x}_t)$. This will make the uncertainty of the prediction, y, low, making the acquisition

function value $\alpha(\mathbf{x})$ lower than in points that are located far away from an evaluation. At the beggining of the optimization process, the first candidate point $\mathbf{x}_1$ at which to evaluate the objective function $f(\mathbf{x})$ can be chosen at random. Then, BO repeats the mentioned steps iteratively. In that way, the acquisition function $\alpha(\mathbf{x})$ is recomputed, giving a new maximizer $\mathbf{x}_t$, where t represents the index of the iteration. After the optimization of the acquisition function $\alpha(\mathbf{x})$, the black-box function $f(\mathbf{x}_t)$ is evaluated at the maximizer $\mathbf{x}_t$. When the computational budget of T evaluations is consumed, we can retrieve a final recommendation $\mathbf{x}^\star$. Generally, the final recommendation can just be the point $\mathbf{x}_t$ associated with the best observation value y_t retrieved by BO. Another possible recommendation point is the one whose value optimizes the GP predictive mean of the objective function $f(\mathbf{x})$. Algorithm 3 summarizes the steps that have been described in previous paragraphs.

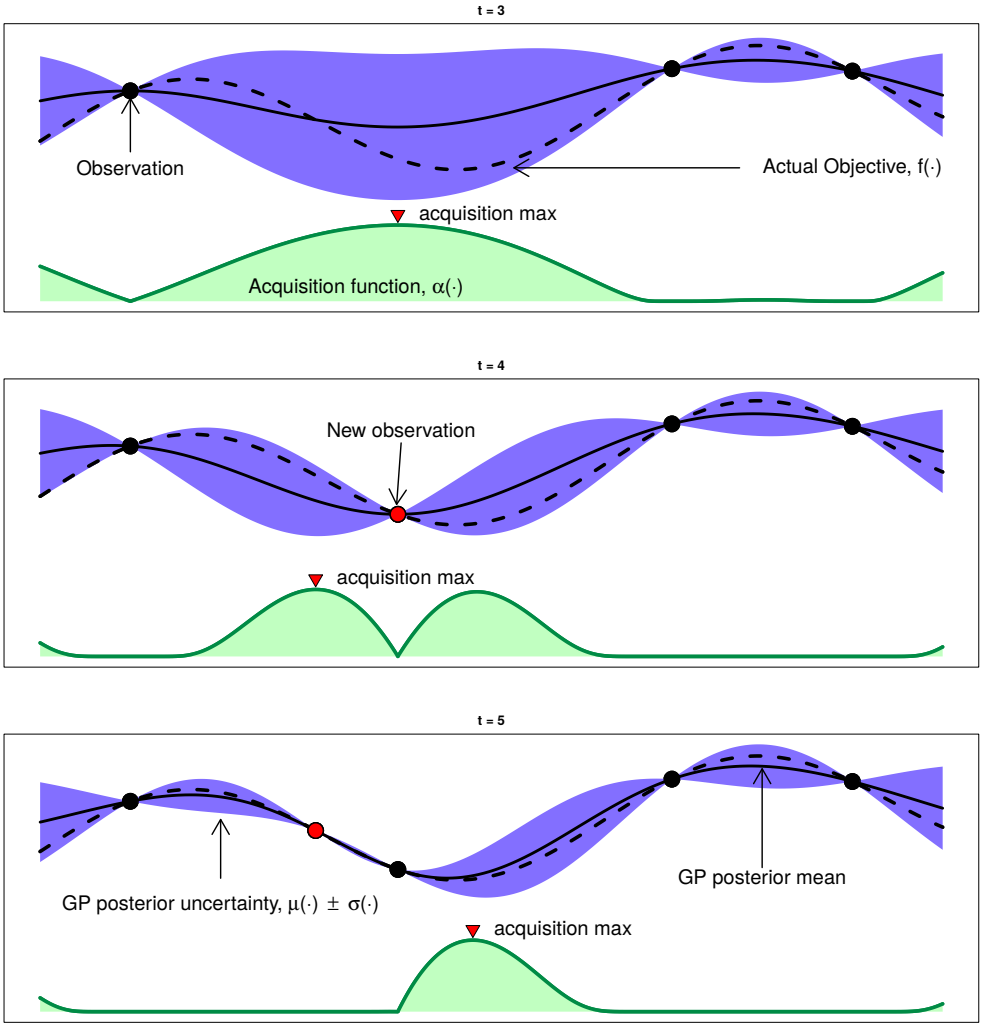

FIGURE 3.1: BO acquisition function $\alpha(\mathbf{x})$ and GP predictive distribution $p(f(\mathbf{x})|\mathcal{D})$ on a toy 1D noiseless problem. The figures show a GP estimation of the objective $f(\mathbf{x})$ over three iterations. The acquisition function $\alpha(\mathbf{x})$ is shown in the lower part of the plot. The acquisition $\alpha(\mathbf{x})$ is high where the GP predicts a low value of the objective $f(\mathbf{x})$ and where the uncertainty about its prediction is high. Those regions in which it is unlikely to find the global minimum $\mathbf{x}^\star$ of $f(\mathbf{x})$ have low acquisition values, and will not be explored.

Input: Maximum number of evaluations T.
for $t = 1, 2, 3, \ldots, T$ **do**
1: if $N = 1$**:**
Choose $\mathbf{x}_t$ at random from $\mathcal{X}$.
else:
Find $\mathbf{x}_t$ by maximizing the acquisition function: $\mathbf{x}_t = \arg\max_{\mathbf{x} \in \mathcal{X}} \alpha_t(\mathbf{x})$.
2: Evaluate the black-box objective $f(\cdot)$ at $\mathbf{x}_t$: $y_t = f(\mathbf{x}_t) + \epsilon_t$.
3: Augment the dataset with the new observation: $\mathcal{D}_{1:t} = \mathcal{D}_{1:t-1} \bigcup \{\mathbf{x}_t, y_t\}$.
4: Fit again the GP model using the augmented dataset $\mathcal{D}_{1:t}$.
end
5: Obtain the recommendation $\mathbf{x}^\star$: Point associated with the value that optimizes the GP prediction or with the best observed value.
Result: Recommended point $\mathbf{x}^\star$

Algorithm 3: BO of a black-box objective function $f(\mathbf{x})$.

BO is an useful methodology when we assume and the black-box function $f(\mathbf{x})$ actually has some properties such as smoothness. The GP covariance function $k(\mathbf{x}, \mathbf{x}')$ encodes the mentioned properties of the black-box function $f(\mathbf{x})$. It also stores the information given by the dataset of previous observations $\mathcal{D}$ in the Gram matrix K, whose entries are $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Therefore, we can search for the minimizer $\mathbf{x}^\star$ using that information in an intelligent way. In order to do so, BO effectively builds an acquisition function $\alpha(\mathbf{x})$ from the GP predictive distribution $p(f(\mathbf{x})|\mathcal{D})$. If we know that the black-box function does not have the mentioned properties, for example that the objective function is not smooth, we can alternatively perform a random search of the objective function $f(\mathbf{x})$. Random search is a pure explorative procedure. Being model-free, it does not take into account any assumption of the objective function $f(\mathbf{x})$. Hence, if the black-box $f(\mathbf{x})$ properties are the ones modelled by the GP, we can hypothesize that random search is going to perform worse than BO. Due to the GP, BO performs a more intelligent search procedure. Let us compare the performance of BO with respect to random search. We can observe in Figure 3.2 how BO tends to explore a particular region. According to the previous information, the assumptions encoded in the covariance function $k(\mathbf{x}, \mathbf{x}')$ and the acquisition function $\alpha(\mathbf{x})$, that region is promising. The search is guided by that information into that region. In contrast, we can observe how random search does not suggest a point based on any criterion rather than pure exploration. Hence, random search does not identify the promising region, evaluating uniformly all the space at random. Therefore, random search will incur in evaluations that will not be useful for the search of the optimizer if the black-box function has smoothness or other properties that can be captured by the GP covariance function $k(\mathbf{x}, \mathbf{x}')$.

Nevertheless, it seems that BO will perform better than random search in all scenarios, that is not necessarily true. Let us suppose that we want to optimize a black-box $f(\mathbf{x})$ that is white noise. This function is not smooth. We do not obtain any advantage by modelling such a function with a GP. Hence, any acquisition function $\alpha(\mathbf{x})$ would probably do as well as random search in this scenario but BO would be more costly than random search due to the overhead of having to optimize the acquistion function and fit the GP. However, white noise is not the common black-box optimization case. Usually, the assumptions made by the GP covariance functions or the ones implied by another surrogate model are accurate about the objective function $f(\mathbf{x})$. When this is

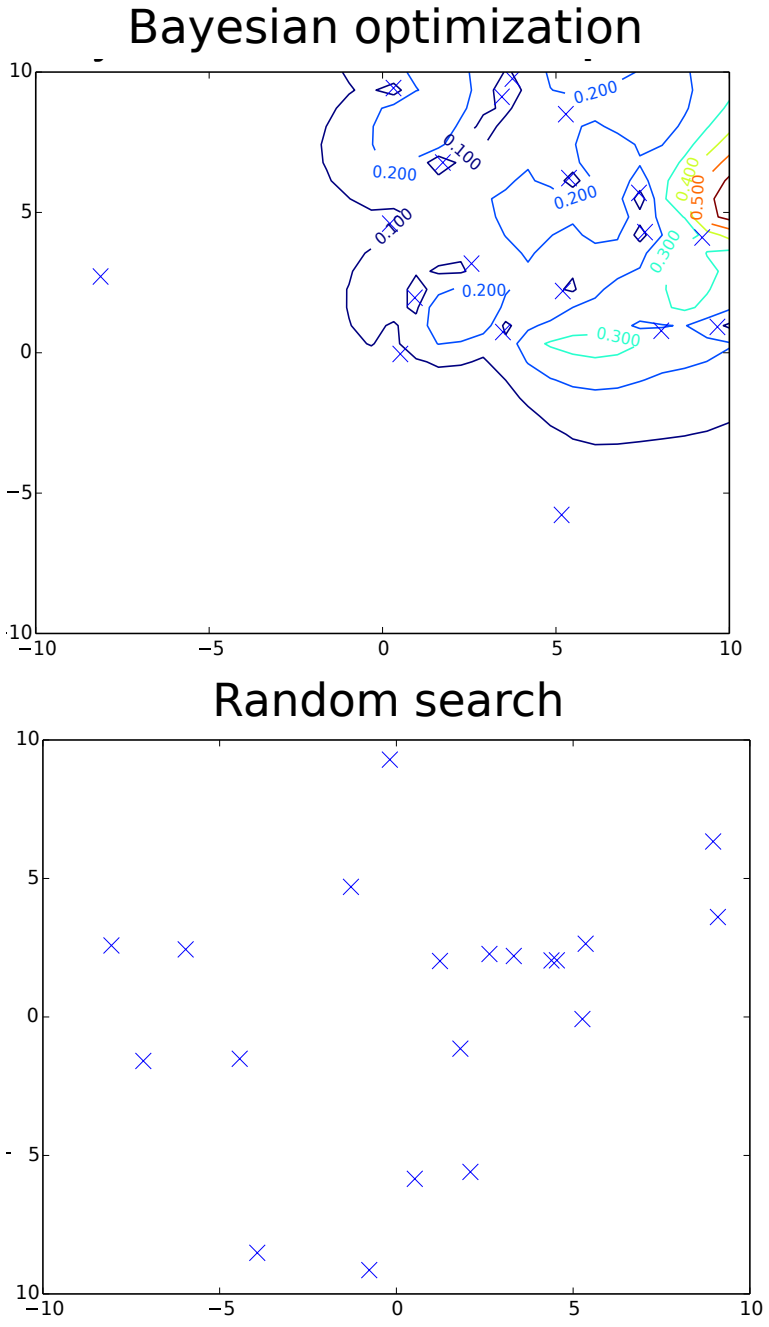

FIGURE 3.2: 2-dimensional toy optimization problem. 20 evaluations performed by a BO procedure (top) and random search (bottom). The contour printed at the top figure represent the acquisition function values, *i.e.* where BO considers that is more useful to evaluate the objective function at the next iteration. Random search uniformly searches the space whilst BO focuses more on promising regions based on the information given by the GP model.

the case, BO obtains, in average, better results than random search, as we can see in a synthetic example in Figure 3.3. in which the actual objective is known. In this figure, the X axis shows the number of evaluations of the objective function $f(\mathbf{x})$. Concretely, 400 evaluations of the objective function $f(\mathbf{x})$ were done in that problem. The Y axis represents the goodness metric, the lower its value the better the result. As in each iteration t both BO and random search suggest a point $\mathbf{x}_t$ to be evaluated, then, we can compute the goodness metric of the evaluation of that point $\mathbf{x}_t$ in every iteration t. The black and yellow line represent the average performance obtained in terms of the goodness metric on 100 experiments, of BO and random search, respectively. The bars of the lines represent the standard deviation of the average performance. At each iteration, represented on the X axis, we can see how the metric is significantly lower in the case of BO. Moreover, if the cost of optimizing the acquisition function and adjusting the GP is negligible, BO provides much better results in less computation time.

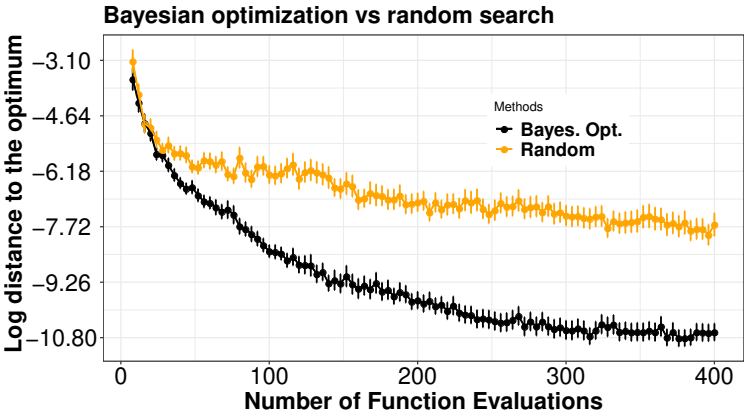

FIGURE 3.3: Obtained performance by BO and random search methods in an optimization problem. The X axis shows the number of evaluations (400) of the objective function $f(\mathbf{x})$. The Y axis represents the goodness metric, the lower the value of this metric, the closer the recommendation to the optimum. We can see how BO outperforms random search in this setting.

3.3 Acquisition Functions

This section describes the details of the, from our point of view, most popular acquisition functions used in BO. We add a special emphasis on those acquisition functions based on information theory concepts. An acquisition function $\alpha(\mathbf{x})$ specifies an exploration and exploitation trade-off. Hence, this section starts covering the trade-off between exploration of unknown areas and exploitation of promising results of objective functions $f(\mathbf{x})$. Then, we describe the most popular acquisition functions used in BO. We conclude the section explaining information theory and acquisition functions based on this theory.

3.3.1 Defining the BO Strategy: Exploration and Exploitation

Acquisition functions $\alpha(\mathbf{x})$ are heuristics that represent an equilibrium between exploration of unknown areas and exploitation of promising solutions. In BO, the shape of the objective function $f(\mathbf{x})$ can only be known through evaluations $y = f(\mathbf{x}) + \epsilon$ of it. Remember that $\epsilon \sim \mathcal{N}(0, \sigma)$ is often additive Gaussian noise, as it has been explained

in previous sections. Let $\mathbf{x}^\star$ be the minimizer of $f(\mathbf{x})$. Good solutions $\mathbf{x}$ are the ones that minimize the regret with respect to the minimum $\mathbf{x}^\star$. The regret is defined as $r = |f(\mathbf{x}^\star) - f(\mathbf{x})|$. The lower the associated regret r of a point $\mathbf{x}$ is, the better this point $\mathbf{x}$ is as a recommendation for the solution of the optimization problem. Points with low regret can be found in unexplored areas and in areas where we have already detected a good point $\mathbf{x}$. We will first comment exploration of unexplored regions of the objective function $f(\mathbf{x})$. These unknown areas may contain points $\mathbf{x}$ with low associated regret r with respect to the extremum $\mathbf{x}^\star$, although we are not certain about this fact, as we have not explored those regions. In particular, they may also contain only bad solutions. Hence, we must also focus on exploiting areas that have provided a point $\mathbf{x}$ with low regret r before. We focus on the latter areas as we assume characteristics of the objective function $f(\mathbf{x})$ such as smoothness. BO assumes that the neighbourhood of a promising evaluation $y_t = f(\mathbf{x}_t) + \epsilon$ can also have promising results. This happens as the function $f(\mathbf{x})$ is assumed to be smooth. Recall that the level of smoothness in every dimension is modelled by the GP lengthscales ℓ_j with $j = 1, ..., D$ and D the dimensionality of the input space. The GP provides analytical expressions for the predictive distribution $p(f(\mathbf{x})|\mathcal{D})$ of the value y_t associated to a new point $\mathbf{x}$. Acquisition functions $\alpha(\mathbf{x})$ enforce exploration by giving high values to points with associated predictions for $f(\mathbf{x})$ with high uncertainty. At the same time, acquisition functions enforce high values for low predictive mean values $f(\mathbf{x})$ assuming minimization. Acquisition functions balance exploration and exploitation of the input space $\mathcal{X}$ by having an analytic expression that balances between the two of them.

Let us describe what happens if the acquisition function explores or exploits excessively. For simplicity, we are assuming that the objective function evaluation is not contaminated by noise, $y = f(\mathbf{x})$. However, it is trivial to extend the reasoning that is going to be described in this paragraph for the noisy scenario. In the case of non-pathological scenarios, if the acquisition function $\alpha(\mathbf{x})$ exploits too much, it is likely that it focuses the evaluations near a local optima $\mathbf{x}$ at the end of the process. Let us define the neighbourhood $n(\mathbf{x})$ of a point $\mathbf{x}$ as the set of all points $n(\mathbf{x}) = \mathcal{S}$ whose distance from the point $\mathbf{x}$ is lower than a certain value ν: $||\mathbf{x} - \mathbf{x}'|| < \nu \; \forall \; \mathbf{x}_i \in \mathcal{S}$. A local optimum $\mathbf{x}$ is a point $\mathbf{x}$ whose evaluation y has lower associated regret than the evaluations of its neighbourhood $\mathbf{y} = f(n(\mathbf{x}))$, that is, $f(\mathbf{x}) < f(\mathbf{x}_i) \; \forall \; \mathbf{x}_i \in \mathcal{S}$ assuming minimization. However, it has worse regret than $f(\mathbf{x}^\star)$, retrieved by the global optimum $\mathbf{x}^\star$. Specifically, the associated regret of evaluating $\mathbf{x}^\star$ is zero. Importantly, the global optimum $\mathbf{x}^\star$ may be found in a different place in space than the neighbourhood $\mathcal{S}$ of the local optimum $\mathbf{x}$. Recall that this set of points $\mathcal{S}$ is a hyper-sphere that belongs to the input space $\mathcal{S} \in \mathcal{X}$. In a minimization context, the local optima $\mathbf{x}$ satisfies both $f(\mathbf{x}) < f(\mathbf{s}_i) \; \forall \; \mathbf{s}_i \in \mathcal{S}$ and $f(\mathbf{x}^\star) < f(\mathbf{x})$. An acquisition function $\alpha(\mathbf{x})$ that exploits too much will drive its suggestions $\mathbf{x}_\alpha = \text{argmax}_{\mathbf{x} \in \mathcal{X}} \alpha(\mathbf{x})$ towards a region with low prediction values assuming minimization. Unexplored regions will then not be visited with high probability. On the other hand, if the acquisition function $\alpha(\mathbf{x})$ explores too much, it will focus on the evaluation of points with high associated uncertainty and will miss to exploit promising regions of the input space $\mathcal{X}$ according to the information given by the GP model. In particular, this would incur in useless evaluations of the space $\mathcal{X}$ if the assumptions taken by the GP covariance function $k(\mathbf{x}, \mathbf{x}')$ about the objective function $f(\mathbf{x})$ are correct. In this way, the promising regions of the input space $\mathcal{X}$ will probably not be evaluated enough to find a point $\mathbf{x}$ with low associated regret.

Nevertheless, we would think, given the provided arguments, that the optimum behaviour for an acquisition function $\alpha(\mathbf{x})$ is a middle term between exploration and

exploitation, it may also not be the best strategy. Let X be the domain of a function f. The function f is convex if $\forall t \in [0,1]$ and $\forall x_1, x_2 \in X$ we have $f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$. Consider, for example, that the objective function $f(\mathbf{x})$ is convex, and we did not know it a priori. In this case, using an acquisition function $\alpha(\mathbf{x})$ with a strong exploitation behaviour may be a good idea. On the other hand, if the objective function $f(\mathbf{x})$ has multiple optima, an acquisition function $\alpha(\mathbf{x})$ that gives more importance to exploration may have more probability to find the global optimum $\mathbf{x}^\star$. Let us consider, as an example, the GP posterior distribution plotted in Figure 3.4. Let us assume that the GP prediction $\mu(\mathbf{x})$ perfectly matches an objective function $f(\mathbf{x})$, although, we do not know it in practice. We can observe that the optimum $\mathbf{x}^\star$, assuming minimization, lies in a region contained in the interval $[0.2, 0.4]$. A exploitative acquisition function $\alpha(\mathbf{x})$ would, in this case, explore this area. An explorative acquisition function $\alpha(\mathbf{x})$ would by contrast focus on the area above 0.8. It will do it because this region is associated with high uncertainty values according to the GP predictive distribution. It is also associated with better prediction values that the area below 0.2. In this particular scenario, exploiting is better, as the area above 0.8 contains a local optimum $\mathbf{x}$, as we had assumed that the GP prediction $\mu(\mathbf{x})$ matches the objective function $f(\mathbf{x})$, $\mathbf{x}^\star \in [0.2, 0.4]$. Nevertheless, if we do not know the objective function $f(\mathbf{x})$ shape, the optimum $\mathbf{x}^\star$ may belong to the area above 0.8. In this case, exploitative acquisition functions $\alpha(\mathbf{x})$ will fail to discover, with a high probability, the area above 0.8, missing the optimum $\mathbf{x}^\star$. We can hence not provide a perfect criterion for exploration and exploitation. It strongly depends on the particular shape of the objective function $f(\mathbf{x})$. There is no single perfect criterion for all function shapes. Since there is an infinite number of shapes

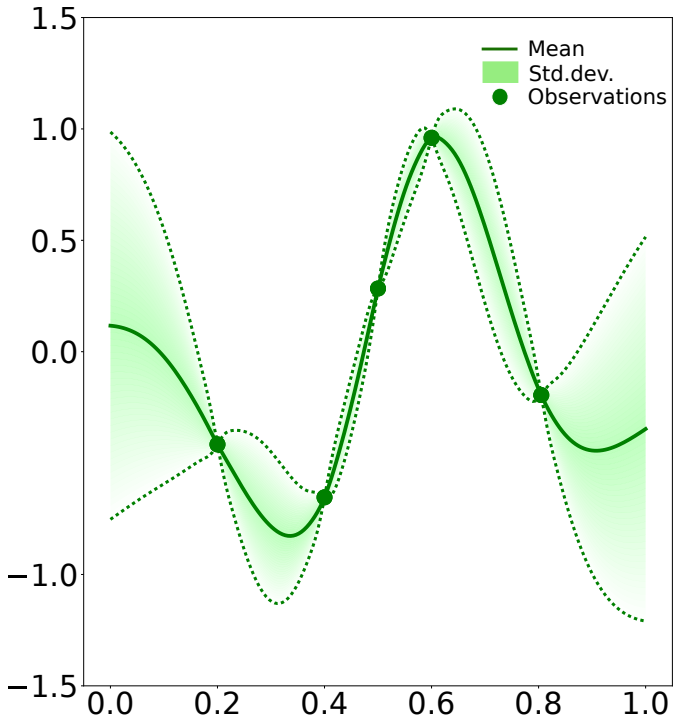

FIGURE 3.4: GP posterior distribution. Observations are plotted as points, the prediction $\mu(\mathbf{x})$ of the objective function $f(\mathbf{x})$, or mean of the GP, is shown as a continuous green line and the standard deviation $\sigma(\mathbf{x})$, or uncertainty over the prediction $\mu(\mathbf{x})$, is shown as shaded green.

for an objective function $f(\mathbf{x})$. Hence, given the provided arguments, there is not a perfect strategy $\alpha(\mathbf{x})$ that is the best for all of them. Nevertheless, designing a criterion $\alpha(\mathbf{x})$ that represents a good trade-off between exploration and exploitation is, generally, the best idea. In practice, objective functions are generally smooth functions. A good trade-off between exploration and exploitation provides good results for such functions.

Another thing to consider in BO is to take into account prior knowledge (Souza et al., 2020). Depending on this knowledge, we may consider using a particular model or acquisition function $\alpha(\mathbf{x})$. Let us consider the case that a black-box $f(\mathbf{x})$ is non-stationary. Concretely, a non-stationary function f is a function whose smoothness changes across the input space $\mathcal{X}$ where the function f is evaluated. Non-stationary functions are not modelled by common GP covariance functions. Hence, this is a problem, as BO is not going to deliver great results if the model assumptions about the objective function $f(\mathbf{x})$ are not satisfied. In this particular case, we may have to resort to a transformation of the space such as the beta-warp transformation (Snoek et al., 2014). In particular, the beta-warp transformation converts a non-stationary objective function $f(\mathbf{x})$ it to a stationary function. But before applying this transformation, the input space $\mathcal{X}$ range is normalized into a $[0, 1]^d$ hyper-cube where d is the number of dimensions of the input space $\mathcal{X}$. Then, the beta-warp transformation performs the warping of the input space $\mathcal{X}$ using a beta cumulative function. More concretely, the beta cumulative function has two parameters: α and β. Specifically, the transformation is carried out by applying a beta cumulative function to every dimension d of the input space $\mathcal{X}$. Hence, we have two vectors of parameters in this transformation $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_d)^T$ and $\boldsymbol{\beta} = (\beta_1, ..., \beta_d)^T$. In order to be successful, the values of the parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ need to be adjusted by the BO method. It is useful to first see how the transformation is applied to understand how the process learns these values. When the entries of the Gram matrix need to be computed for the posterior distribution of the GP, the method changes the covariance function to be $k(w(\mathbf{x}), w(\hat{\mathbf{x}}))$, such that for every dimension d a function $w_d(\mathbf{x}_d)$ is applied:

$$w_d(\mathbf{x}_d) = \text{betaCDF}(\mathbf{x}_d, \alpha_d, \beta_d) = \int_0^{\mathbf{x}_d} \frac{u^{\alpha_d - 1}(1-u)^{\beta_d - 1}}{B(\alpha_d, \beta_d)} du\,, \tag{3.4}$$

where d is an index of a dimension of the input space, u are the values of the input space, betaCDF is an acronym for the beta cumulative function and $B(\alpha_d, \beta_d)$ is the beta function (Abramowitz et al., 1988). The values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ that convert the function into an stationary one will maximize the marginal likelihood of the GP since the GP assumes a stationary function. Hence, if the function is stationary the GP will perform a more accurate fit than if the function is not stationary. Unfortunately, providing a point estimate of these values by maximizing the marginal likelihood is likely to incur in overfitting as we are going to jointly maximize the hyper-parameters of the GP covariance function along with the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ hyper-parameters. To avoid the described issue, these vectors, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, can as well be treated as hyper-parameters of the GP covariance function. Therefore, we can use Markov Chain Monte Carlo via slice sampling to obtain samples from them, as described in Chapter 2. Finally, a log-normal distribution is used for every hyper-parameter α_d and β_d as a prior distribution. We can observe in Figure 3.5 how the beta transformation can change functions shape. More concretely, in the figures of the left we can observe non-stationary functions, an exponential decay function and a periodic non-stationary function. Additionally, in the figures of the center we can see the beta cumulative functions adjusted to convert the non-stationary functions of the figures of the left to stationary functions. Finally, we observe how, by applying the

beta transformation, the functions plotted on the figures of the left are converted on stationary functions, plotted on the figures of the right.

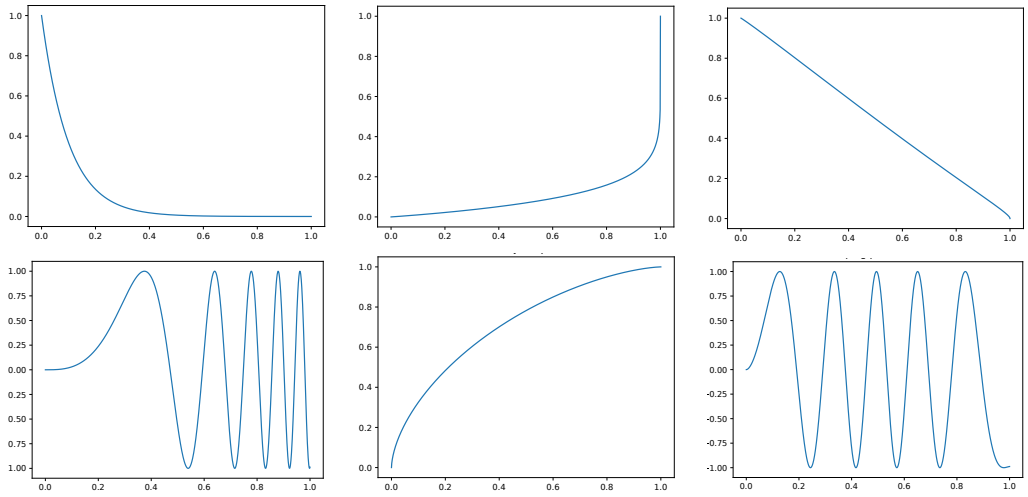

FIGURE 3.5: Beta transformation effects on non-stationary functions. (Top-left) Exponential decay function. (Top-center) Adjusted beta cumulative function to make the exponential decay stationary. (Top-right) Beta transformation on the exponential decay function. (Down-left) Non-stationary periodic function. (Down-center) Adjusted beta cumulative function to make the periodic function stationary. (Down-right) Beta transformation on the periodic function.

As we have said before, if we know that the objective function $f(\mathbf{x})$ is monotonic or convex in the majority of space $\mathcal{X}$, we may consider an exploitative acquisition function $\alpha(\mathbf{x})$. If instead, the black-box $f(\mathbf{x})$ is multi-modal, we may choose an exploratory acquisition function $\alpha(\mathbf{x})$. On the other hand, if we do not have prior knowledge about the objective function $f(\mathbf{x})$, there is no single optimum strategy. This issue is defined as the No Free Lunch theorem of optimization (Ho and Pepyne, 2002). It basically tells us that a general-purpose, universal optimization strategy $\alpha(\mathbf{x})$ will not be optimal for every possible function shape. Concretely, the only way one strategy can outperform another is if it has characteristics that are more suitable for solving a specific problem. Nevertheless, we can compare the behaviour of the most popular acquisition functions. In order to do so, the performance of a set of acquisition functions can be compared in a benchmark of popular objective functions $f(\mathbf{x})$. In particular, it can be observed that certain strategies $\alpha(\mathbf{x})$ tend to outperform others in the majority of cases (Jamil and Yang, 2013). It is because of this reason that research on acquisition functions $\alpha(\mathbf{x})$ is important. Although theoretically we cannot propose an universal best strategy $\alpha(\mathbf{x})$, in practice certain strategies outperform others in several problems of interest used as benchmarks to compare acquisition functions.

3.3.2 Acquisition Function criteria

We now describe in detail the, from our point of view, most popular acquisition functions $\alpha(\mathbf{x})$. We will first study acquisition functions $\alpha(\mathbf{x})$ that are not based on information theory concepts. Acquisition functions based on information theory will be illustrated after studying the fundamentals of information theory.

- **Probability of Improvement (PI)**: Let $\kappa = \max_i \mu_i(\mathbf{x}_i)$ be the best observed predicted value, where the i index represents the iteration $i = 1, ..., t$ with t the

current iteration. κ is also known as the incumbent. This acquisition function measures the probability of improvement with respect to κ of every point $\mathbf{x} \in \mathcal{X}$ according to the predictive distribution $p(f(\mathbf{x})|\mathcal{D})$ (Kushner, 1964). The intuitive idea of this acquisition function $\alpha(\mathbf{x})$ is to select the point $\mathbf{x}$ whose expected associated value y maximizes the probability of improvement over the current best value κ. As it is easy to observe, this acquisition function $\alpha(\mathbf{x})$ is merely exploitative, ignoring an exploration behaviour. We may only consider this kind of acquisition function $\alpha(\mathbf{x})$ when we suspect, according to prior knowledge, that the objective function $f(\mathbf{x})$ is convex or monotonic. To remedy this behaviour, we can add to the PI acquisition function $\alpha(\mathbf{x})$ analytical expression an $\epsilon \geq 0$ trade-off parameter. If this parameter has a high value, exploration is priorized, if it has a value near zero, exploitation is encouraged. An popular heuristic is to decrease this value over iterations, although empirical evidence suggest that it has not shown good practical results (Lizotte, 2008). PI is defined as:

$$\mathrm{PI}(\mathbf{x}) = p(f(\mathbf{x}) \geq \kappa + \epsilon) = \Phi\left(\frac{\mu(\mathbf{x}) - \kappa - \epsilon}{\sigma(\mathbf{x})}\right). \tag{3.5}$$

Where Φ is the standard Gaussian cumulative distribution function (CDF). We can see an example of the PI acquisition function shape in Figure 3.6. In particular, we can observe that PI tends to evaluate an infinitesimally close point to an already evaluated point as $\epsilon = 0$. Although the improvement is infinitesimally small, the probability of improvement is very high. Nevertheless, as we can see, the exploitative behaviour does not let PI explore a different area from the one already observed. This behaviour may cause that PI will not discover the global optimum of the objective function $f(\mathbf{x})$.

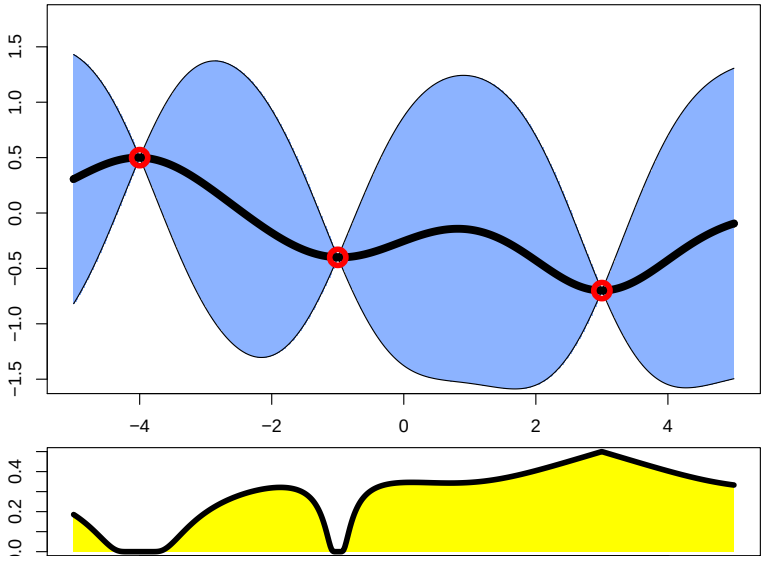

FIGURE 3.6: (Top) GP model fitting un unknown function $f(\mathbf{x})$. It is assumed that there is no additive noise. Observations are displayed in red. The prediction of the unknown function $\mu(\mathbf{x})$ is plotted as a continuous black line. The uncertainty over the unknown function is shown in blue. (Down) PI acquisition function $\alpha(\mathbf{x})$. The point $\mathbf{x}^\star$ associated with the maximum PI value $\mathbf{x}^\star = \mathrm{argmax}_{\mathbf{x}\in\mathcal{X}}\alpha(\mathbf{x})$ is the suggested for evaluation.

- **Gaussian Process Upper Confidence Bound (GP-UCB)**: This acquisition function is a simple yet effective criterion that easily trades-off exploration and exploitation (Srinivas et al., 2009). Its details have already been described in Chapter 1 as a quick example of an acquisition function $\alpha(\mathbf{x})$. Its main advantage with respect to other acquisition functions is its interpretability. The analytical expression is just a summation of the mean and the standard deviation given by the GP predictive distribution of $f(\mathbf{x})$: $\alpha(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}) \; \forall \, \mathbf{x} \in \mathcal{X}$, where $\mathcal{X}$ is the input space, κ is a parameter to switch exploration and exploitation, $\mu(\mathbf{x})$ is the GP predictive mean and $\sigma(\mathbf{x})$ is the standard deviation. Figure 1.5 shows the GP-UCB acquisition function in Chapter 1. GP-UCB is easy to tune, as one only has to modify the κ parameter to trade-off exploration and exploitation. More concretely, higher values of κ make GP-UCB exploit more and lower values of κ make GP-UCB explore more. Moreover, GP-UCB also has lots of theoretical properties involving the regret r of the suggestions that other acquisition functions lack of. Hence, from a theoretical point of view, it is a robust acquisition function. However, in practice, it tends to deliver worse results than other acquisition functions $\alpha(\mathbf{x})$ (Wang and Jegelka, 2017).

  The GP-UCB acquisition function shape can be seen in Figure 3.7. We can see, in the top plot, the GP predictive distribution $p(y|\mathbf{x}, \mathcal{D})$ of an unknown function $f(\mathbf{x})$. The predictive mean $\mu(\mathbf{x})$ is shown as a black line and the standard deviation $\sigma(\mathbf{x})$ is shown in blue. In the lower plot, it is displayed the GP-UCB acquisition function $\alpha(\mathbf{x})$. We see how it takes higher values where the prediction given by the GP model $\mu(\mathbf{x})$ is low and the uncertainty about the prediction $\sigma(\mathbf{x})$ is high. For example, in areas where the GP has already been conditioned on previous observations, GP-UCB values are lower than in unexplored zones.

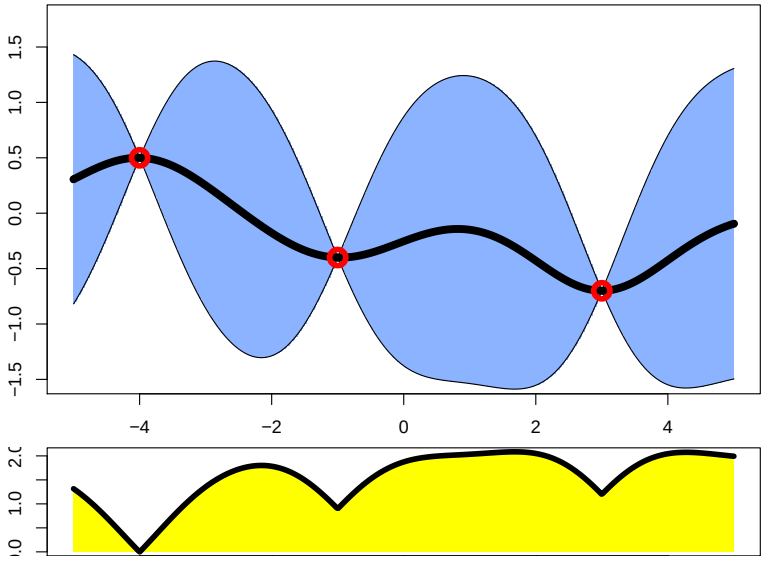

FIGURE 3.7: (Top) GP model fitting un unknown function $f(\mathbf{x})$. It is assumed that there is no additive noise. Observations are displayed in red. The prediction of the unknown function $\mu(\mathbf{x})$ is plotted as a continuous black line. The uncertainty over the unknown function $\sigma(\mathbf{x})$ is shown in blue. (Bottom) GP-UCB acquisition function $\alpha(\mathbf{x})$. The point $\mathbf{x}^\star$ associated with the maximum GP-UCB acquisition function value $\mathbf{x}^\star = \text{argmax}_{\mathbf{x}\in\mathcal{X}}\alpha(\mathbf{x})$ is suggested for evaluation.

- **Expected Improvement (EI)**: This acquisition function $\alpha(\mathbf{x})$ generalizes PI by studying the magnitude of the improvement over the incumbent κ that the evaluation y associated with a point $\mathbf{x}$ is expected to produce. In particular, we have seen that PI sometimes improves the solution infinitesimally although the probability of improvement is very high. Fortunately, by also taking into account the magintude of improvement we circumvent that issue. Moreover, due to its simplicity of implementation and good empirical results, EI is arguably the most popular BO acquisition function $\alpha(\mathbf{x})$. It does not explore as much as information theoretical based acquisition functions $\alpha(\mathbf{x})$ but it works very well for objective functions $f(\mathbf{x})$ that are not very complex. Let $\chi(\mathbf{x}) = \frac{\mu(\mathbf{x})-\kappa-\epsilon}{\sigma(\mathbf{x})}$, EI is given by:

$$\mathrm{EI}(\mathbf{x}) = (\mu(\mathbf{x}) - \kappa - \epsilon)\Phi(\chi(\mathbf{x})) + \sigma(\chi(\mathbf{x}))\phi(\chi(\mathbf{x}))\,, \tag{3.6}$$

  where $\Phi(\cdot)$ is the Gaussian CDF and $\phi(\cdot)$ is the Gaussian PDF. In the case of $\sigma(\mathbf{x}) = 0$, then $\mathrm{EI}(\mathbf{x}) = 0$, to avoid an indetermination. We can see the shape of EI in Figure 3.8. It is a heavily based exploitative criterion.

  As we have seen, acquisition functions that are not based on information theory evaluate a point $\mathbf{x} \in \mathcal{X}$ by maximizing a heuristic $\alpha(\mathbf{x})$ that converts the marginal belief over the function evaluation y at that point $\mathbf{x}$, which is an univariate Gaussian distribution, into an utility for evaluation designed to a have high value at locations close to the minimum of the objective function (Hennig and Schuler, 2012). Hence, although being computationally lightweight and easy to implement, the described acquisition functions are local measures that cannot capture the retrieved gain of knowledge of an evaluation of an unknown broad region of space $\mathcal{X}$ with low uncertainty. These acquisitions tend to evaluate a small region of high uncertainty that may be less interesting to explore in terms of the knowledge that is gained about the objective function $f(\mathbf{x})$ (Hennig and Schuler, 2012). As we will describe in Section 3.3.4, information theory acquisition functions circumvent this issue and are global measures that tend to explore more than acquisition functions that are not based on information theory. However, information theory acquisition functions are not as popular as the expected improvement criterion, probably because they are more difficult to implement.

- **GP-Hedge**: Previous acquisition functions $\alpha(\mathbf{x})$ deliver good empirical results in a plethora of scenarios (Snoek et al., 2012). Nevertheless, we observe that there is not a best acquisition function $\alpha(\mathbf{x})$ for every single objective function $f(\mathbf{x})$, as we have introduced earlier by the concept of No Free Lunch for optimization problems (Ho and Pepyne, 2002). If there is not a single ideal acquisition function $\alpha(\mathbf{x})$, why cannot we combine them in an intelligent way?. By performing this task, intuitively, we could think that we gain the best from all the considered acquisition functions. This is the motivation behind GP-Hedge. Let Γ be a set of acquisition functions, also called acquisition function portfolio. Let N be its size and w_i an associated weight i to an acquistion function $\alpha_i(\mathbf{x}) \in \Gamma$ such that $\sum_{i=1}^{N} w_i = 1$. We can then hypothesize that if we average $\sum_{i=1}^{N} \alpha_i(\mathbf{x})/N$ the shapes of different acquisition functions $\alpha_i(\mathbf{x}) \in \Gamma$, we can incorporate the particular characteristics of every single acquisition function Γ into a single aggregated acquisition function. This is the idea of the Portfolio Allocation for Bayesian Optimization (Hoffman et al., 2011). This approach, instead of employing a single acquisition function $\alpha(\mathbf{x})$, takes into account a portfolio of acquisition functions Γ. The obvious disadvantage of this

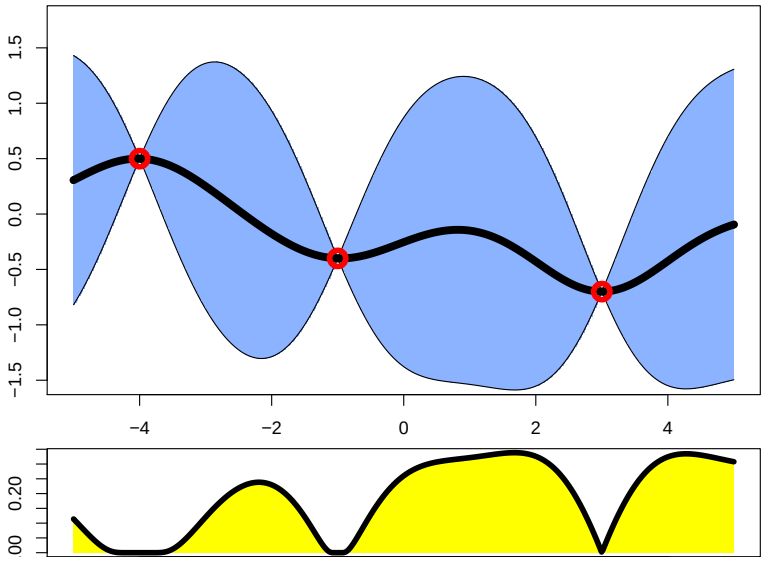

FIGURE 3.8: (Top) GP model fitting un unknown function $f(\mathbf{x})$. It is assumed that there is no additive noise. Observations are displayed in red. The prediction of the unknown function $\mu(\mathbf{x})$ is plotted as a continuous black line. The uncertainty over the unknown function $\sigma(\mathbf{x})$ is shown in blue. (Bottom) EI acquisition function $\alpha(\mathbf{x})$. The point $\mathbf{x}^\star$ associated with the maximum EI value $\mathbf{x}^\star = \text{argmax}_{\mathbf{x}\in\mathcal{X}}\alpha(\mathbf{x})$ is suggested for evaluation.

technique is the extra computational time spent on computing the combination of acquisition functions Γ. Another problem is the difficulty of implemntation, as it requires the implementation of a portfolio of acquisition functions Γ. This means that the difficulty of implementation and the computation time of GP-Hedge will be, at least, the sum of the difficulties of implementation and the computational time of the considered acquisition functions Γ. In the implementation GP-Hedge, GP-UCB, EI and PI were considered (Hoffman et al., 2011). Although, GP-Hedge could be generalized to any number of different acquisition functions $\alpha(\mathbf{x})$.

We now describe how GP-Hedge suggests a point $\mathbf{x}$ for evaluation. Let us consider N different acquisition functions, *i.e.*, the size of Γ. First, GP-Hedge suggests the N points $\mathbf{X}$, where $\mathbf{X}$ is a matrix whose rows $\mathbf{x}_j$ are the N points that maximize each acquisition function $\alpha_j(\mathbf{x})$ separately $\mathbf{x}_j = \underset{\mathbf{x}\in\mathcal{X}}{\text{argmax}}\ \alpha_j(\mathbf{x})$, where j is the index of the acquisition function $\alpha_j(\mathbf{x}) \in \Gamma$. GP-Hedge places a generalized Bernoulli distribution over the set of acquisition functions Γ. That is, GP-Hedge associates a probability $p_t(j)$ to each point $\mathbf{x}_j$ such that $\sum_{j=1}^{N} p_t(j) = 1$ where t is an iteration index. Let the cumulative reward c_j of an acquisition function $\alpha_j(\mathbf{x})$ be the sum of inmediate rewards r_j^t of the evaluations y_j^t retrieved from an acquisition function $\alpha_j(\mathbf{x})$. That is, $c_j = \sum_{i=1}^{t} r_j^t$. The probability $p_t(j)$ that GP-Hedge assigns to an acquisition function $\alpha_j(\mathbf{x})$ is based on the cumulative rewards c_j of $\alpha_j(\mathbf{x})$. GP-Hedge selects the suggestion $\mathbf{x}_j$ of $\mathbf{X}$ by sampling the categorical distribution. Those acquisition functions $\alpha_j(\mathbf{x})$ with a higher associated probability value $p_t(j)$ would have more probability of being selected. After selecting an acquisition function $\alpha_j(\mathbf{x})$, GP-Hedge suggests the point $\mathbf{x}_j = \underset{\mathbf{x}\in\mathcal{X}}{\text{argmax}}\ \alpha_j(\mathbf{x})$ for evaluation. Let $\mathbf{g}$ be a vector whose elements g_j represent the probability $p_t(j)$ of choosing an acquisition function $\alpha_j(\mathbf{x})$. We define $\mathbf{g}$ as the gain vector. Once an evaluation $y = f(\mathbf{x}_j)$

is retrieved, GP-Hedge computes a reward r_j for each acquisition function $\alpha_j(\mathbf{x})$ and updates the gain vector $\mathbf{g}$ that contains the probability $p_t(j)$ of choosing an acquisition function $\alpha_j(\mathbf{x})$. More details about how these computations are performed are given in the GP-Hedge article (Hoffman et al., 2011). By employing such a procedure, the idea is that the optimum acquisition function $\alpha(\mathbf{x})$ for a given objective function $f(\mathbf{x})$ is learned as more evaluations y of $f(\mathbf{x})$ are performed. The drawback of GP-Hedge is that all the acquisition functions $\alpha(\mathbf{x})$ must be optimized at every iteration t. Not only that, but if the gain vector $\mathbf{g}$ is not updated in a smartly way, we can learn an acquisition function $\alpha(\mathbf{x})$ that is effective in the early stage of the optimization but not at the end. For example, consider that an exploratory acquisition function is sampled in the last stage of the optimization as it has high probability value $p_t(j)$ in the gain vector. This is undesirable, as in the last stage of the optimization, an exploitation behaviour is preferred as we assume that we have already identified promising areas. We do not need to explore more but to exploit those areas. Hence, if the gain vector $\mathbf{g}$ is not updated correctly, we will not have the desired behaviour in the optimization process.

We can observe in Figure 3.9 how combining EI, PI and GP-UCB acquisition functions can be considered a good idea as the aggregated acquisition function combines the properties of the mentioned acquisition functions. This function is simply the sum of these criteria divided by the number of the criteria $\sum_{i=1}^{3} \alpha_i(\mathbf{x})/3$.

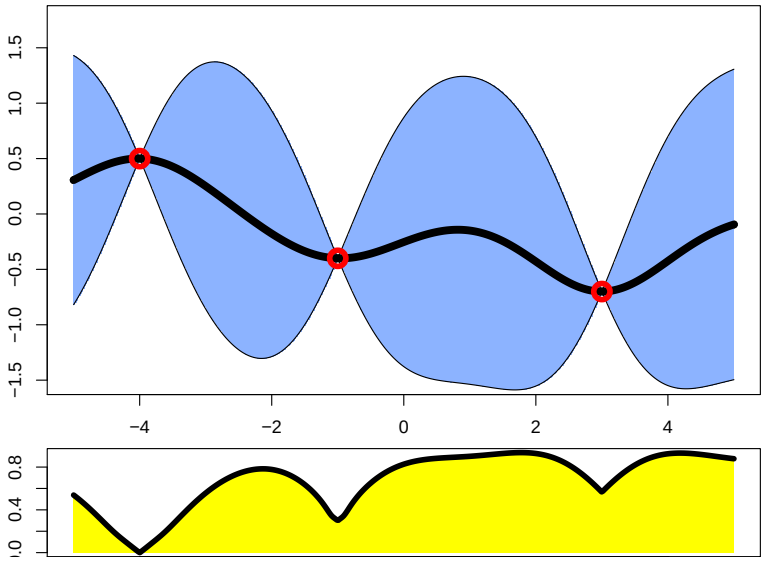

FIGURE 3.9: (Top) GP model fitting un unknown function $f(\mathbf{x})$. It is assumed that there is no additive noise. Observations are displayed in red. The prediction of the unknown function $\mu(\mathbf{x})$ is plotted as a continuous black line. The uncertainty over the unknown function $\sigma(\mathbf{x})$ is shown in blue. (Bottom) Combined acquisition function $\alpha(\mathbf{x})$ of EI, PI and GP-UCB. The point $\mathbf{x}^\star$ associated with the maximum GP-Hedge value $\mathbf{x}^\star = \text{argmax}_{\mathbf{x}\in\mathcal{X}}\alpha(\mathbf{x})$ is suggested for evaluation.

There are another family of acquisition functions $\alpha(\mathbf{x})$ that are based on information theory concepts. To better understand them, it is useful to first study the fundamentals of information theory.

3.3.3 Information Theory

Information theory studies the quantification, storage, and communication of information (Cover and Thomas, 2012; Ghahramani, 2006; MacKay, 2003). In particular, is a computer science subfield that, amongst other things, provides an answer to two fundamental questions regarding information. First, one deals with the ultimate data compression. Here, information theory provides the entropy $H(\cdot)$ measure. Second, it answers what is the ultimate transmission rate of communication, proposing the channel capacity $C(\cdot)$. Information theory is a broad field but, in this thesis, we do only need to know a pair of concepts of the field. Those concepts are necessary to understand the methods described in the following chapters. For further information regarding this field, we encourage to read the work of MacKay (2003).

Both the entropy $H(\cdot)$ and the channel capacity $C(\cdot)$ are functions of underlying probability distributions $\mathcal{P}$. The entropy $H(\cdot)$ can be viewed as a measure of information for a probability distribution $\mathcal{P}$ associated with a random variable X. That is, its self-information. It can be used as a measure of uncertainty of a random variable X. When the random variable is continuous, we refer to the entropy as differential entropy (MacKay, 2003). The entropy of an uni-dimensional continuous random variable X with a probability density function $p(x)$, or differential entropy $H[p(X)]$, is given by the following expression:

$$H[p(X)] = -\int_S p(x) \log p(x) dx \,. \tag{3.7}$$

Where S is the support of the random variable X, that is, the space where $p(x)$ is defined. The entropy $H(\cdot)$ is useful to model the following relation: If we have a random variable X with high entropy $H(\cdot)$, that means that we have low information about the values that it may take. On the other hand, if we consider a random variable X with low entropy $H(\cdot)$, it is a sign that we have high information about the potential values that the variable X can take. In other words, the uncertainty about that random variable X is low. We graphically illustrate this idea in Figure 3.10. Another interesting concept regarding

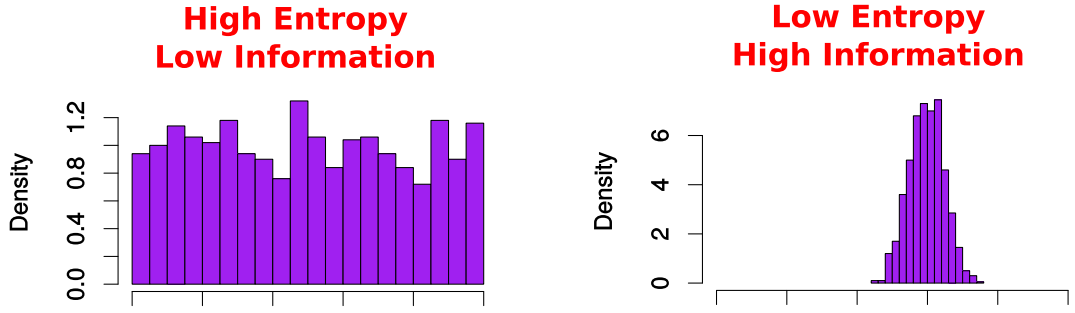

FIGURE 3.10: (Left) Probability distribution of a random variable with high entropy. (Right) Probability distribution of a random variable with low entropy.

information theory, that is used in this thesis, is the mutual information $I(X;Y)$ of two random variables X and Y. Mutual information is defined as the amount of information that a random variable X contains about another random variable Y. It is the reduction in the uncertainty of one random variable X due to the knowledge of the other. Mutual information is a symmetric function. Consider two random variables X and Y with a joint probability density function $p(x, y)$ and marginal probability density functions $p(x)$ and $p(y)$. The mutual information $I(X;Y)$ is the relative entropy between the joint

distribution $p(x, y)$ and the marginal distributions $p(x)$ and $p(y)$:

$$I(X;Y) = \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \,. \tag{3.8}$$

Interestingly, the mutual information $I(X;Y)$ is also the KL divergence between the joint distribution $p(x, y)$ and the product of marginal distributions $p(x)$ and $p(y)$. In the next subsection, we will illustrate how differential entropy $H[p(X)]$ is used to search for the extremum $\mathbf{x}^\star$ of an optimization problem. We will also describe how the concept of mutual information $I(X;Y)$ is used to propose an acquisition function $\alpha(\mathbf{x})$.

3.3.4 Information Theory Based Acquisition Functions

We can use the previously seen concepts related to information theory to build acquisition functions $\alpha(\mathbf{x})$. In particular, we can model the extremum of an optimization problem $\mathbf{x}_\star$ as a random variable X. Hence, it will have an associated probability distribution $\mathcal{P}$ and a probability density function $p(\mathbf{x})$. As we have seen, we can compute the entropy $H[p(\mathbf{x})]$ of a random variable X with associated probability density function $p(\mathbf{x})$. This entropy $H[p(\mathbf{x})]$ is representing the amount of knowledge that we have about the random variable X that models the extremum of the optimization problem $\mathbf{x}_\star$. If the entropy is high $H(\cdot) >>$, that means that we have a high uncertainty about the location of the extremum $\mathbf{x}^\star$. On the other hand, if the entropy is low $H(\cdot) <<$, that means that we have low uncertainty about the location of the extremum $\mathbf{x}^\star$. This is the desired situation in an optimization problem, where we want to know the location of $\mathbf{x}^\star$ in $\mathcal{X}$. Hence, the intuition tells us that if we want to discover the location of the extremum $\mathbf{x}^\star$, we need to minimize the entropy of the location of the extremum $H[p(\mathbf{x}^\star)]$ at every BO iteration. This is the idea behind information theory acquistion functions that we are going to describe in this section. They basically measure the amount of information that we have about the minimizer $\mathbf{x}^\star$ of the problem and try to increase it the most at each iteration.

As we have explained in Section 3.3.2, acquisition functions $\alpha(\mathbf{x})$ such as EI are local measures based on heuristics over the marginal belief of the evaluation y associated with a point $\mathbf{x} \in \mathcal{X}$. Hence, they do not take into account the gain of information that is retrieved when a point $\mathbf{x}$ is evaluated. On the other hand, acquisition functions based on information theory are global measures (Hennig and Schuler, 2012) that do take into account that gain. As a result, they tend to be more exploratory. In other words, acquisition functions that are based on information theory do not just give a measure of the improvement over an incumbent κ, as EI does performing, hence, more exploitation than exploration. More formally, let the image space $\mathcal{Y}$ be the one generated by the evaluations $f(\mathbf{x})$ or expected values $\mu(\mathbf{x})$ of the points $\mathbf{x}$ belonging to the input space $\mathcal{X}$, that is, $\mathcal{Y} = \mu(\mathcal{X})$. As EI is only a function of a particular point $\mathbf{x}$ and not of all the GP predictive variance over the image space $\mathcal{Y}$, we can argue that the improvement over the incumbent κ is a local measure and not a global measure (Frazier, 2018). If the measure was taken by considering all the GP predictive distribution, it would be a global measure of the uncertainty given by the GP predictive distribution. Precisely, information based acquisition functions are more exploratory than the previous explained acquisition functions. This is useful for highly dimensional, complex or multi-modal objective functions $f(\mathbf{x})$ where exploitative acquisition functions can guide the search to local optima. We can hypothesize that, for these objective functions $f(\mathbf{x})$, information based acquisition functions may provide substantial benefits relative to other acquisition

functions such as EI. The most popular acquisition functions that use information theory are the following ones:

- **Entropy Search (ES):** This approach uses the differential entropy of probability distributions to choose the next point to evaluate. It was first proposed by Villemonteix et al. (2009) and then by Hennig and Schuler (2012). First, we must consider the uncertainty over the objective function, given by the posterior distribution of the GP, $p(f(\mathbf{x})|\mathcal{D})$. Hence, we also have uncertainty over the optimum $\mathbf{x}^\star$ of the problem. ES models the optimum $\mathbf{x}^\star$ as a random variable with p.d.f. $p(\mathbf{x}^\star)$. By doing so, the optimum $\mathbf{x}^\star$ has an associated probability density function $p(\mathbf{x}^\star)$ and, hence, an associated differential entropy $H[p(\mathbf{x}^\star)]$. Let $\mathbf{x}_\star$ be the point with associated optimum value $y^\star$ and $p(\mathbf{x}^\star|\mathcal{D})$ be the posterior distribution of the location of the optimum $\mathbf{x}^\star$ given by a GP conditioned on a dataset of observations $\mathcal{D}$. We can model the entropy of the location of the optimum $\mathbf{x}^\star$ at the current iteration by $H[p(\mathbf{x}^\star|\mathcal{D})]$. This quantity is representing the amount of knowledge that we have about the location of $\mathbf{x}^\star$. Bigger entropy will mean less knowledge than lower entropy. We are hence interested in minimizing $H[p(\mathbf{x}^\star|\mathcal{D})]$ at each BO iteration. Specifically, we aim at augmenting the dataset $\mathcal{D}$ with the point $\mathbf{x}$ that is expected to minimize $H[p(\mathbf{x}^\star|\mathcal{D})]$. Critically, we cannot evaluate every single point $\mathbf{x}$, as the objective function is very expensive. Nevertheless, we can use the GP predictive distribution $p(y|\mathbf{x}, \mathcal{D})$ to predict the potential value y of the resulting evaluation of $\mathbf{x}$. Let $H[p(\mathbf{x}^\star|\mathcal{D} \cup (\mathbf{x}, y))$ be the entropy of the conditioned posterior distribution of the optimum $\mathbf{x}^\star$ given the new observation $(\mathbf{x}, y)$. As said before, we do not know the particular value of y but we can use the predicted value y of the GP predictive distribution. Hence, we can compute the expected entropy of the conditional distribution for $\mathbf{x}^\star$ given the new observation $(\mathbf{x}, y)$, that is, $\mathbb{E}_{\mathbf{y}}\{H[p(\mathbf{x}^\star|\mathcal{D} \cup (\mathbf{x}, y))]\}$ where the expectation is with respect to $p(y|\mathbf{x}, \mathcal{D})$. If this quantity is low, it means that we are expected to learn a lot about the location of the optimum $\mathbf{x}^\star$ by performing an evaluation at the candidate point $\mathbf{x}$. On the other hand, if it is high, it means that we can expect that evaluating the objective at $\mathbf{x}$ does not give us significant information about the optimum $\mathbf{x}^\star$. Hence, we are interested in the point that minimizes the expected differential entropy of the conditioned posterior distribution $\mathbb{E}_{\mathbf{y}}\{H[p(\mathbf{x}^\star|\mathcal{D} \cup (\mathbf{x}, y))]\}$ of the optimum $\mathbf{x}^\star$. That is, the point $\mathbf{x}$ that maximizes the expected reduction in the differential entropy $H[\cdot]$ of the posterior for $\mathbf{x}^\star$. The analytical expression of ES represents this idea. We can write the ES acquisition function $\alpha(\mathbf{x})$ as:

$$\text{ES}(\mathbf{x}) = H[p(\mathbf{x}^\star|\mathcal{D})] - \mathbb{E}_{\mathbf{y}}\{H[p(\mathbf{x}^\star|\mathcal{D} \cup \{(\mathbf{x}, y)\})]\}\,, \tag{3.9}$$

where the expectation is taken with respect to the predictive distribution $p(y|\mathcal{D}, \mathbf{x})$ of the black-box function $f(\mathbf{x})$. The problem with ES is that the exact computation of the above expression is infeasible in practice. Concretely, computing the probability distribution of the minimizer given the data $p(\mathbf{x}^\star|\mathcal{D})$ is intractable. To circumvent this issue, ES has to resort to complex approximations. In particular, this problem has been addressed differently first by Villemonteix et al. (2009) and then by Hennig and Schuler (2012). Specifically, in the paper of Villemonteix et al. (2009), $p(\mathbf{x}^\star|\mathcal{D})$ is approximated via samples of the GP conditioned on previous observations. Then, these samples are optimized by an algorithm similar to the Efficient Global Optimization (EGO) algorithm (Jones et al., 1998). Hence, with the set of optimized samples, we can compute the approximation of $p(\mathbf{x}^\star|\mathcal{D})$. However, in order for the

approximation to be accurate, we need to compute and optimize a high number of GP samples, because of the uncertainty that the GP model has of the areas that have not been evaluated yet. On the other hand, in the paper of Hennig and Schuler (2012) the distribution $p(\mathbf{x}_*|\mathcal{D} \cup \{(\mathbf{x}, y)\})$ is discretized. In particular, a high resolution grid of $\mathbf{x}$ and y values is used during the optimization of the previous expression. However, computing all the grid values is computationally very expensive. Moreover, since both entropies $H[p(\mathbf{x}_*|\mathcal{D})]$ and $H[p(\mathbf{x}_*|\mathcal{D} \cup \{(\mathbf{x}, y)\})]$ do not have analytical expressions, Hennig and Schuler (2012) proposes to approximate both entropies using a linear approximation or using the expectation propagation algorithm. As it has been seen, these approaches result in a high number of approximations. Hence, the approaches of Villemonteix et al. (2009) and Hennig and Schuler (2012) are computationally very expensive and very hard to implement.

We can observe the shape of the ES acquisition function $\alpha(\mathbf{x})$ in Figure 3.11. Concretely, we can see how both exploration and exploitation are taken into account in this acquisition function $\alpha(\mathbf{x})$. Hence, we can see that its shape is different from the one displayed by acquisition functions $\alpha(\mathbf{x})$ that are not information-theoretical based.

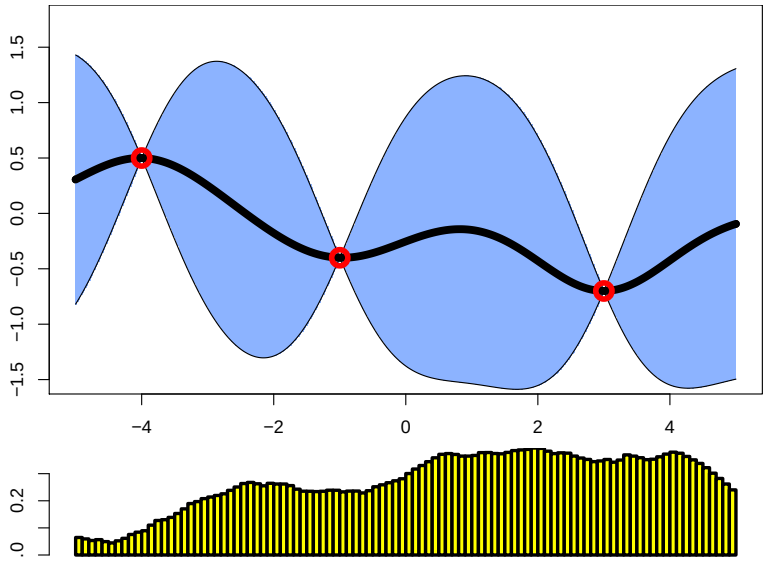

FIGURE 3.11: (Top) GP model fitting un unknown function $f(\mathbf{x})$. It is assumed that there is no additive noise. Observations are displayed in red. The prediction of the unknown function $\mu(\mathbf{x})$ is plotted as a continuous black line. The uncertainty over the unknown function $\sigma(\mathbf{x})$ is shown in blue. (Down) ES acquisition function $\alpha(\mathbf{x})$. The point $\mathbf{x}^\star$ associated with the maximum ES value $\mathbf{x}^\star = \text{argmax}_{\mathbf{x} \in \mathcal{X}} \alpha(\mathbf{x})$ is suggested for evaluation.

- **Predictive Entropy Search (PES):** As we have seen, entropy search uses various difficult approximations that would be desirable to avoid. The PES acquisition function $\alpha(\mathbf{x})$ is an ES equivalent expression that does not compute as many approximations as ES and it is easier to implement (Hernández-Lobato et al., 2014). As in the case of ES, PES maximizes the information about the location of the global extremum $\mathbf{x}^\star$. In order to alleviate the described issues of ES, it is possible to perform a trick to convert the analytical expression of ES into another analytical expression that is easier to approximate. This trick is based

on the concept of mutual information $I(X,Y)$ of random variables X and Y. In particular, PES uses the fact that mutual information $I(X,Y)$ is symmetric. Let $\mathcal{D}$ be the dataset of all evaluations processed by BO until a given iteration t, namely, $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., t\}$. The ES expression can be equivalently written as the mutual information between $\mathbf{x}^\star$ and y given $\mathcal{D}$, namely, $ES(\mathbf{x}) = I(\mathbf{x}^\star, y)$. Since the mutual information $I(X,Y)$ is a symmetric function, we can swap the roles of y and $\mathbf{x}^\star$ in the ES($\mathbf{x}$) analytical expression. By doing it so, ES can be rewritten as:

$$\text{PES}(\mathbf{x}) = H[p(y|\mathcal{D}, \mathbf{x})] - \mathbb{E}_{p(\mathbf{x}^\star|\mathcal{D})}[H[p(y|\mathcal{D}, \mathbf{x}, \mathbf{x}^\star)]]\,, \tag{3.10}$$

where $p(y|\mathcal{D}, \mathbf{x}, \mathbf{x}^\star)$ is the conditional predictive distribution for y at $\mathbf{x}$ given the observed data $\mathcal{D}$ and the location $\mathbf{x}^\star$ of the global optimum of the objective function $f(\mathbf{x})$. Concretely, the expectation is now taken on the posterior distribution of the optimizer $p(\mathbf{x}^\star|\mathcal{D})$. The main advantage with respect to ES is that this acquisition function $\alpha(\mathbf{x})$ is based on the entropies of the predictive distributions $H[p(y|\mathcal{D}, \mathbf{x})]$ and $H[p(y|\mathcal{D}, \mathbf{x}, \mathbf{x}^\star)]$, which are expected to be easier to evaluate than the entropy of $\mathbf{x}^\star$. These expressions are analytic or can be easily approximated. This is an advantage with respect to ES, where the approximations of the entropies were more challenging and costly to compute.

The first term of PES $H[p(y|\mathcal{D}, \mathbf{x})]$ corresponds to the entropy of the GP predictive distribution $p(y|\mathcal{D}, \mathbf{x})$ of the objective function $f(\mathbf{x})$ potentially contaminated with noise. This is essentially the entropy of an univariate Gaussian distribution. This computation can be solved analytically, since the entropy of an univariate Gaussian distribution has a closed-form expression. Let $p(\mathbf{x}) = N(\mathbf{x}|\mu, \sigma^2)$ be an univariate Gaussian distribution. Its entropy $H[p(\mathbf{x})]$ corresponds to the following expression: $H[p(\mathbf{x})] = \ln(\sigma^2\sqrt{2\pi\text{e}})$. Nevertheless, the second term of PES in Equation (3.10) must be approximated. First, the expectation is approximated by averaging over samples of $\mathbf{x}^\star$ drawn from $p(\mathbf{x}^\star|\mathcal{D})$. The distribution of the global minimizer given the observed data, $p(\mathbf{x}^\star|\mathcal{D})$, can be alternatively expressed as $p(f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})|\mathcal{D})$. That is, the probability that the evaluation of the function $f(\mathbf{x})$ of the point $\mathbf{x}$ is the minimum $\mathbf{x}^\star$ with respect to the input space $\mathcal{X}$. Let $\mathbf{f}$ be a vector that represents a discretization of a function that is sampled from the GP. If we have represented a function sampled from a GP by a vector $\mathbf{f}$, we can measure the probability of the ith element of $\mathbf{f}$ of being the optimum. This probability is computed as the product of the probability of the samples given data $p(\mathbf{f}|\mathcal{D})$ multiplied by a factor that represents whether the ith element is the optimum of that sample or not: $\int p(\mathbf{f}|\mathcal{D})\delta[f_i \geq f_j]d\mathbf{f}$. We can sample from this distribution by Thompson sampling, that is, sampling the GP and retrieving the index of the optimum point. By doing this process iterativelly, we obtain several samples of the extremum that in expectation represent the conditional distribution of the global extremum given the observed data $p(f(\mathbf{x}^\star) = \max_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})|\mathcal{D})$. In particular, to obtain a sample from the GP, we can use a procedure defined as random features (Rahimi et al., 2007). Random features accelerates the process of sampling functions from a GP, avoiding costly matrix operations. First, this method maps the input data to a randomized low-dimensional feature space. Concretely, the random features are designed so that the inner products of the transformed data are approximately equal to those in the feature space of popular GP covariance functions. More information about how these features are sampled can be found on the work by Rahimi et al. (2007).

For each sample of the minimizer, the entropy of the conditional predictive distribution $H[p(y|\mathcal{D}, \mathbf{x}, \mathbf{x}^\star)]$ has to be computed. The conditional predictive distribution $p(y|\mathcal{D}, \mathbf{x}, \mathbf{x}^\star)$ can be decomposed as $\int p(y|f(\mathbf{x})p(f(\mathbf{x})|\mathcal{D}, \mathbf{x}^\star)df(\mathbf{x})$. When the likelihood is Gaussian, $p(f(\mathbf{x})|\mathcal{D})$ has closed-form expression as it is the posterior distribution of the GP. But if we condition the previous expression to the global minimizer $\mathbf{x}^\star$, that is, $p(f(\mathbf{x})|\mathcal{D}, \mathbf{x}^\star)$, it includes additional constraints that are not Gaussian distributions. Hence, it is intractable to compute. Although, this conditional predictive distribution can be approximated using EP (expectation propagation) (Hernández-Lobato et al., 2015). EP provides Gaussian factor approximations to these step factors introduced as constraints on the potential values of $f(\mathbf{x})$ to account for $\mathbf{x}^\star$ being the global optimum of the function. These constraints model that the associated value of $\mathbf{x}^\star$, $f(\mathbf{x}^\star)$, must be larger than past observations and that $f(\mathbf{x}) < f(\mathbf{x}^\star)$ for the candidate point $\mathbf{x}$ at which to evaluate the acquisition. The approximated Gaussian factors computed by EP are multiplied to $p(f(\mathbf{x})|\mathcal{D})$, which results in a Gaussian approximation of $p(f(\mathbf{x})|\mathcal{D}, \mathbf{x}^\star)$, making the computation of PES feasible in practice. Importantly, it only involves EP approximations on $H[p(y|\mathcal{D}, \mathbf{x}, \mathbf{x}_*)]$ rather than in the whole expression as ES, making PES simpler to implement in practice than ES.

- **Max-Value Entropy Search (MES):** PES delivers exceptional empirical results and computes less approximations than ES (Hernández-Lobato et al., 2015). However, it relies on using EP to approximate the conditional predictive distribution $p(y|\mathbf{x}, \mathbf{x}^\star, \mathcal{D})$ non-Gaussian factors regarding the optimum $\mathbf{x}^\star$ of the problem. It would be desirable to propose a method with similar empirical performance as ES and PES without resorting to EP. Driven by that motivation, Max-Value Entropy Search (MES) uses the information about the maximum function value $y^\star$ instead of its associated location $\mathbf{x}^\star$. In this setting, we assume noiseless objective functions that give as a result $y^\star = f(\mathbf{x}^\star)$. Instead of reducing the entropy of $\mathbf{x}^\star$, MES reduces the entropy of $y^\star$ by considering the function value $y^\star$ instead of its associated location $\mathbf{x}^\star$. By doing it so, it tremendously lightens the computational burden (Wang and Jegelka, 2017). MES is able to use an entropy search derived methodology being less expensive in computation time. MES is also more robust to the number of samples used for computing the entropy, and hence more efficient for higher-dimensional problems. It is important to remark that the maximum function value $y^\star$ belongs to a one-dimensional space. In contrast to the associated location $\mathbf{x}^\star$ to this value, that belongs to the input space $\mathcal{X}$. This fact greatly facilitates the estimation of the mutual information. The mutual information is estimated via an approximation done with a Gumbel distribution (Gumbel, 1958) or a Monte Carlo approach that uses random features. More information about the approximation of the mutual information is available on (Hernández-Lobato et al., 2015).

  MES uses the information about the maximum value $y^\star = f(\mathbf{x}^\star)$ instead of the information about the location $\mathbf{x}^\star$. MES maximizes the expected reduction of the differential entropy of the unconditioned predictive distribution $p(y|\mathcal{D}_t, \mathbf{x})$ minus the expected differential entropy of the conditioned predictive distribution $p(y|\mathcal{D}_t, \mathbf{x}, y^\star)$.

More formally,

$$\begin{aligned}\text{MES}(\mathbf{x}) &= H[p(y|\mathcal{D}_t, \mathbf{x})] - \mathbb{E}(H[p(y|\mathcal{D}_t, \mathbf{x}, y^\star)]) \approx \\ &\approx \frac{1}{K}\sum_{y^\star \in Y^\star}[\frac{\phi_{y^\star}(\mathbf{x})\chi(\phi_{y^\star}(\mathbf{x}))}{2\Phi(\chi(\phi_{y^\star}(\mathbf{x}))} - \log(\phi(\chi(\phi_{y^\star}(\mathbf{x})))]\,, \end{aligned} \tag{3.11}$$

where $\phi_{y^\star}(\mathbf{x})) = \frac{y^\star - \mu_t(\mathbf{x})}{\sigma_t(\mathbf{x})}$, ϕ is the probability density function of a normal distribution and Φ the cumulative density function of a normal distribution. The previous expression is a result of approximating the conditional distribution $p(y|\mathcal{D}, \mathbf{x}, y^\star)$ by a Gaussian distribution truncated on $y^\star$. Considering the truncated Gaussian distribution is a simpler approximation of MES than the approximation that uses the conditional distribution $p(y|\mathcal{D}, \mathbf{x}, y^\star)$. K are the number of samples used to approximate the expectation. Samples of $y_\star$ are approximated via a Gumbel distribution. We can also sample functions from the GP posterior distribution and maximize them to empirically obtain samples of $y^\star$ via a Monte Carlo random features approach as in PES.

3.4 Constrained Multi-Objective Scenario

Until this section, we have considered optimization problems of a single objective function $f(\mathbf{x})$. In this section, we will give an overview of optimization problems with multiple objectives and constraints. The constrained multi-objective scenario is more difficult to solve, as we will further explain in detail in Chapters 4 and 5. As an example of this setting, let the estimation of the generalization error of a machine learning algorithm be an objective function $f_1(\mathbf{x})$. Let also be the prediction time of a machine learning algorithm another black-box function $f_2(\mathbf{x})$. We may be interested in obtaining hyperparameters that both minimize the generalization error and the prediction time. However, in most cases, a machine learning algorithm with low generalization error incurs in a high prediction time and viceversa. In particular, consider the simultaneous optimization of the estimation of the generalization error of a deep neural network trained on a dataset $\mathcal{D}$ and its prediction time for new instances $\mathbf{X}^\star$. We assume that wider deep neural networks require more time to predict an instance and generally obtain less generalization error. We can see that solutions that optimize the generalization error will incur in a higher prediction time, which is undesirable. Moreover, small deep neural networks that can optimize prediction time will provide a higher estimation error. It is hence not possible to obtain solutions that optimize both objectives in this setting. Critically, the objectives $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ are conflicting. In other words, a single global minimizer $\mathbf{x}^\star$ of both objectives, $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$, does not exist. However, BO can also be applied for this scenario. In particular, it can be applied to the simultaneous optimization of K objective functions. We define this setting as multi-objective BO. More formally, let $\mathbf{f}(\mathbf{x}) : \mathcal{X} \to \mathbb{R}^K$ define a set of K black-boxes to be optimized. The multi-objective problem is defined by minimizing this set of functions as we can see in Eq. (3.12):

$$\min_{\mathbf{x}\in\mathcal{X}} \quad f_1(\mathbf{x}), \ldots, f_K(\mathbf{x})\,. \tag{3.12}$$

Recall that the black-boxes $f_1(\mathbf{x}), \ldots, f_K(\mathbf{x})$ are conflicting. In this setting, we assume that the functions $\mathbf{f}(\mathbf{x})$ are independent. Hence, for simplicity, we model each function $f_k(\mathbf{x})$ by an independent GP. Although, it is also possible to get rid of the independence

assumption and consider, in a more advanced scenario, correlations between the GP models (Shah and Ghahramani, 2016).

In the single objective scenario, we were interested the point $\mathbf{x}^\star$ associated with the minimum evaluation value $f(\mathbf{x}^\star)$ across all the input space $\mathcal{X}$. In the multi-objective scenario, we can obtain not only one point $\mathbf{x}^\star$ but a potentially infinite set of optimal points $\mathcal{X}^\star$. These points $\mathcal{X}^\star$ represent the best trade-off between the objectives $\mathbf{f}(\mathbf{x})$. The set of the optimal points $\mathcal{X}^\star$ is defined as the *Pareto set* (Shan and Wang, 2005). Let us describe the properties of $\mathcal{X}^\star$. First, we define that a point $\mathbf{x}$ dominates another point $\mathbf{x}'$ if $f_k(\mathbf{x}) \leq f_k(\mathbf{x}')\ \forall k$, with at least one inequality being strict. The Pareto set $\mathcal{X}^\star$ is defined by the points in $\mathcal{X}$ that are not dominated by any other point in the input space $\mathcal{X}$. The size of the set is potentially infinite. The purpose of a multi-objective algorithm is to approximate the Pareto set $\mathcal{X}^\star$ by a countable set of points. If we evaluate the points of the Pareto set $\mathcal{X}^\star$ by the functions being optimized $\mathbf{f}(\mathcal{X}^\star)$ we obtain, in the image space $\mathcal{Y}$, a set of points defined as the *Pareto frontier* $\mathcal{Y}^\star$. The Pareto frontier $\mathcal{Y}^\star$ represents the values associated with the evaluation of the Pareto set $\mathcal{X}^\star$. These values $\mathcal{Y}^\star$ are the best trade-off among the objectives $\mathbf{f}(\mathbf{x})$. Hence, the Pareto set $\mathcal{X}^\star$ are the points, solutions or configurations of the multi-objective problem.

As we have said, a Pareto set $\mathcal{X}$ is a set of points that best associated trade-off among the objectives $\mathbf{f}(\mathbf{x})$ of the optimization problem. However, the user has to decide a single configuration among the Pareto set $\mathcal{X}$. Hence, we face the problem of which point of the Pareto set $\mathcal{X}$ is the best one for the user preferences. To solve this problem, the user can define a set of preferences, that is, criteria that help to determine which is the best point of the Pareto set for the user. Let us recover the example of the deep neural network optimization problem where we simultaneously minimize the estimation of the generalization error and the prediction time. Recall that the objectives were conflicting. In this scenario, an example of a simple preference can be that an user may prefer to recover the configuration of the Pareto set that minimizes the prediction time, independently of the estimation of the generalization error. Preferences can be more complex. For example, we can assign a weight to each of the objectives and evaluate the Pareto set solutions according to the values of its associated Pareto frontier $\mathcal{Y}$. More generally, we can also define a score given by an analytical expression that combines each of the objectives of the associated Pareto frontier $\mathcal{Y}$ according to the user needs. The retrieved configuration will be the one that gives the best score.

When we optimize a single objective function $f(\mathbf{x})$, we can evaluate its solution $\mathbf{x}$ by minimizing its regret r with respect to the optimum of the problem $\mathbf{x}^\star$. Recall that the regret can be computed as: $r = |f(\mathbf{x}^\star) - f(\mathbf{x})|$. This metric is not applicable to the multi-objective scenario. In this case, we need to define a different metric to evaluate the quality of an estimate of the Pareto set $\hat{\mathcal{X}}^\star$. The metric that is usually used to evaluate the quality of an estimated Pareto set $\hat{\mathcal{X}}^\star$ is the hypervolume. The hypervolume is the area covered by the Pareto frontier $\hat{\mathcal{Y}}^\star$ with respect to a reference point in the space. There exist several mechanisms to efficiently compute the hypervolume metric (While et al., 2006). For a two-objective problem we can visualize this area by plotting the estimated Pareto frontier $\hat{\mathcal{Y}}^\star$ associated at the estimate of $\hat{\mathcal{X}}^\star$. We will consider the best method the one whose Pareto frontier $\mathcal{Y}^\star$ is wider. Figure 3.12 illustrates curves computed by interpolating the points belonging to the Pareto frontiers $\hat{\mathcal{Y}}^\star$ associated to different methods trying to approximate the Pareto set $\mathcal{X}^\star$ of the problem described in previous paragraphs. The method whose associated Pareto frontier $\hat{\mathcal{Y}}^\star$ is the blue curve would be the one with highest hypervolume since it is the one with the largest area above it.

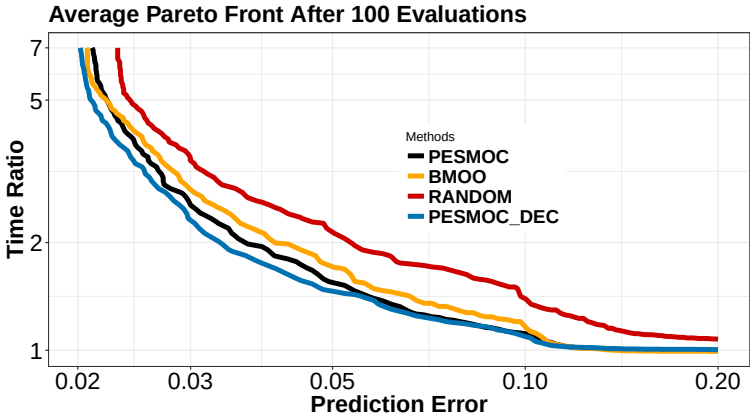

FIGURE 3.12: Pareto frontiers computed by the solutions delivered by four different Multi-objective BO methods, each one represented in one colour. The solutions are the best trade-off between the prediction error of the deep neural network and the prediction time normalized with respect to the smallest possible prediction time, *i.e.*, the time ratio.

The described scenario can be generalized to the broader constrained scenario. In this scenario, the set of objectives $\mathbf{f}(\mathbf{x})$ is optimized under the presence of a set of C constraint functions $c_1(\mathbf{x}), ..., c_C(\mathbf{x})$. As in the case of the objective functions $\mathbf{f}(\mathbf{x})$, these constraints $\mathbf{c}(\mathbf{x})$ are also independent black-boxes. We will also model these constraints $\mathbf{c}(\mathbf{x})$ by independent GPs. We formalize the problem in Equation (3.13):

$$\min_{\mathbf{x}\in\mathcal{X}} \quad f_1(\mathbf{x}), \ldots, f_K(\mathbf{x}) \qquad \text{s.t.} \qquad c_1(\mathbf{x}) \geq 0, \ldots, c_C(\mathbf{x}) \geq 0\,. \tag{3.13}$$

In this setting, all the candidate points $\mathbf{x} \in \mathcal{X}^\star$ have to validate the non-negativity of all the constraints, *i.e.*, $c_c(\mathbf{x}) \geq 0 \; \forall \; c$. We define the *feasible space* $\mathcal{F}$ as the subset of the input space $\mathcal{F} \subset \mathcal{X}$ whose points satisfy all the constraints. Recall that we cannot evalaute the real constraints $\mathbf{c}(\mathbf{x})$ for all $\mathcal{F}$. Hence, we use the GP predictive distribution to determine which points $\mathbf{x} \in \mathcal{X}$ are expected to be feasible in each step of BO. If a point is determined as feasible, *i.e.*, the probability of being feasible given by all the GP predictive distributions is above a predefined threshold, we can compute its hypervolume. However, if a recommended point turns out to be infeasible according to the evaluations of the constraints, we set its associated hypervolume to zero.

Figure 3.13 illustrates the constrained multi-objective scenario with its associated objectives, constraints, Pareto set and Pareto frontier. In the first row of the figure, we can see the shape of each black-box, $\mathbf{f}(\mathbf{x})$ and $\mathbf{c}(\mathbf{x})$. In the figure of the left of the second row, we can observe the feasible space $\mathcal{F}$ as black dots that are distributed over the input space $\mathcal{X}$. We can also see its associated values in the feasible image space $\mathcal{Y}$, represented by black dots, in the figure of the right. The feasible Pareto set points $\mathcal{F}^\star$ are represented by red dots. Critically, we can observe how the feasible Pareto set $\mathcal{F}^\star$ is a subset of the feasible space $\mathcal{F}$. We can observe its associated frontier $\mathcal{Y}^\star$ in the figure of the right. Recall that the points of the Pareto frontier $\mathcal{Y}^\star$ dominate the other associated values of the feasible space $\mathcal{F}$ points.

Constrained multi-objective problems demand specific Bayesian optimization techniques, as the acquisition functions described in Section 3.3 do not work in this scenario. Chapter 4 describes the predictive entropy search for multi-objective Bayesian optimization with constraints method (Garrido-Merchán and Hernández-Lobato, 2019b), that is a

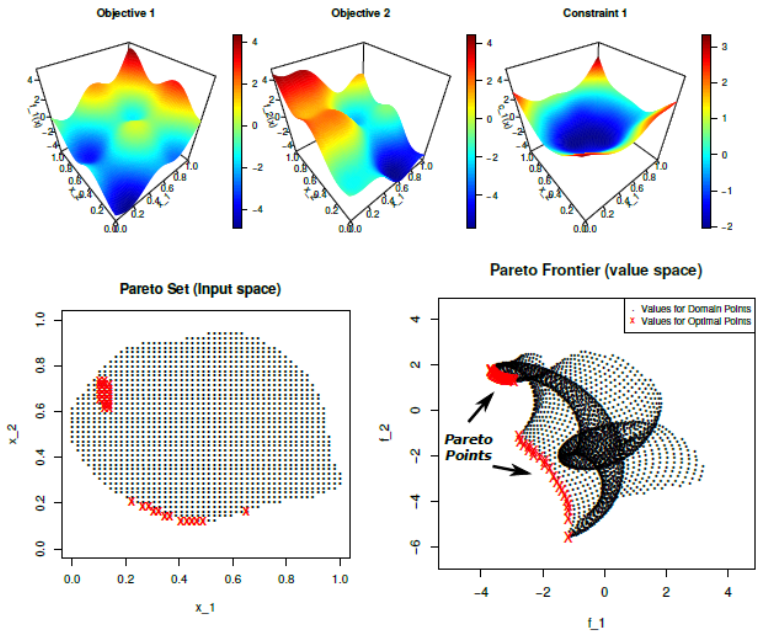

FIGURE 3.13: Constrained multi-objective optimization problem with two objectives and one constraint. (Top) Shapes of the objectives and constraints of the constrained multi-objective optimization problem. (Bottom, left) Feasible space represented by black dots and feasible Pareto set represented by red dots. (Bottom, right) Associated values of the feasible space represented by black dots and Pareto frontier represented by red dots.

generalization of the Predictive entropy search acquisition function described in Section 3.3.4 adapted to the constrained multi-objective scenario. Another acquisition function that tackles this scenario is an extension of the expected improvement for the constrained multi-objective scenario that is referred as BMOO, from Bayesian Multi-Objective Optimization (Féliot et al., 2017). We also describe this approach more in detail in Chapter 4. Alternatively, we can always perform a random search of the input space $\mathcal{X}$, but, in this scenario, it will also incur in the issues described in Section 3.1.

3.5 Bayesian Optimization Software

Several BO free software tools are available online. Some examples are Spearmint, BoTorch, Skopt, SMAC, GPyOpt or mlrMBO (Balandat et al., 2019; Bischl et al., 2017a; González, 2016; Hutter et al., 2011; Markov, 2017; Snoek et al., 2012). All the methods for BO described in this thesis have been implemented in the Spearmint tool, available in this link `https://github.com/EduardoGarrido90/Spearmint`.

Spearmint is a Python 2.7 tool that can be used to optimize black-boxes (Snoek et al., 2012). It allows to optimize a black-box $f(\mathbf{x})$ whose implementation can be written in several languages in a separate file. This file is imported in the tool via a configuration file, where we can specify several options for BO. Some of these options include the specification of multiple black-boxes as objectives $\mathbf{f}(\mathbf{x})$ and constraints $\mathbf{c}(\mathbf{x})$. We can specify different acquisition functions such as EI or PES. We can also specify whether the variables $\mathcal{X}$ involved in the optimization process are real, integer or categorical valued. It is also possible to use different GP covariance functions $k(\mathbf{x}, \mathbf{x}')$. We can

make different transformations for the input space variables $\mathcal{X}$ such as the beta-warp transformation. Recall that this transformation is used for non-stationary input spaces $\mathcal{X}$. Spearmint delivers a recommendation after the configured BO number of iterations based on the best observation value or the optimization of the GP mean. We can specify the number of GP hyper-parameter samples to use for the optimization process. Each GP has hyper-parameters $\boldsymbol{\theta}$ obtained via slice sampling of the hyper-parameter distribution $p(\boldsymbol{\theta}|\mathbf{y})$. Spearmint averages the acquisition function $\alpha(\mathbf{x})$ and predictive distribution $p(y|\mathbf{x}, \mathcal{D})$ of the black-box $f(\mathbf{x})$ across the configured number of GP hyper-parameter samples. Spearmint implements a Sobol and uniform grid to avoid local optima in the optimization of the acquisition function and the GP mean. The resolution of those grids is 1000 multiplied by the number of dimensions D of the problem. Spearmint has been extensively used by researchers to implement novel BO methodologies. The tool can be downloaded in this link `https://github.com/EduardoGarrido90/Spearmint`.

BoTorch is the BO module of the deep learning Python 3 framework pyTorch (Balandat et al., 2019; Ketkar, 2017). It is an extensible modular library that enables the implementation of new probabilistic models, acquisition functions and optimizers. It uses autograd for automatic differentiation (Maclaurin et al., 2015). Autograd avoids having to analytically derive gradients of acquisition functions. It enables parallel multi-objective BO with a parallelization of the expected hypervolume criterion. BoTorch, as Spearmint, uses GPs as the probabilistic surrogate model but it also contains multi-task GPs, scalable and deep GPs. It can even work in a model-agnostic way, letting the user specify a predictive distribution $p(y|\mathbf{x}, \mathcal{D})$ of the evaluation of the objective function independently of the model used to build the acquisition function $\alpha(\mathbf{x})$. The target audience of this tool are researchers and sophisticated practitioners in BO. More information about BoTorch can be found in this link `https://botorch.org/`.

Skopt is the BO module of the famous machine learning Python 3 library library Scikit-learn (Markov, 2017; Pedregosa et al., 2011). As the previous tools, Skopt uses a GP as the probabilistic surrogate model. It includes methods to plot the evaluation of the suggestions that the optimization is proposing. The main motivation of using this library is that it is a part of the Scikit-learn framework. Nevertheless, it only includes basic functionality and acquisition functions. As it is currently developed, it is tool for a quick prototype of BO applications, but the rest of the tools described in this section are more recommended than Skopt for BO research or deployment. More information about this module is accesible in the Skopt BO web page `https://scikit-optimize.github.io/stable/auto_examples/bayesian-optimization.html`.

SMAC is an alternative to the rest of GP based BO frameworks which uses random forests as the probabilistic model surrogate (Hutter et al., 2011). It provides a flexible API for Python 3 to define the input space $\mathcal{X}$ and optimize black-boxes $\mathbf{f}(\mathbf{x})$. It has been used for AutoML. SMAC is suited for AutoML as random forest naturally deal with hierarchical spaces and high dimensional spaces. Hence, we can not only optimize the hyper-parameters of a single machine learning algorithm. Instead, we can simultaneously optimize a function with respect to a set of machine learning algorithms along with their hyper-parameters. Examples of automatic machine learning tools that use SMAC are Auto-Weka and Auto-sklearn (Feurer et al., 2019; Thornton et al., 2013). SMAC is available at the following location: `https://www.automl.org/automated-algorithm-design/algorithm-configuration/smac/`.

GPyOpt is a Python 3 BO library included as an extension of the GP regression library GPy `https://sheffieldml.github.io/GPy/`. GpyOpt was written and it was maintained by the ML group at Sheffield University (González, 2016). It is currently

not maintained anymore, so although GPy is an excellent library of GPs, GPyOpt may not be the best option for BO in production. It includes different acquisition functions $\alpha(\mathbf{x})$, parallel BO, cost aware BO and mixed variable optimization. It also handles big data sets by using an implementation of sparse GPs. GPyOpt is available at the web site `https://github.com/SheffieldML/GPyOpt`.

mlrMBO, written on R, is an alternative to the previous libraries which are coded on Python (Bischl et al., 2017a). It is an extension of the machine learning R framework mlr3 (Bischl et al., 2016). mlrMBO is a configurable R toolbox for BO of black-box functions $\mathbf{f}(\mathbf{x})$. It provides support for multi-objective BO, mixed variable optimization, noisied black-boxes, parallel BO, visualizations, and several acquisition functions $\alpha(\mathbf{x})$. It contains GPs and random forests as probabilistic surrogate models. It is available for download at the web page `https://mlrmbo.mlr-org.com/`.

3.6 Conclusions

In this chapter, we have studied fundamental concepts about BO. In particular, the BO algorithm is defined by two components: a probabilistic surrogate model of the objective function $f(\mathbf{x})$ and the acquisition function $\alpha(\mathbf{x})$. Acquisition functions are the criteria that suggest, in each iteration, a point $\mathbf{x}$ to be evaluated. No single acquisition function $\alpha(\mathbf{x})$ is better for all optimization problems. Acquisition functions $\alpha(\mathbf{x})$ are criteria that represent an exploration-exploitation trade-off. For problems whose objective function $f(\mathbf{x})$ is convex or unimodal, acquisition functions $\alpha(\mathbf{x})$ that favour more exploitation are preferred, as for example expected improvement. On the other hand, if the objective function $f(\mathbf{x})$ is supposed to be very complex and multi-modal, acquisition functions $\alpha(\mathbf{x})$ like predictive entropy search are preferred, as they favour more exploration. In this chapter, we have included a detailed list with the most popular acquisition functions $\alpha(\mathbf{x})$ in BO. The other basic component of BO are probabilistic surrogate models, which have been described in the previous chapter.

We have also introduced how BO can be applied to constrained multi-objective scenarios. In this setting, each objective $f(\mathbf{x})$ and constraint $c(\mathbf{x})$ is modelled by an independent probabilistic surrogate model. We have seen that BO is a powerful methodology to be applied in practice, so we also included a section where we have described some of the most popular BO free software. This software is available in Python and R.

Chapter 4

Predictive Entropy Search For Multi-Objective Bayesian Optimization With Constraints

This chapter describes the first proposed approach in this thesis, PESMOC, Predictive Entropy Search for Multi-objective Bayesian Optimization with Constraints. PESMOC is an information-based strategy for the simultaneous optimization of multiple expensive-to-evaluate black-box functions under the presence of several constraints. It is an enhancement of the previously shown PES acquisition function, adapted to a constrained multi-objective scenario. Concretely, it expands the PESM approach, that applied PES to multiple objectives and the PESC approach, that applied PES to multiple constraints. PESMOC gains the best of the two worlds, being able to solve multiple objectives restricted to multiple constraints. The constraints considered in PESMOC have similar properties to those of the objectives in typical Bayesian optimization problems. In this chapter, we present strong empirical evidence in the form of synthetic, benchmark and real-world experiments that illustrate the effectiveness of PESMOC with respect to other methods that are also going to be explained in this chapter. In the multi-objective setting, acquisition functions are a linear combination of the information provided by each black-box. We will see that, in PESMOC, the acquisition function is decomposed as a sum of objective and constraint specific acquisition functions. This enables the use of PESMOC in decoupled evaluation scenarios in which objectives and constraints can be evaluated separately, each of them with different costs. The results obtained also show that a decoupled evaluation scenario can lead to significant improvements over a coupled one in which objectives and constraints are evaluated at the same input.

4.1 Introduction

Many practical problems involve the simultaneous optimization of several objectives subject to a set of constraints being simultaneously satisfied. Furthermore, often these functions are black-boxes, meaning that we will not have access to their analytical form, and the time needed for their evaluation can be fairly large. An example is tuning the control system of a four-legged robot. We may be interested in finding the optimal control parameters to minimize the robot's energy consumption and maximize locomotion

speed (Ariizumi et al., 2014), under the constraint that the amount of weight placed on a leg of the robot does not exceed a specific value, or similarly, that the maximum angle between the legs of the robot is below some other value for safety reasons. Measuring the objectives and the constraints in this case may involve an expensive computer simulation or doing some actual experiment with the robot. There is no analytical expression to describe the output of that process which can take a significant amount of time. Another example can be found in the design of a new type of low-calorie cookie (Gelbart et al., 2014). In this case, the parameter space is the space of possible recipes and baking times. Here we may be interested in minimizing the number of calories per cookie and maximizing tastiness. Moreover, we may also want to keep production costs below a particular level or we may want that the cookie is considered to be crispy for at least 90% of the population. A last example considers finding the architecture of a deep neural network and training parameters to simultaneously maximize prediction accuracy on some task and minimize prediction time. We may also be interested in codifying such network in a chip so that the energy consumption or its area is below a particular value.

As we have explained in previous chapters, Bayesian Optimization (BO) has been proved to be a good technique to tackle optimization problems with the characteristics described above (Mockus et al., 1978). Namely, problems in which one does not have access to the analytic expression of the objectives or the constraints, and can only obtain (potentially noisy corrupted) values for some input by running some expensive process. In BO methods an input location on which the objectives and constraints are evaluated is iteratively suggested in an intelligent way. The aim is finding the solution of the problem with the smallest possible number of evaluations of the objectives and constraints (Brochu et al., 2010; Shahriari et al., 2015). At each iteration, the observations collected so far are carefully used for this task. Moreover, because evaluating each black-box function is expected to be very expensive, the time needed to suggest a candidate point is considered negligible.

In the literature, there are several BO methods that have been proposed to efficiently address multi-objective problems (Emmerich, 2008; Hernández-Lobato et al., 2016; Knowles, 2006; Picheny, 2015; Ponweiser et al., 2008) and also constraint optimization problems (Gelbart et al., 2014; Hernández-Lobato et al., 2016; Hernández-Lobato et al., 2015). However, the problem of considering several objectives and several constraints at the same time has received significantly less attention from the BO community, with a few exceptions (Féliot et al., 2017). In this chapter we provide a practical BO method based on information theory that can address this type of problems.

More precisely, in this chapter, we consider the problem of simultaneously minimizing K functions $f_1(\mathbf{x}), ..., f_K(\mathbf{x})$ which we define as objectives, subject to the non-negativity of C constraints $c_1(\mathbf{x}), ..., c_C(\mathbf{x})$, over some bounded domain $\mathcal{X} \in \mathbb{R}^d$, where d is the dimensionality of the input space. The problem considered is:

$$\min_{\mathbf{x} \in \mathcal{X}} \quad f_1(\mathbf{x}), \ldots, f_K(\mathbf{x}) \qquad \text{s.t.} \qquad c_1(\mathbf{x}) \geq 0, \ldots, c_C(\mathbf{x}) \geq 0\,. \tag{4.1}$$

We say that a point $\mathbf{x} \in \mathcal{X}$ is feasible if $c_j(\mathbf{x}) \geq 0$, $\forall j$, that is, it satisfies all the constraints. This leads to the concept of feasible space $\mathcal{F} \subset \mathcal{X}$, that is the set of points that are feasible. In this scenario, only the solutions contained in $\mathcal{F}$ are considered valid.

Focusing in the multi-objective optimization part of the problem, most of the times is impossible to optimize all the objective functions at the same time, as they may be conflicting. For example, in the control system of the robot described before, most probably maximizing locomotion speed will lead to an increase in the energy consumption.

Similarly, in the low-calorie cookie example, minimizing the number of calories will also decrease tastiness, and in the neural network example minimizing the prediction error will involve using bigger networks, increasing the prediction time. In spite of this, it is still possible to find a set of optimal points $\mathcal{X}^\star$ known as the *Pareto set* (Siarry and Collette, 2003). More formally, we define that the point $\mathbf{x}$ dominates the point $\mathbf{x}'$ if $f_k(\mathbf{x}) \leq f_k(\mathbf{x}')\ \forall k$, with at least one inequality being strict. Then, the Pareto set is the subset of non-dominated points in $\mathcal{F}$, the feasible space, which is equivalent to this expression $\forall \mathbf{x}^\star \in \mathcal{X}^\star \subset \mathcal{F}\,, \forall \mathbf{x} \in \mathcal{F}\, \exists\, k \in 1, ..., K$ such that $f_k(\mathbf{x}^\star) < f_k(\mathbf{x})$. The Pareto set is considered to be optimal because for each point in that set one cannot improve in one of the objectives without deteriorating some other objective. Given $\mathcal{X}^\star$, a final user may then choose a point from this set according to their preferences, *e.g.*, locomotion speed vs. energy consumption.

To solve efficiently the previous problems, *i.e.*, find the Pareto set in $\mathcal{F}$ with a small number of evaluations, BO methods fit a probabilistic model, typically, a Gaussian process (GP) to the observed data of each black-box function (objective or constraint). The uncertainty about the potential values of these functions given by the predictive distribution of the GPs is then used to build an acquisition function. The maximum of this function indicates the most promising location on which to evaluate next the objectives and the constraints to solve the optimization problem. After enough observations have been collected like this, the probabilistic models can be optimized to provide an estimate of the Pareto set of the original problem. Importantly, the acquisition function only depends on the uncertainty provided by the probabilistic models and not on the actual objectives or constraints. This means that it can be evaluated and optimized very quickly to identify the next evaluation point. By carefully choosing the points on which to evaluate the objectives and the constraints, BO methods find a good estimate of the solution of the original optimization problem with a small number of evaluations (Brochu et al., 2010; Shahriari et al., 2015).

In this chapter, we describe a strategy for constrained multi-objective optimization. For this, we extend previous work that uses information theory to optimize several objectives (D. et al., 2016) or a single objective with several constraints (Hernández-Lobato et al., 2015). The result is a strategy that can handle several objectives and several constraints at the same time. The proposed strategy chooses the next point on which to evaluate the objectives and the constraints as the one that is expected to reduce the most the uncertainty about the Pareto set in the feasible space, measured in terms of Shannon's differential entropy. Intuitively, a smaller entropy implies that the Pareto set is better-identified (Hennig and Schuler, 2012; Hernández-Lobato et al., 2014; Villemonteix et al., 2009). The proposed approach is called Predictive Entropy Search for Multi-objective Bayesian Optimization with Constraints (PESMOC).

Importantly, in PESMOC the acquisition function is expressed as a sum of acquisition functions, one for each objective and constraint. This enables the use of PESMOC in decoupled scenarios in which one can choose to only evaluate a subset of objectives and constraints at any given location, each time. More precisely, PESMOC not only gives information about what input location gives more information about the problem, but also about what objective or constraint or subset of these to evaluate next. This may have important applications in practice. Consider the robot's example described before. One might be able to decouple the problems by estimating energy consumption from a simulator, even if the locomotion speed and the constraints could only be evaluated by running a real experiment with the robot. In the low-calorie cookie, calories can be a simple function of the ingredients. However, measuring crispness could require

human trials. Therefore, their evaluation could be decoupled. Similarly, in the neural network example measuring prediction error could require training the network. However, measuring power consumption may only require running an expensive simulation. The benefit of decoupled evaluations in the context of BO has been already observed in the case of a single objective and several constraints (Hernández-Lobato et al., 2016), and in the case of several objectives and no constraints (Hernández-Lobato et al., 2016). In this chapter we show that PESMOC can give significantly better results in decoupled evaluation scenarios in problems involving several objectives and several constraints at the same time.

This chapter is organized as follows: Section 4.2 describes the proposed approach, PESMOC, and how to compute an approximate acquisition function based on the expected reduction of the entropy. Section 4.3 reviews important work related to the problem or techniques employed to solve multi-objective optimization problems under the presence of constraints and also previous approaches that also allow for decoupled evaluations. Section 4.4 describes several experiments where, using multiple synthetic, benchmark and real-world problems, we show that the proposed approach has significant advantages over current state-of-the-art methods. Finally, Section 4.5 gives the conclusions of this chapter.

4.2 Predictive Entropy Search for Multi-objective Optimization with Constraints

The proposed method, PESMOC, maximizes the information gain about the solution of the problem. Namely, the Pareto set $\mathcal{X}^\star$ over the feasible set $\mathcal{F}$. $\mathcal{X}^\star$ is assumed to be unknown and can be regarded as a random variable. The uncertainty about $\mathcal{X}^\star$ arises naturally from considering a probabilistic model for each objective and constraint of the problem. More precisely, each model will output a posterior probability distribution for a particular black-box function. This posterior distribution summarizes the potential values that the function can take given the observations collected so far. Therefore, the uncertainty about the objectives and constraints is directly translated into uncertainty about $\mathcal{X}^\star$.

4.2.1 Modeling Black-box Functions Using Gaussian Processes

A critical point of BO methods is therefore building a probabilistic model for each black-box function. Such a model must provide a predictive distribution for the potential values of the function at each point of the input space. This predictive distribution will be used to guide the search, by focusing only on those regions of the input space that are expected to provide the most information about the solution of the optimization problem. Typically, the model employed is a Gaussian process (GP) (Rasmussen, 2003), although other models such as random forests, T-Student processes or deep neural networks are possible (Hutter et al., 2011; Shah et al., 2014; Snoek et al., 2015).

A GP is a prior distribution over functions. We assume each black-box function $f(\mathbf{x})$ has been generated from a GP, which is characterized by a zero mean and a covariance function $k(\mathbf{x}, \mathbf{x}')$, that is, $f(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}'))$. The particular characteristics assumed for $f(\cdot)$ (*e.g.*, level of smoothness, additive noise, etc.) are specified by the covariance function $k(\mathbf{x}, \mathbf{x}')$, which receives as an input two points, $\mathbf{x}$ and $\mathbf{x}'$ at which the covariance between $f(\mathbf{x})$ and $f(\mathbf{x}')$ has to be evaluated. A typical covariance function employed for BO is the Matérn function (Snoek et al., 2012).

Given some observations $\mathcal{D} = (\mathbf{x}_i, y_i)_{i=1}^N$ of the black-box function, where $y_i = f(\mathbf{x}_i) + \epsilon_i$ with ϵ_i some additive Gaussian noise, a GP builds a predictive distribution for the potential values of $f(\cdot)$ at a new input point $\mathbf{x}^\star$. This distribution is Gaussian. Namely, $p(f(\mathbf{x}^\star)|\mathbf{y}) = \mathcal{N}(f(\mathbf{x}^\star), m(\mathbf{x}^\star), v(\mathbf{x}^\star))$, where the mean and variance are respectively given by

$$m(\mathbf{x}^\star) = \mathbf{k}_\star^T (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y}\,, \tag{4.2}$$

$$v(\mathbf{x}^\star) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}_\star^T (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{k}_\star\,, \tag{4.3}$$

where $\mathbf{y} = (y_1, \ldots, y_N)^{\mathrm{T}}$ is a vector with the observations collected so far; σ^2 is the variance of the additive Gaussian noise; $\mathbf{k}_\star$ is a N-dimensional vector with the prior covariances between $f(\mathbf{x}^\star)$ and each $f(\mathbf{x}_i)$; and $\mathbf{K}$ is a $N \times N$ matrix with the prior covariances among each $f(\mathbf{x}_i)$, for $i = 1, \ldots, N$. See (Rasmussen, 2003) for further details.

In practice, however, a GP has some hyper-parameters that need to be adjusted. These include the variance of the additive Gaussian noise σ^2, but also any potential hyper-parameter of the covariance function $k(\cdot, \cdot)$. These can be, *e.g.*, the amplitude and the length-scales. Instead of finding point estimates for these hyper-parameters, an approach that has shown good empirical results is to compute an approximate posterior distribution for them using slice sampling (Snoek et al., 2012). The previous predictive distribution is then simply averaged over the generated samples of the hyper-parameters. The process of generating these samples and computing the final predictive distribution takes only a few seconds at most. This time can be considered negligible compared to the cost of evaluating the actual black-box function.

4.2.2 Specification of the Acquisition Function

Let the K black-box objectives of the optimization problem $\{f_1, \ldots, f_K\}$ be denoted with $\mathbf{f}$ and the C black-box constraints $\{c_1, \ldots, c_C\}$ with $\mathbf{c}$. We will assume a GP model for each of these functions, as described in the previous section. For simplicity, we will consider first a coupled setting, in which all functions are evaluated at the same candidate input location at each iteration. Later on, we will describe the extension to a decoupled evaluation setting in which only a subset of the objectives or constraints need to be evaluated each time.

Let $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ denote all the observations collected up to step N of the optimization process, where $\mathbf{y}_n$ is a $K + C$-dimensional vector with the values resulting from the evaluation of the K objectives and the C constraints at step n, and $\mathbf{x}_n$ is a vector in the input space representing the corresponding input location. In PESMOC, the next point $\mathbf{x}_{N+1}$ on which the objectives and constraints should be evaluated is chosen as the one that maximizes the expected reduction, after the corresponding evaluation, of the differential entropy $H(\cdot)$ of the posterior distribution over the Pareto set $\mathcal{X}^\star$ in the feasible space $\mathcal{F}$, $p(\mathcal{X}^\star|\mathcal{D})$. More precisely, the acquisition function $\alpha(\cdot)$ of PESMOC is

$$\alpha(\mathbf{x}) = H(\mathcal{X}^\star|\mathcal{D}) - \mathbb{E}_{\mathbf{y}}[H(\mathcal{X}^\star|\mathcal{D} \cup \{(\mathbf{x}, \mathbf{y})\})]\,, \tag{4.4}$$

where $H(\mathcal{X}^\star|\mathcal{D})$ is the entropy of $\mathcal{X}^\star$ given by the current probabilistic models; $H(\mathcal{X}^\star|\mathcal{D} \cup \{(\mathbf{x}, \mathbf{y})\})$ is the entropy of $\mathcal{X}^\star$ after including the observation $(\mathbf{x}, \mathbf{y})$ in $\mathcal{D}$; and the expectation is taken with respect to the potential values of $\mathbf{y}$ at $\mathbf{x}$, given by the predictive distribution of the GP models. Namely, the posterior distribution of the potentially noisy

evaluations of the objectives $\mathbf{f}$ and constraints $\mathbf{c}$, at $\mathbf{x}$. This distribution is

$$p(\mathbf{y}|\mathcal{D},\mathbf{x}) = \prod_{k=1}^{K} p(y_k|\mathcal{D},\mathbf{x}) \prod_{j=1}^{C} p(y_{K+j}|\mathcal{D},\mathbf{x}) \,, \tag{4.5}$$

under the assumption of independence among objectives and constraints. This assumption is maintained in the rest of the chapter. Each $p(y_k|\mathcal{D},\mathbf{x})$ and $p(y_{K+j}|\mathcal{D},\mathbf{x})$ in the previous expression is simply given by the predictive distribution described in Section 4.2.2, in which the variance of the additive Gaussian noise σ^2 is added to the predictive variance $v(\mathbf{x})$. The next point at which the objectives and constraints should be evaluated is hence chosen by PESMOC simply as $\mathbf{x}_{N+1} = \arg\max_{\mathbf{x}\in\mathcal{X}} \alpha(\mathbf{x})$.

Choosing the next point on which to perform the evaluation of the objectives and constraints as the one that reduces the most the entropy of the solution of the optimization problem is known in the BO literature as entropy search (Hennig and Schuler, 2012; Villemonteix et al., 2009). Nevertheless, the practical evaluation of (4.4) is very challenging since it involves the entropy of a set of points, the Pareto set $\mathcal{X}^\star$, of potentially infinite size. Thus, in general, the exact evaluation of this expression is infeasible and it must be approximated. For this, we perform a reformulation of the previous acquisition function that significantly simplifies its evaluation. Following Houlsby et al. (2012) and Hernández-Lobato et al. (2014), we note that (4.4) is simply the mutual information between $\mathcal{X}^\star$ and $\mathbf{y}$, $I(\mathcal{X}^\star;\mathbf{y})$. Since the mutual information is symmetric, *i.e.*, $I(\mathcal{X}^\star;\mathbf{y}) = I(\mathbf{y};\mathcal{X}^\star)$, the roles of $\mathcal{X}^\star$ and $\mathbf{y}$ can be swapped, leading to the following simplified but equivalent expression to (4.4). Namely,

$$\alpha(\mathbf{x}) = H(\mathbf{y}|\mathcal{D},\mathbf{x}) - \mathbb{E}_{\mathcal{X}^\star}[H(\mathbf{y}|\mathcal{D},\mathbf{x},\mathcal{X}^\star)] \,, \tag{4.6}$$

where the expectation is now with respect to the posterior distribution of the Pareto set, $\mathcal{X}^\star$ in the feasible space, given the observed data, $\mathcal{D}$; $H(\mathbf{y}|\mathcal{D},\mathbf{x},\mathcal{X}^\star)$ measures the entropy of $p(\mathbf{y}|\mathcal{D},\mathbf{x})$, *i.e.*, the predictive distribution for the objectives and the constraints at $\mathbf{x}$ given $\mathcal{D}$; and $H(\mathbf{y}|\mathcal{D},\mathbf{x},\mathcal{X}^\star)$ measures the entropy of $p(\mathbf{y}|\mathcal{D},\mathbf{x},\mathcal{X}^\star)$, *i.e.*, the same predictive distribution conditioned to $\mathcal{X}^\star$ being the solution of the optimization problem. This alternative formulation significantly simplifies the evaluation of the acquisition function $\alpha(\cdot)$ because we no longer have to evaluate the entropy of $\mathcal{X}^\star$, which can be very complicated. Importantly, the acquisition function obtained in (4.6) favors the evaluation in regions of the input space in which $\mathcal{X}^\star$ (the solution of the optimization problem) is more informative about $\mathbf{y}$. These are also the regions in which $\mathbf{y}$ is more informative about $\mathcal{X}^\star$. We refer to the expression in (4.6) as Predictive Entropy Search for Multi-objective Optimization with Constraints (PESMOC).

We now give the details about how to evaluate (4.6), approximately. Note that the first term in the r.h.s. of (4.6) is simply the entropy of the predictive distribution of the GP models, $p(\mathbf{y}|\mathcal{D},\mathbf{x})$, which is a factorizing $K+C$-dimensional Gaussian distribution. Therefore,

$$H(\mathbf{y}|\mathcal{D},\mathbf{x}) = \frac{K+C}{2}\log(2\pi e) + \sum_{k=1}^{K} 0.5\log(v_k^{\text{PD}}) + \sum_{j=1}^{C}\log(s_j^{\text{PD}}) \,, \tag{4.7}$$

where v_k^{PD} and s_j^{PD} are the predictive variances of the objectives and the constraints at $\mathbf{x}$, respectively. The difficulty comes from the evaluation of the second term in the r.h.s. (4.6), which is intractable and has to be approximated. For this, we follow

(Hernández-Lobato et al., 2016, 2014) and use a Monte Carlo estimate of the expectation. This estimate is obtained by drawing samples of the Pareto set $\mathcal{X}^\star$ given $\mathcal{D}$. This involves sampling several times the objectives and the constraints from their posterior distributions given by the GP models. This is done following the approach based on random features described in (Hernández-Lobato et al., 2016, 2014). Given a sample of the objectives and the constraints, we solve the corresponding optimization problem to generate a sample of $\mathcal{X}^\star$. For this, we use a grid search approach, although more efficient methods based on evolutionary strategies may be used in the case of high dimensional spaces. $\mathcal{X}^\star$ needs to be located in the feasible space $\mathcal{F}$. Thus, we discard all input grid locations in which the sampled constraints are strictly negative. $\mathcal{X}^\star$ is simply obtained by returning all the non-dominated grid locations. Note that unlike the true objectives and constraints, the sampled functions can be evaluated very fast. Given a sample of $\mathcal{X}^\star$, the differential entropy of $p(\mathbf{y}|\mathcal{D}, \mathbf{x}, \mathcal{X}^\star)$ is estimated using expectation propagation as described next.

4.2.3 EP Approximation of the Conditional Predictive Distribution

We employ expectation propagation (EP) to approximate the entropy of the conditional predictive distribution (CPD) $p(\mathbf{y}|\mathcal{D}, \mathbf{x}, \mathcal{X}^\star)$ (Minka, 2001a). Consider the deterministic distribution $p(\mathcal{X}^\star|\mathbf{f}, \mathbf{c})$ of the Pareto set in the feasible space given specific values for the objectives and the constraints. The value of $p(\mathcal{X}^\star|\mathbf{f}, \mathbf{c})$ should be zero for any set of points that is different from the actual Pareto set for the specific values of $\mathbf{f}$ and $\mathbf{c}$. $\mathcal{X}^\star$ is the Pareto set in the feasible space $\mathcal{F}$ if and only if $\forall \mathbf{x}^\star \in \mathcal{X}^\star$, $\forall \mathbf{x}' \in \mathcal{X}$, $c_j(\mathbf{x}^\star) \geq 0$ $\forall j$, and if $c_j(\mathbf{x}') \geq 0$, $\forall j$, then $\exists k$ s.t. $f_k(\mathbf{x}^\star) < f_k(\mathbf{x}')$ assuming minimization. In other words, each point of the Pareto set has to be better or equal to any other feasible point in at least one of the objectives. These conditions can be informally summarized as the following unnormalized distribution:

$$p(\mathcal{X}^\star|\mathbf{f}, \mathbf{c}) \propto \prod_{\mathbf{x}^\star \in \mathcal{X}^\star} \left(\left[\prod_{j=1}^{C} \Phi_j(\mathbf{x}^\star) \right] \left[\prod_{\mathbf{x}' \in \mathcal{X}} \Omega(\mathbf{x}', \mathbf{x}^\star) \right] \right) , \tag{4.8}$$

where $\Phi_j(\mathbf{x}^\star) = \Theta(c_j(\mathbf{x}^\star))$ with $\Theta(\cdot)$ the Heaviside step function (using the convention that $\Theta(0) = 1$) and the factor $\Omega(\mathbf{x}', \mathbf{x}^\star)$ is defined as:

$$\Omega(\mathbf{x}', \mathbf{x}^\star) = \left[\prod_{j=1}^{C} \Theta(c_j(\mathbf{x}')) \right] \psi(\mathbf{x}', \mathbf{x}^\star) + \left[1 - \prod_{j=1}^{C} \Theta(c_j(\mathbf{x}')) \right] \cdot 1 , \tag{4.9}$$

$$\psi(\mathbf{x}', \mathbf{x}^\star) = 1 - \prod_{k=1}^{K} \Theta(f_k(\mathbf{x}^\star) - f_k(\mathbf{x}')) . \tag{4.10}$$

Note that $\prod_{j=1}^{C} \Phi_j(\mathbf{x}^\star)$ in (4.8) guarantees that every point in the Pareto set $\mathcal{X}^\star$ belongs to the feasible space $\mathcal{F}$. Otherwise, $p(\mathcal{X}^\star|\mathbf{f}, \mathbf{c})$ is equal to zero. The factors $\Omega(\mathbf{x}', \mathbf{x}^\star)$ in (4.8) are explained as follows: The product $\prod_{j=1}^{C} \Theta(c_j(\mathbf{x}'))$ checks that the input location $\mathbf{x}'$ belongs to the feasible space $\mathcal{F}$. If the point $\mathbf{x}'$ is not feasible, we do not really care about $\mathbf{x}'$, *i.e.*, we simply multiply everything by one. Otherwise, the input location $\mathbf{x}'$ has to be dominated by the Pareto point $\mathbf{x}^\star$. That is, $\mathbf{x}^\star$ has to be better than $\mathbf{x}'$ in at least one objective. That is precisely checked by (4.10). In summary, the r.h.s. of (4.8) takes value one if $\mathcal{X}^\star$ is a valid Pareto set and zero otherwise.

We now show how to approximate the conditional predictive distribution $p(\mathbf{y}|\mathcal{D}, \mathbf{x}, \mathcal{X}^\star)$. For simplicity, we consider a noiseless case in which we observe the actual objectives and constraints: $p(\mathbf{y}|\mathbf{x}, \mathbf{f}, \mathbf{c}) = \prod_{k=1}^{K} \delta(y_k - f_k(\mathbf{x})) \prod_{j=1}^{C} \delta(y_{K+j} - c_j(\mathbf{x}))$, where $\delta(\cdot)$ is a Dirac's delta function. In the case of noisy observations, one simply has to replace the Dirac's delta function with a Gaussian with the corresponding noise variance (we assume i.i.d Gaussian noise). The unnormalized version of $p(\mathbf{y}|\mathcal{D}, \mathbf{x}, \mathcal{X}^\star)$ is:

$$
\begin{aligned}
p(\mathbf{y}|\mathcal{D}, \mathbf{x}, \mathcal{X}^\star) \propto \int p(\mathbf{y}|\mathbf{x}, \mathbf{f}, \mathbf{c}) p(\mathcal{X}^\star|\mathbf{f}, \mathbf{c}) p(\mathbf{f}|\mathcal{D}) p(\mathbf{c}|\mathcal{D}) d\mathbf{f} d\mathbf{c} \propto \\
\int \prod_{k=1}^{K} \delta(y_k - f_k(\mathbf{x})) \prod_{j=1}^{C} \delta(y_{K+j} - c_j(\mathbf{x})) \times \\
\prod_{\mathbf{x}^\star \in \mathcal{X}^\star} \prod_{j=1}^{C} \Phi_j(\mathbf{x}^\star) \times \prod_{\mathbf{x}^\star \in \mathcal{X}^\star} \left(\Omega(\mathbf{x}, \mathbf{x}^\star) \prod_{\mathbf{x}' \in \mathcal{X} \setminus \{\mathbf{x}\}} \Omega(\mathbf{x}', \mathbf{x}^\star) \right) \times \\
p(\mathbf{f}|\mathcal{D}) p(\mathbf{c}|\mathcal{D}) d\mathbf{f} d\mathbf{c} \,, \qquad (4.11)
\end{aligned}
$$

where we have separated out the factors $\Omega(\cdot, \cdot)$ that do not depend on $\mathbf{x}$, *i.e.*, the point on which the acquisition function is going to be evaluated.

In (4.11) the posterior distribution of each objective and constraint (*i.e.*, $p(\mathbf{f}|\mathcal{D})$ and $p(\mathbf{c}|\mathcal{D})$) and the delta functions are all Gaussian. The other factors are not. Furthermore, $\mathcal{X}$ can potentially be of infinite size. All this makes the evaluation of (4.11) intractable in practice. To overcome this limitation we provide an efficient approximation based on two steps. First, $\mathcal{X}$, the set of all potential input locations, is approximated as $\hat{\mathcal{X}} = \{\mathbf{x}_n\}_{n=1}^{N} \cup \mathcal{X}^\star \cup \{\mathbf{x}\}$, where $\{\mathbf{x}_n\}_{n=1}^{N}$ are the input locations where the objectives and constraints have been evaluated so far. Second, all non-Gaussian factors in (4.11), *i.e.*, $\Phi_j(\cdot)$ and $\Omega(\cdot, \cdot)$ are replaced with corresponding approximate Gaussian factors, $\tilde{\Phi}_j(\cdot)$ and $\tilde{\Omega}(\cdot, \cdot)$. This last step is carried out using the expectation propagation (EP) algorithm (Minka, 2001a). More precisely, each $\Phi_j(\cdot)$ factor is approximated by a one-dimensional un-normalized Gaussian over $c_j(\mathbf{x}^\star)$:

$$
\Phi_j(\mathbf{x}^\star) \approx \tilde{\Phi}_j(\mathbf{x}^\star) \propto \exp\{-0.5 \cdot c_j(\mathbf{x}^\star)^2 \tilde{v}_j^{\mathbf{x}^\star} + c_j(\mathbf{x}^\star) \tilde{m}_j^{\mathbf{x}^\star}\} \,, \qquad (4.12)
$$

where $\tilde{v}_j^{\mathbf{x}^\star}$ and $\tilde{m}_j^{\mathbf{x}^\star}$ are natural parameters adjusted by EP. Similarly, each $\Omega(\mathbf{x}', \mathbf{x}^\star)$ factor is approximated by a product of C one-dimensional and K two-dimensional un-normalized Gaussians:

$$
\begin{aligned}
\Omega(\mathbf{x}', \mathbf{x}^\star) \approx \tilde{\Omega}(\mathbf{x}', \mathbf{x}^\star) \propto \prod_{k=1}^{K} \exp\{-0.5 \cdot \boldsymbol{\upsilon}_k^{\mathrm{T}} \tilde{\mathbf{V}}_k^{\Omega} \boldsymbol{\upsilon}_k + (\tilde{\mathbf{m}}_k^{\Omega})^{\mathrm{T}} \boldsymbol{\upsilon}_k\} \times \\
\prod_{j=1}^{C} \exp\{-0.5 \cdot c_j(\mathbf{x}^\star)^2 \tilde{v}_j^{\Omega} + c_j(\mathbf{x}^\star) \tilde{m}_j^{\Omega}\} \,, \qquad (4.13)
\end{aligned}
$$

where $\boldsymbol{\upsilon}_k = (f_k(\mathbf{x}'), f_k(\mathbf{x}^\star))^{\mathrm{T}}$ and $\tilde{\mathbf{V}}_k^{\Omega}$, $\tilde{\mathbf{m}}_k^{\Omega}$, $\tilde{v}_j^{\Omega}$ and $\tilde{m}_j^{\Omega}$ are natural parameters adjusted by EP. Note that $\tilde{\mathbf{V}}_k^{\Omega}$ is a 2×2 matrix and $\tilde{\mathbf{m}}_k^{\Omega}$ is a two-dimensional vector.

EP refines all these approximate factors iteratively until their parameters do not change any more. This ensures that they look similar to the corresponding exact factors. The factors, that do not depend on $\mathbf{x}$ are reused each time that the acquisition function has to be computed at a new input location $\mathbf{x}$. The other factors that depend on $\mathbf{x}$ need

to be computed relatively fast to guarantee that the acquisition function is not very expensive to evaluate. Therefore, these factors are only updated once by EP in practice.

4.2.4 The PESMOC's Acquisition Function

Once EP has finished, the conditional predictive distribution $p(\mathbf{y}|\mathcal{D}, \mathbf{x}, \mathcal{X}^\star)$ is approximated by the distribution that results from replacing in (4.11) each non-Gaussian factor by the corresponding EP Gaussian approximation. Because the Gaussian distribution is closed under the product operation, the resulting distribution is Gaussian. That is:

$$p(\mathbf{y}|\mathcal{D}, \mathbf{x}, \mathcal{X}^\star) \approx \prod_{k=1}^{K} \mathcal{N}(f_k(\mathbf{x})|m_k^{\text{CPD}}, v_k^{\text{CPD}}) \prod_{j=1}^{C} \mathcal{N}(c_j(\mathbf{x})|m_j^{\text{CPD}}, s_j^{\text{CPD}}) \,, \tag{4.14}$$

where the parameters $m_k^{\text{CPD}}, v_k^{\text{CPD}}, m_j^{\text{CPD}}, s_j^{\text{CPD}}$ can be obtained from the product of the approximate factors, $p(\mathbf{f}|\mathcal{D})$ and $p(\mathbf{c}|\mathcal{D})$, and the delta functions. Then, PESMOC's acquisition function is simply given by the sum of the differences between the entropies before and after conditioning on the Pareto set. This, in combination with the expression shown in (4.7) gives:

$$\begin{aligned} \alpha(\mathbf{x}) \approx & \sum_{j=1}^{C} \log s_j^{\text{PD}}(\mathbf{x}) + \sum_{k=1}^{K} \log v_k^{\text{PD}}(\mathbf{x}) - \\ & \frac{1}{M} \sum_{m=1}^{M} \left[\sum_{j=1}^{C} \log s_j^{\text{CPD}}(\mathbf{x}|\mathcal{X}_{(m)}^\star) + \sum_{k=1}^{K} \log v_k^{\text{CPD}}(\mathbf{x}|\mathcal{X}_{(m)}^\star) \right] , \end{aligned} \tag{4.15}$$

where M is the number of Monte Carlo samples of the Pareto set $\{\mathcal{X}_{(m)}^\star\}_{m=1}^{M}$ used to approximate the expectation in the r.h.s. of (4.6); and $v_k^{\text{PD}}(\mathbf{x})$, $s_j^{PD}(\mathbf{x})$, $v_k^{\text{CPD}}(\mathbf{x}|\mathcal{X}_{(m)}^\star)$ and $s_j^{\text{CPD}}(\mathbf{x}|\mathcal{X}_{(m)}^\star)$ are the variances of the predictive distribution before and after conditioning on the Pareto set $\mathcal{X}_{(m)}^\star$. In the case of noisy observations around each objective or constraint we simply increase the predictive variances by adding the corresponding variance of the Gaussian additive noise. The next point at which to evaluate the objectives and the constraints is the one that maximizes (4.15).

We note that the acquisition function in (4.15) can be expressed as a sum across the objectives and the constraints. That is,

$$\alpha(\mathbf{x}) = \sum_{k=1}^{K} \alpha_k^{\text{obj}}(\mathbf{x}) + \sum_{j=1}^{C} \alpha_j^{\text{const}}(\mathbf{x}) \,, \tag{4.16}$$

where

$$\alpha_k^{\text{obj}}(\mathbf{x}) = \log v_k^{\text{PD}}(\mathbf{x}) - \frac{1}{M} \sum_{k=1}^{K} \log v_k^{\text{CPD}}(\mathbf{x}|\mathcal{X}_{(m)}^\star) \,, \tag{4.17}$$

$$\alpha_c^{\text{const}}(\mathbf{x}) = \log s_j^{\text{PD}}(\mathbf{x}) - \frac{1}{M} \sum_{j=1}^{C} \log s_j^{\text{CPD}}(\mathbf{x}|\mathcal{X}_{(m)}^\star) \,. \tag{4.18}$$

Intuitively each of these functions measures the reduction in the entropy of the Pareto set after an evaluation of the corresponding objective or constraint. Therefore PESMOC can be used to identify not only where to perform the next evaluation, but also which black-box function (objective or constraint) or subset of these should be evaluated next. This allows for a decoupled evaluation scenario. In the simplest case in which we consider an acquisition function per black-box function, we only have to maximize independently each of these $K + C$ acquisition functions to identify the most promising black-box function to evaluate next. We expect that this approach is more effective for reducing the entropy of the Pareto set in the feasible space, leading to better optimization results with a smaller number of black-box evaluations.

An illustrative example of the computation of PESMOC's acquisition functions is shown in Figure 4.1 for a simple one-dimensional problem with two objectives and one constraint. The first column displays the data collected so far. Each black-box function is modelled using a GP, whose predictive distribution is shown in terms of the mean prediction and one standard deviation. The second column of the figure displays a function sampled from the predictive distribution of each black-box function. These samples are then optimized to obtain a sample of the Pareto set $\mathcal{X}^{\star}_{(m)}$, which is displayed in the figure using blue crosses. These points dominate all other points for which the corresponding values of the constraint are positive. The third column of this figure shows the predictive distribution of each black-box function conditioned to $\mathcal{X}^{\star}_{(m)}$ being the solution to the optimization problem. This predictive distribution is approximated using EP. Note that the predictive variance is reduced significantly in some locations of the input space. These are the locations that are expected to be most informative about the actual Pareto set $\mathcal{X}^{\star}$. Finally, the last column shows the corresponding acquisition function for each black-box function, alongside with the corresponding maximizer. The acquisition is simply given by the difference in the logarithm of the predictive variance before and after the conditioning. Note that those regions of the input space in which the acquisition is high correspond to those regions in which the predictive variance is significantly reduced.

The acquisition function obtained for a coupled evaluation setting is shown also in Figure 4.2 for reference. A comparison between Figure 4.1 and Figure 4.2 shows the potential benefits of a decoupled evaluation approach. The acquisition function obtained in the coupled scenario is simply the sum of all the previous acquisition functions. Furthermore, note that the maximizer of this function need not be equal to the maximizers of any of the individual acquisition functions. Therefore, the sum of the individual maximums of each of the three different acquisition functions displayed in Figure 4.1 is expected to be larger than the maximum of the acquisition function displayed in Figure 4.2. A decoupled evaluation setting is hence expected to be more useful for decreasing the entropy of the Pareto set in the feasible space, and to give better results with a smaller number of black-box evaluations.

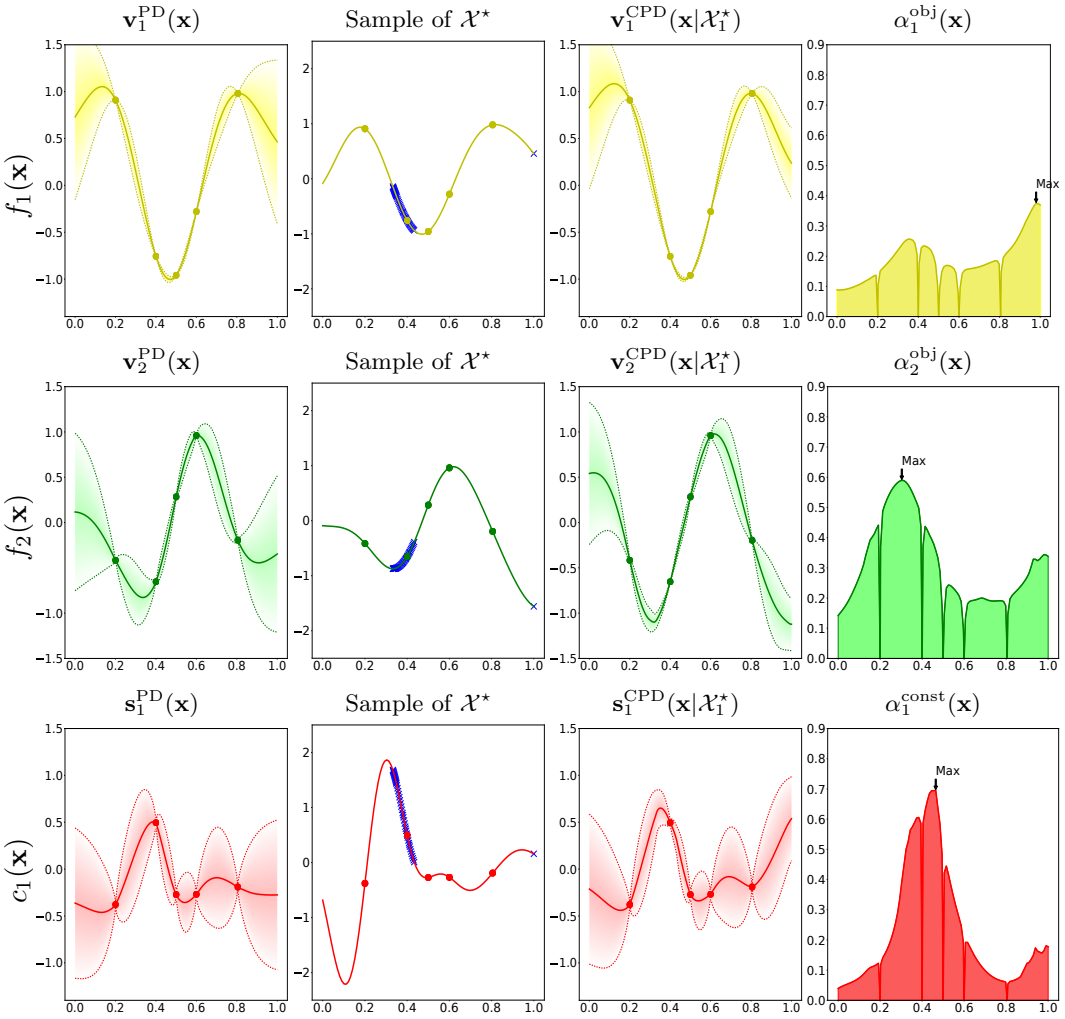

FIGURE 4.1: Different steps needed to compute PESMOC's acquisition function in a decoupled evaluation scenario. **(first column)** Predictive distribution for each black-box function conditioned on the observed data given by a GP. **(second column)** Sample from the posterior distribution of each GP alongside with the corresponding Pareto set $\mathcal{X}^\star_{(m)}$ in the feasible space displayed using blue crosses. **(third column)** Predictive distribution of each black-box function conditioned to the sampled Pareto set $\mathcal{X}^\star_{(m)}$ being the solution to the optimization problem. **(fourth column)** Acquisition function obtained by the difference in the entropy of the predictive distribution before and after the conditioning.

4.2.5 Computational Cost of PESMOC's Acquisition Function

The cost of running EP and evaluating the acquisition function is $\mathcal{O}((K+C)q^3)$, where $q = N + |\mathcal{X}^\star_{(m)}|$, and N is the number of observations collected so far, K is the number of objectives and C is the number of constraints. In practice EP is run only once per sample of the Pareto set $\mathcal{X}^\star_{(m)}$ because it is possible to re-use the factors that are independent of the candidate location $\mathbf{x}$ at which the acquisition function has to be evaluated. Thus, the complexity of computing the predictive variance is $\mathcal{O}((K+C)|\mathcal{X}^\star_{(m)}|^3)$. In practice, we set the size of the Pareto set sample $\mathcal{X}^\star_{(m)}$ to be equal to 50, making q just a few

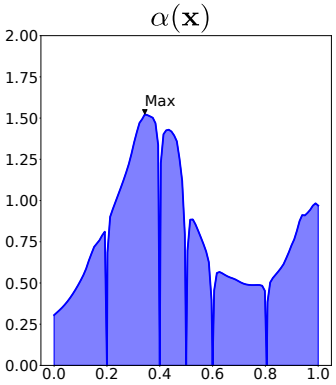

FIGURE 4.2: Acquisition function of PESMOC for the coupled setting. In this case $\alpha(\cdot)$ is simply the sum of the acquisition functions of the three black boxes shown in Figure 4.1, in which the decoupled approach was displayed.

hundreds at most. We provide more details in the supplementary material about how to the conditional predictive distribution $p(\mathbf{y}|\mathcal{D}, \mathbf{x}, \mathcal{X}^\star)$ is obtained after running EP.

4.3 Related Work

In this section we review important related work to multi-objective optimization under the presence of several constraints, when both the objectives and the constraints can be regarded as black-boxes. We also describe related methods for Bayesian optimization and previous approaches that also allow for decoupled evaluations.

4.3.1 Evolutionary Strategies and Meta-heuristics

The problem of constrained multiobjective optimization where the analytical form of the objectives or the constraints is unknown has been already tackled in the literature. In order to solve these problems one can employ evolutionary strategies such as the ones described in (Cai and Wang, 2006; Fonseca and Fleming, 1998). Similarly, other techniques adapted to this scenario include particle swarm optimization (Coello et al., 2004) or ant colony optimization (Alaya et al., 2007). These techniques perform a search in the target space guided by some criterion that tries to find the best trade-off between exploration of good solutions far away from the regions already explored, and exploitation of the best known solutions. The problem of these techniques, also known as meta-heuristics, is that they usually require a large number of evaluations in order to achieve good results. This is un-affordable in our scenario in which the black-box functions are expected to be very expensive to evaluate. BO methods, which exploit the information provided by the probabilistic models to make intelligent decisions about where to evaluate next these functions, will perform much better in a scenario that includes a limited evaluation budget (a few hundred evaluations at most). Empirical evidence supporting this is found, for example, in (Bischl et al., 2017b; Hernández-Lobato et al., 2016).

4.3.2 Related Bayesian Optimization Methods

In the literature most BO methods have traditionally focused on the un-constrained single objective scenario. A comprehensive summary of different works targeting this type of problems can be found in (Brochu et al., 2010; Shahriari et al., 2015). The first BO methods based on entropy search were proposed to address these simple optimization problems (Hennig and Schuler, 2012; Villemonteix et al., 2009). The corresponding re-formulation based on predictive entropy search in such a setting is described in (Hernández-Lobato et al., 2014). This reformulation provides an approximation of the acquisition function of entropy search that is more accurate, as the required computations are simplified significantly, and that also leads to better optimization results in practice. In any case, all these works can only optimize a single objective under no constraints.

The multi-objective case in which several objectives need to be simultaneously optimized in an un-constrained scenario has also received the attention of the BO community. In particular, several BO methods have been proposed to address these problems, including ParEGO, SMS-EGO, expected hyper-volume improvement (EHI) and sequential uncertainty reduction (SUR) (Emmerich and Klinkenberg, 2008; Knowles, 2006; Picheny, 2015; Ponweiser et al., 2008). A multi-objective BO method based on using entropy search and the corresponding re-formulation based on predictive entropy search is described in (Hernández-Lobato et al., 2016). However, such a method cannot consider constraints. The work described here is a natural extension that allows to incorporate several constraints to the multi-objective problem. Importantly, this extension is not trivial since it involves the use of more complicated factors in the computation of the conditional predictive distribution. Furthermore, the EP update operations required to compute the approximation of the acquisition function are also more arduous.

The problem of optimizing a single objective under several constraints has also been considered by the BO community. The methods proposed with this goal include variants of the expected improvement (EI) acquisition function in which one simply chooses the point that is expected to improve the most the best observed result so far. For example, the expected improvement with constraints (EIC) (Gardner et al., 2014; Gelbart et al., 2014; Parr, 2013; Schonlau et al., 1998; Snoek, 2013). A method that is able to tackle this type of problems and that is based on entropy search and the corresponding reformulation using predictive entropy search has also been proposed in (Hernández-Lobato et al., 2016; Hernández-Lobato et al., 2015). Such a method, however, cannot optimize several objectives at the same time, unlike PESMOC, the method described in this chapter. Optimizing several objectives at the same time is a significantly more complicated problem. In particular, when the objectives are conflictive, the solution to the optimization problem is a set of points, the Pareto set in the feasible space, of potentially infinite size.

4.3.3 Bayesian Multi-Objective Optimization

A BO method proposed in the literature to optimize several objectives under several constraints is Bayesian Multi-objective optimization (BMOO) (Féliot et al., 2017). Such a method is based on the expected hyper-volume improvement acquisition function (EHI) (Emmerich and Klinkenberg, 2008), in which the expected increase in the hyper-volume is computed after performing an evaluation of the black-box functions at a particular input location. The hyper-volume is simply the volume of points in functional space above the Pareto front (*i.e.*, the function values associated to the Pareto set), which is maximized by the actual Pareto set. It is hence a natural measure of quality or utility of the current

solution of the multi-objective problem. When several constraints are introduced in the problem, this criterion boils down to the product of a modified EHI criterion (where only feasible points are considered) and the probability of feasibility, as indicated by the probabilistic models. Importantly, the utility function of this acquisition function (the acquisition is simply the expectation of the utility function under the predictive distribution of the probabilistic models) is constant (equal to zero) as long as no feasible point has been observed. Therefore, it is not an appropriate utility function for heavily constrained problems, where finding feasible points is sometimes the main difficulty. As indicated by Féliot et al. (2017), not all unfeasible points are equivalent. A point that does not satisfy a constraint by a small amount has probably more value than one that does not satisfy the constraint by a large amount, and should therefore contribute more to the utility.

With the goal of overcoming the limitations described before, Féliot et al. (2017) propose an extended domination rule to handle objectives and constraints in a unified way. This domination rule considers both objectives $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$ and constraints $\mathbf{c}(\mathbf{x}) = (c_1(\mathbf{x}), \ldots, c_m(\mathbf{x}))$. For this, the space of potential objective values $\mathbf{f}(\mathbf{x}) \in \mathcal{Y}_o \subset \mathbb{R}^K$ and the space of potential constraint values $\mathbf{c}(\mathbf{x}) \in \mathcal{Y}_c \subset \mathbb{R}^C$ are joined, giving as a result the extended space $\mathcal{Y}_o \times \mathcal{Y}_c$. Define $\mathbf{y}^o_{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ and $\mathbf{y}^c_{\mathbf{x}} = \mathbf{c}(\mathbf{x})$. That is $\mathbf{y}^o_{\mathbf{x}}$ is a vector with the objective values associated to $\mathbf{x}$ and $\mathbf{y}^c_{\mathbf{x}}$ is a vector with the constraint values. The extended domination rule states that a point $\mathbf{x}$ dominates another one $\mathbf{x}'$, if $\Psi(\mathbf{y}^o_{\mathbf{x}}, \mathbf{y}^c_{\mathbf{x}})$ dominates $\Psi(\mathbf{y}^o_{\mathbf{x}'}, \mathbf{y}^c_{\mathbf{x}'})$, using the classical Pareto domination rule. That is, $\Psi(\mathbf{y}^o_{\mathbf{x}}, \mathbf{y}^c_{\mathbf{x}}) \prec \Psi(\mathbf{y}^o_{\mathbf{x}'}, \mathbf{y}^c_{\mathbf{x}'})$ i.f.f $\Psi(\mathbf{y}^o_{\mathbf{x}}, \mathbf{y}^c_{\mathbf{x}})$ is better than $\Psi(\mathbf{y}^o_{\mathbf{x}'}, \mathbf{y}^c_{\mathbf{x}'})$ in at least one component. Let $\overline{\mathbb{R}}$ be the extended real line. The transformation $\Psi(\cdot, \cdot) : \mathcal{Y}_o \times \mathcal{Y}_c \to \overline{\mathbb{R}}^K \times \mathbb{R}^C$ is defined as:

$$\Psi(\mathbf{y}^o_{\mathbf{x}}, \mathbf{y}^c_{\mathbf{x}}) = \begin{cases} (\mathbf{y}^o_{\mathbf{x}}, \mathbf{0}) & \text{if } \mathbf{y}^c_{\mathbf{x}} \geq 0, \\ (+\infty, \min(\mathbf{y}^c_{\mathbf{x}}, \mathbf{0})) & \text{otherwise.} \end{cases} \tag{4.19}$$

That is, if the point is feasible (*i.e.*, all the constraints are positive or equal to zero), only the objective values are considered. Conversely, if the point is infeasible, the constraint values will play a role. More precisely, under this rule a solution that is infeasible but close to being feasible will dominate other infeasible solutions that are further away from being feasible. As described by Féliot et al. (2017), the previous rule has these properties:

1. For unconstrained problems the extended domination rule boils down to the classical Pareto domination rule.

2. Feasible solutions (corresponding to $\mathbf{y}^c_{\mathbf{x}} \geq \mathbf{0}$) are compared using the Pareto domination rule applied in the objective space.

3. Non-feasible solutions (corresponding to $\mathbf{y}^c_{\mathbf{x}} \ngeq \mathbf{0}$) are compared using the Pareto domination rule applied to the vector of constraint violations.

4. Feasible solutions always dominate non-feasible solutions.

The extended domination rule presented above makes it possible to define a notion of expected hyper-volume improvement in the extended space. This is the acquisition function considered by Féliot et al. (2017). A problem is, however, that evaluating this quantity can be expensive if the number of objectives and constraints is large. To overcome this limitation an efficient approximate computation method is proposed by those authors. This approximation is obtained by noticing that the proposed acquisition function at a candidate point $\mathbf{x}$ is given by the expected value of the probability that $\Psi(\mathbf{y}^o_{\mathbf{x}}, \mathbf{y}^c_{\mathbf{x}})$

dominates a point $\mathbf{y}$ belonging to the set of non-dominated points in the extended output space, when $\mathbf{y}$ is chosen uniformly at random from that set. In particular,

$$\begin{aligned}\alpha_{\text{BMOO}}(\mathbf{x}) &= \mathbb{E}_{\mathbf{y}^o_{\mathbf{x}},\mathbf{y}^c_{\mathbf{x}}}\left[\int_{\mathcal{G}_N} I(\Psi(\mathbf{y}^o_{\mathbf{x}},\mathbf{y}^c_{\mathbf{x}}) \prec \mathbf{y}) d\mathbf{y}\right] \\ &= \int_{\mathcal{G}_N} p(\Psi(\mathbf{y}^o_{\mathbf{x}},\mathbf{y}^c_{\mathbf{x}}) \prec \mathbf{y}) d\mathbf{y} \\ &\approx \frac{1}{M}\sum_{m=1}^{M} p(\Psi(\mathbf{y}^o_{\mathbf{x}},\mathbf{y}^c_{\mathbf{x}}) \prec \mathbf{y}_m)\,, \end{aligned} \tag{4.20}$$

where $\mathcal{G}_N$ is the set of non-dominated points (up to the current iteration N) in the extended output space; $I(\cdot)$ is an indicator function; the expectation is given by the predictive distribution of the probabilistic models fitting each objective and constraint; $p(\Psi(\mathbf{y}^o_{\mathbf{x}},\mathbf{y}^c_{\mathbf{x}}) \prec \mathbf{y})$ is the probability that $\Psi(\mathbf{y}^o_{\mathbf{x}},\mathbf{y}^c_{\mathbf{x}})$ dominates $\mathbf{y}$ and M is the number of samples for $\mathbf{y} \in \mathcal{G}_n$ used in the approximation. Importantly, there is a closed form expression for $p(\Psi(\mathbf{y}^o_{\mathbf{x}},\mathbf{y}^c_{\mathbf{x}}) \prec \mathbf{y})$ when the probabilistic models are Gaussian processes. The only problem is hence how to generate uniform variables over the set $\mathcal{G}_N$.

To generate the samples required in (4.20) Féliot et al. (2017) propose a Monte Carlo method based on the Metropolis-Hastings algorithm targeting the uniform distribution in $\mathcal{G}_N$. At each iteration of this algorithm the current samples (particles) are slightly perturbed. This step is only accepted if the new particle falls in $\mathcal{G}_N$. Of course, when a new observation is obtained, improving the current solution, the set $\mathcal{G}_{N+1}$ has a smaller size than $\mathcal{G}_N$. Therefore, some particles may have to be removed. Féliot et al. (2017) describe an intelligent method to avoid the elimination of a large number of particles at each iteration, which will reduce the quality of the generated samples and the approximation.

In our experiments we have compared the proposed approach, PESMOC, with the BMOO method just described. We have observed that BMOO suffers from the limitations of traditional methods based on the expected improvement (EI). In particular, it often is too greedy and tends to explore the limits of the input space too much. In high dimensions this can be a problem since there are a lot of these corners. An extra difficulty of BMOO is that, in practice, one has to bound the space of potential output values for the objectives and the constraints, and this information may not be available before hand.

4.3.4 Existing Methods for Decoupled Evaluations

Decoupled evaluations in a BO setting were first considered by Gelbart et al. (2014) for a single objective and several constraints. In that work it is shown that the standard acquisition function known as expected improvement (EI) leads to a pathology that prevents decoupled evaluations. The reason for that is that no-improvement over the current best solution (this is the utility function of EI) can occur if we observe only the objective or the constraints, separately. More precisely, two conditions are required to produce positive values of the utility: (i) the evaluation for the objective must achieve a lower value than the best observed feasible solution so far and, (ii), the evaluations for the constraints must produce non-negative values. These two conditions cannot be simultaneously satisfied by a single observation (objective or constraint). Therefore, standard EI cannot be used for decoupled evaluations. The problem described is solved by using a two stage process in which standard EI is used to pick-up a candidate point

$\mathbf{x}_{N+1}$, and then, entropy search is used to choose the black-box function to evaluate next (Villemonteix et al., 2009). This approach is sub-optimal and a joint selection of $\mathbf{x}_{N+1}$ and the black-box is expected to perform better.

The limitations of the previous method are circumvented in (Hernández-Lobato et al., 2016). In particular, PESC, the strategy described in that chapter for single-objective constrained Bayesian optimization, allows to perform decoupled evaluations that can simultaneously choose $\mathbf{x}_{N+1}$ and the black-box function to evaluate next. This strategy is also based on predictive entropy search and expectation propagation. A comprehensive analysis of the decoupled evaluation setting in this type of optimization problems is carried out in that work. Importantly, two different decoupled configurations are evaluated: (i) competitive decoupling, in which the black-boxes compete for a single resource available; and (ii) non-competitive decoupling, in which the black-boxes can be evaluated in parallel at different input locations. Only competitive decoupling is found to perform significantly better than a coupled evaluation setting.

A decoupled evaluation method for un-constrained multi-objective Bayesian optimization is described in (Hernández-Lobato et al., 2016). PESMO, the technique described by those authors, also uses predictive entropy search and expectation propagation to choose which black-box function and which input location $\mathbf{x}_{N+1}$ to evaluate next. These authors only consider competitive decoupled evaluations. The results obtained show that such a setting can significantly outperform coupled evaluations in the multi-objective case.

The method we propose here, PESMOC, can be seen a natural extension of the two works described above, PESC, and PESMO. PESMOC also allows for decoupled evaluations and combines the possibility of considering several objectives and several constraints at the same time. Our results also indicate that a decoupled evaluation setting may have important benefits in the constrained multi-objective case. The method we compare with, BMOO, is based on a generalization of EI for constrained multi-objective problems. Therefore, this method suffers from the same limitations as standard EI for the single-objective constrained setting and cannot be used to perform decoupled evaluations.

PESMOC is significantly different from PESC. In particular, PESC can only provide solutions to single-objective constrained optimization probems. PESMOC, on the other hand, can be used to find the solution of multi-objetive optimization problems under several constraints. When the objectives are conflictive, the solution is a set of points, the Pareto set in the feasible space, of potentially infinite size. This makes multi-objective problems significantly more challenging.

PESMOC also differs from PESMO. In PESMO there are no constraints in the optimizaton problem. Incorporating constraints in the multi-objective problem is challenging and requires to add extra factors to compute the conditional predictive distribution described in (4.11). These extra factors have to be approximated by expectation propagation (EP), which results in different EP updates from those of PESMO. Furthermore, when computing the acquisition funciton one has to take into account the predictive variances of the latent functions corresponding to the constraints. In PESMO there are no constraints, so they can be ignored. Importantly, when sampling $\mathcal{X}^\star$ in PESMOC one also needs to consdier the feasible space $\mathcal{F}$ by sampling also the constraints, which must be taken into account when solving the corresponding optimization problem. This makes the process of sampling $\mathcal{X}^\star$ more complicated in PESMOC. PESMO does not have to consider the possibility of having constraints. Finally, when doing a recommendation, PESMOC has to take into account that the provided solution must be feasible with high probability, as indicated in Section 4.4. PESMO does not have to worry about

this. Summing up, PESMOC can be used to solve a collection of problems that PESMO cannot address.

4.4 Experiments

We carry out several experiments to evaluate the performance of PESMOC, the proposed method for constrained multi-objective Bayesian optimization. In these experiments we compare coupled evaluations and competitive decoupled evaluations. In the second case, we not only choose which is the next input location but also which black-box function should be evaluated next. We compare the results of PESMOC with those of the BMOO method of Féliot et al. (2017) and a base-line strategy that explores the input space uniformly at random (Random). Note that this strategy is expected to perform worse than either PESMOC or BMOO because it does not use the probabilistic models to identify the next point on which to do the next evaluation. All these methods have been implemented in the software for Bayesian optimization Spearmint (`https://github.com/EduardoGarrido90/Spearmint`). In each experiment carried out in this section we report average results and the corresponding standard deviations. The results reported are averages over 100 repetitions of the corresponding experiment. Means and standard deviations are estimated using 200 bootstrap samples. In the synthetic problems we consider two scenarios. Namely, noiseless and noisy observations, and report results for both of them.

In each method, *i.e.*, PESMOC and BMOO, a Matérn covariance function is used for the GPs that model the objectives and the constraints. The hyper-parameters of each GP (length-scales, level of noise and amplitude) are approximately sampled from their posterior distribution using slice sampling as in (Snoek et al., 2012). We generate 10 samples for each hyper-parameter, and the acquisition function of each method is averaged over these samples. In PESMOC the parameter M which specifies the number of Monte Carlo samples of $\mathcal{X}^\star$ in (4.15) is set to 10. This is the value used by previous approaches based on predictive entropy search for single objective and un-constrained multi-objective optimization (Hernández-Lobato et al., 2016; Hernández-Lobato et al., 2016). Furthermore, the supplementary material includes some experiments showing that setting M to this value gives a good trade-off between performance and computational cost. For each method, at each iteration of the optimization process, we output a recommendation obtained by optimizing the GPs mean functions. For this, we use a uniform grid of $1000 \times d$ points, where d is the dimensionality of the problem. We also approximate the Pareto set with 50 points.

To guarantee that only points that are feasible with high probability are recommended, we consider that a constraint $c_j(\cdot)$ is satisfied at an input location $\mathbf{x}$ if the probability that the constraint is larger than zero is above $1-\delta$ where δ is 0.05. That is, $p(c_j(\mathbf{x} \geq 0) \geq 1-\delta$. When no feasible solution is found, we simply return the points that are most likely to be feasible by iteratively increasing δ in 0.05 units. This is the approach followed by Gelbart et al. (2014) and Hernández-Lobato et al. (2016) for single-objective constrained optimization, and we have observed that it provides good empirical results in our experiments. Under the assumption that the constraints have generated from a GP prior, this approach will guarantee that the provided solutions are feasible with high probability. Note that this approach to provide recommendations is not specific of PESMOC. It is shared by all the methods considered for constrained multi-objective optimization. Namely, PESMOC, BMOO and the random search strategy.

The acquisition function of each method is maximized using L-BFGS (a grid of size $1,000 \times d$, where d is the input dimension, is used to find a good starting point). The gradients of the acquisition function are approximated by differences. In BMOO we set the number of samples used to approximate the evaluation of the acquisition function to $1,000$. These samples are perturbed at each iteration as described in (Féliot et al., 2017).

The experiments contained in this section are organized as follows: A first set of experiments evaluate the quality of PESMOC's approximation to target acquisition function described in (4.6). Then, we compare the performance of PESMOC and BMOO on synthetic experiments where the objectives and constraints are sampled from a GP prior. This comparison is then carried out using 7 well-known benchmark problems for multi-objective optimization with constraints. In this case, the objectives and constraints have not been sampled from GP prior and model bias can be important. Finally, we consider two real optimization problems: finding an optimal ensemble of decision trees on the dataset German IDA and finding an optimal deep neural network for the MNIST dataset.

4.4.1 Quality of the Approximation to the Acquisition Function

As described previously, the acquisition function of the proposed method, PESMOC, is intractable and needs to be approximated. The exact evaluation requires computing an expectation that has no closed form solution and computing the conditional predictive distribution of the probabilistic models given some Pareto set $\mathcal{X}^\star$. In Section **??** we propose to approximate these quantities using Monte Carlo samples and expectation propagation, respectively. In this section we check the accuracy of this approximation to see if it resembles the actual acquisition function. For this, we consider a simple one dimensional problem with two objectives and one constraint generated from a GP prior. In this simple setting, it is possible to compute a more accurate estimate of the acquisition function using a more expensive sampling technique, combined with a non-parametric estimator of entropy (Singh et al., 2003). More precisely, we discretize the input space and generate a sample of the Pareto set $\mathcal{X}^\star$ by optimizing a sample of the black-box functions. This sample is generated as in the PESMOC approximation. We then generate samples of the black-box functions and keep only those that are compatible with $\mathcal{X}^\star$ being the solution to the optimization problem. This process is repeated $10,000$ times. Then, a non-parametric method is used to estimate the entropy of the predictive distribution at each region of the input space before and after the conditioning. The difference in the entropy at each input location gives a more accurate estimate of the acquisition function of PESMOC. Of course, this approach is too expensive to be used in practice for solving optimization problems.

We consider first a coupled evaluation setting. Figure 4.3 (top) shows the posterior distribution (mean and one standard deviation) of the three black-box functions at a particular step of the optimization process. The bottom of this figure shows a comparison between the two estimates of the exact acquisition function. The one described above (exact) and the one suggested as an approximation. We observe that both estimates of the acquisition function take higher values in regions with high uncertainty and promising predictions. Similarly, both estimates take lower values in regions with low uncertainty. Importantly, both acquisition functions are pretty similar in the sense that they take high and low values in the same regions of the input space. Therefore, both acquisition functions are extremely correlated. This empirical result supports that the approximation proposed in this chapter is an accurate estimate of the actual acquisition function.

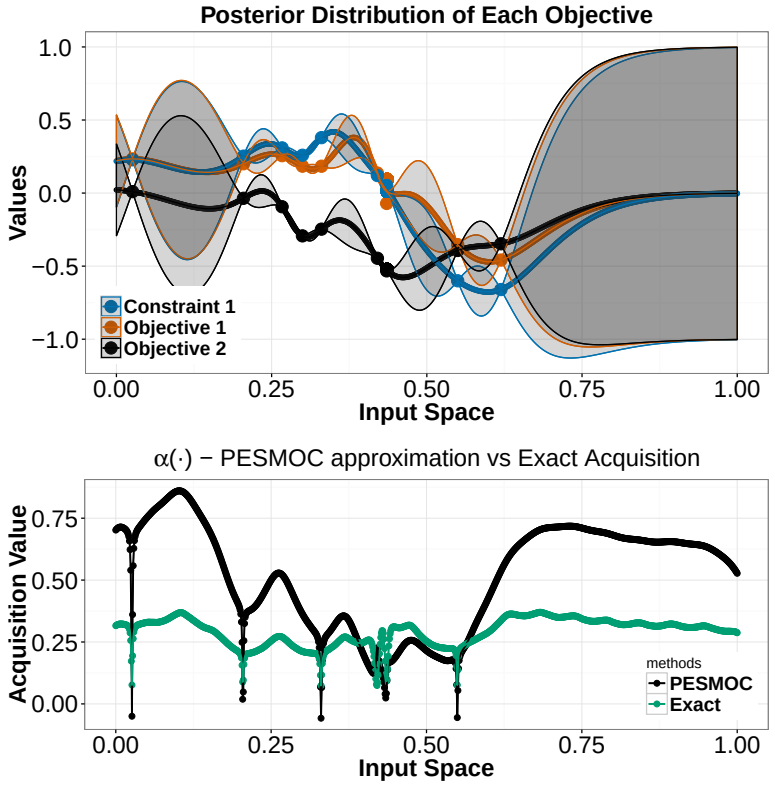

FIGURE 4.3: (top) Posterior distribution of each black-box function (mean and standard deviation). (bottom) Acquisition function estimated by PESMOC and by a more expensive but accurate Monte Carlo method combined with a non-parametric estimator of the entropy (exact).

We repeat these experiments in a decoupled scenario in which the different black-boxes need not be evaluated at the same input location. The results are displayed in Figure 4.4. In this case, we show the estimates of the acquisition function corresponding to each black-box function. Therefore, there are three different acquisition functions displayed. The plots show again that the PESMOC's approximation is accurate w.r.t the exact acquisition function, as estimated by the more expensive process described above. Again, each pair of estimates of the acquisition function for each black-box are heavily correlated, suggesting similar maximizers and often similar acquisition values. We believe that this results provides empirical evidence of the quality of the acquisition approximation carried out in the proposed method, PESMOC. The accuracy of this approximation is also validated by the good results obtained in the rest of the experiments described in this chapter.

4.4.2 Synthetic Experiments

We compare the performance of PESMOC and BMOO with that of a random search strategy when the objectives and constraints are sampled from a GP prior. For this, we generate 100 optimization problems involving 2 objectives and 2 constraints in a 4-dimensional input space. This experiment is repeated to consider a more complicated setting. In this case, we generate 100 optimization problems involving 4 objectives and 2 constraints in a 6-dimensional input space. Each strategy (PESMOC, PESMOC

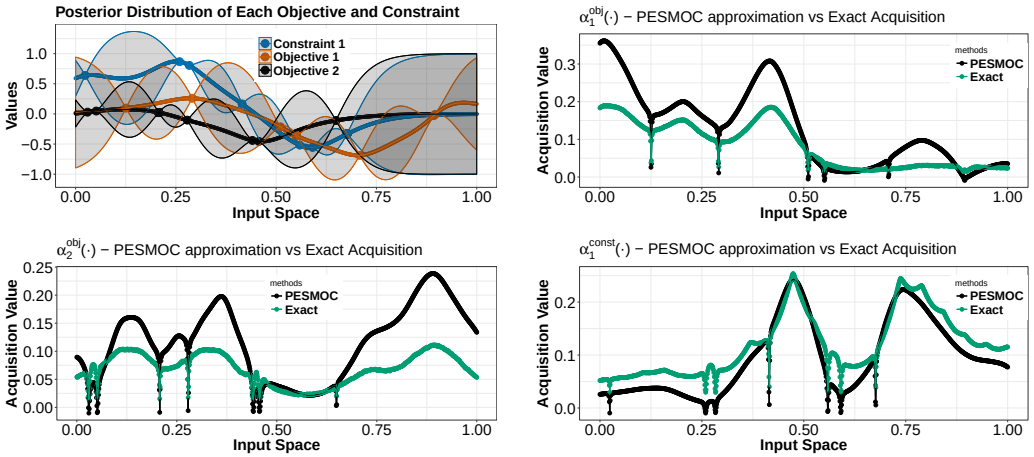

FIGURE 4.4: (top left) Posteriors distribution of each black-box function (mean and standard deviation). (top right to bottom right) Acquisition function for each black-box estimated by PESMOC and by a more expensive but accurate Monte Carlo method combined with a non-parametric estimator of the entropy (exact).

decoupled, BMOO and Random) is run on each problem until 100 evaluations of each black-box are made. We report results for a noiseless and noisy evaluation scenario, in which we observe the evaluations are contaminated with additive Gaussian noise with standard deviation equal to 0.1. After each iteration of the optimization process, each strategy outputs a recommendation in the form of a Pareto set obtained by optimizing the posterior means of the GPs, as indicated at the beginning of this section. The performance criterion used is the hyper-volume of the corresponding solution. Recall that the hyper-volume is the volume of points in functional space above the optimal points contained in the recommendation. This quantity is maximized by the actual Pareto set (Zitzler and Thiele, 1999). In the case that the recommendation produced contains an infeasible point, we simply set the hyper-volume of the recommendation equal to zero. For each method evaluated we report the logarithm of the relative difference between the hyper-volume of the actual Pareto set and the hyper-volume of the recommendation.

Figure 4.5 shows the average results obtained for each method and the corresponding error bars. We observe that the PESMOC approaches outperform both BMOO and the random search approach in the two settings considered. In particular, PESMOC is able to find better solutions to the optimization problems considered, which are more accurate than those obtained by the other methods. These solutions have a hyper-volume that is closer to the hyper-volume of the actual Pareto set. The random search method is also outperformed by BMOO in the two settings considered. However, BMOO gives worse results in the noisy scenario, as it tends to provide results that are closer to this method. Importantly, the decoupled version of PESMOC is similar or even better than the corresponding coupled counterpart. When the input dimension d grows the improvements become evident. These results confirm the benefits of a decoupled evaluation setting. In particular, the decoupled version of PESMOC is the best overall method, significantly outperforming all other approach in the 6-dimensional setting.

We also compare here the average time used by each strategy to choose the next evaluation. For this, we consider the synthetic experiments that involves 2 objectives and 2 constraints, in a 4-dimensional input space, with noisy evaluations, and the experiment

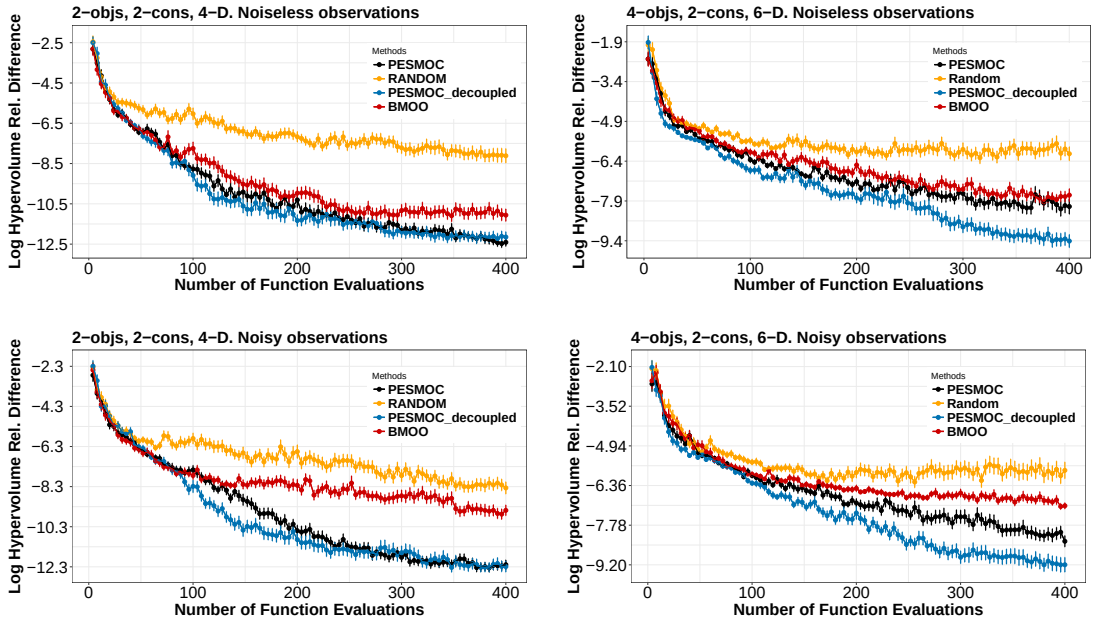

FIGURE 4.5: Logarithm of the relative difference between the hyper-volume of the recommendation obtained by each method and the hyper-volume of the actual solution. We report results after each evaluation of the black-box functions. (left column) Two objectives and two constraints. Input space of four dimensions ($d = 4$). (right column) Four objectives and two constraints. Input space of six dimensions ($d = 6$). (top row) Noiseless evaluation scenario. (bottom row) Noisy evaluation scenario. Best seen in color.

that involves 4 objectives, 2 constraints and a 6-dimensional input space, not considering noise. For each method and each iteration, we measure the average time spent in the computation and maximization of the acquisition function. In the case of PESMOC this time includes the time required to run EP until convergence, and the time required to optimize the acquisition function. In the case of BMOO this time includes the time required to generate the Monte Carlo samples used in (4.20) to approximate the acquisition function, and the time required for its optimization. The results obtained are shown in Table 4.1. We do not include in this table the random search strategy, since the time it requires to choose the next evaluation is negligible.

TABLE 4.1: Average time in seconds spent on each iteration by each method. First row corresponds to the 4-dimensional input space experiment and the second row corresponds to the 6-dimensional input space experiment.

Experiment	**PESMOC coupled**	**PESMOC decoupled**	**BMOO**
First	41.40 ± 1.48	49.63 ± 1.02	67.41 ± 4.29
Second	112.94 ± 3.25	264.60 ± 3.36	307.90 ± 34.00

Table 4.1 shows that the fastest strategy is PESMOC, followed by the decoupled version of PESMOC and BMOO. BMOO is significantly slower than PESMOC, due to the need of running the Metropolis-Hastings algorithm. By constrast, in PESMOC, EP converges in just a few iterations. Note that the decoupled version of PESMOC is slightly slower than the coupled counterpart per iteration. The reason is that it requires the optimization of one acquisition function per each black-box function, to determine the next evaluation, instead of just one as in the PESMOC case. Note that this only

represents a small fraction of the total time per iteration of PESMOC in the decoupled setting (the time of fitting the GPs and running the EP algorithm to approximate the factors that do not depend on the candidate point $\mathbf{x}$ is similar for the coupled and the decoupled setting). Importantly, however, the decoupled version will need as many more iterations as black boxes are present in the problem. For example, in the first problem, which has 4 black-box functions, to perform 400 evaluations of the black-boxes, the coupled version of PESMOC will require 100 iterations, while the decoupled version will require 400. The results shown in the table are expected to generalize to other problems involving a different number of black-boxes or input dimensions.

We also illustrate here the shape of the acquisition function of PESMOC on a toy 2-dimensional optimization problem with input domain $\mathcal{X}$ given by the box $[-10, 10] \times [-10, 10]$:

$$\min_{\mathbf{x} \in \mathcal{X}} \quad f_1(x, y) = xy, \quad f_2(x, y) = -yx \quad \text{s.t.} \quad x \geq 0, y \geq 0\,.$$

In this experiment the feasible space $\mathcal{F}$ is given by the box $[0, 10] \times [0, 10]$. Figure 4.6 shows the location of the first 20 evaluations made by each method (blue crosses) and the level curves of the acquisition function of PESMOC and BMOO. We observe that PESMOC and BMOO quickly identify the feasible space $\mathcal{F}$, and focus on evaluating the black-box functions in that region. By contrast, the random search strategy explores the space more uniformly and evaluates the black-boxes more frequently in regions of the input space that are infeasible. We observe that the acquisition functions of PESMOC and BMOO take high values inside $\mathcal{F}$ and low values outside $\mathcal{F}$.

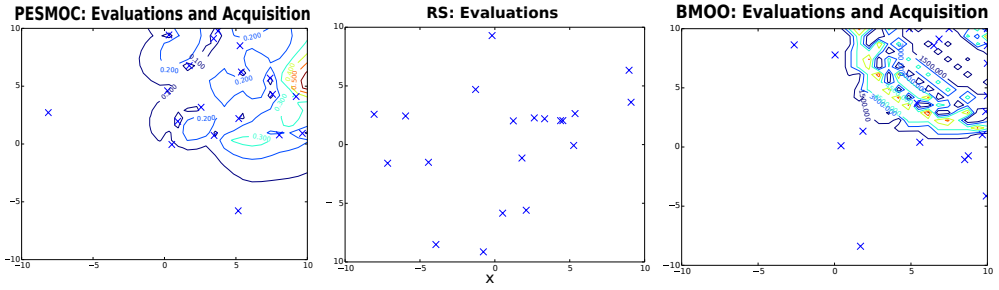

FIGURE 4.6: Location in input space (denoted with a blue cross) of each of the evaluations made by PESMOC (left), random search (middle) and BMOO (right). In the case of PESMOC and BMOO, we also plot the level curves of the acquisition function. The feasible region is the the box $[0, 10] \times [0, 10]$. Best seen in color.

4.4.3 Benchmark Experiments

In the previous experiments the black-boxes are sampled from a GP prior, which is the underlying model assumed by the different BO methods compared. This hypothesis need not be satisfied in practice. Therefore, model misspecification may have an impact in the performance of BO methods. In this section we carry out extra experiments with the goal of comparing the different methods under such a scenario. For this, we consider 7 classical benchmark problems used to assess multi-objective optimization methods with constraints (Chafekar et al., 2003; Deb et al., 2002). A summary of these problems is displayed in Table 4.2 and 4.3. All problems contain several input variables, multiple objectives and several constraints. Importantly, in these experiments we transform each constraint $c_j(\mathbf{x})$ so that the corresponding optimization problem can be expressed as

in (4.1). Furthermore, we also consider a noiseless and a noisy setting, in which the evaluations of the black-boxes are contaminated with additive Gaussian noise. The variance of the noise is set to 1% of the range of potential values of the corresponding black-box. This range of values is found by evaluating each black-box function on a grid. These experiments are repeated 100 times for each method and each dataset and we report average results. The metric used to assess the performance of each method is the same as the one employed in the previous section.

TABLE 4.2: Summary of BNH, SRN, TNK and OSY problems used in the benchmark experiments.

Benchmark Experiments		
Problem Name	Input Space	Objectives $f_k(\mathbf{x})$ and Constraints $c_j(\mathbf{x})$
BNH	$x_1 \in [0, 5]$ $x_2 \in [0, 3]$	$f_1(\mathbf{x}) = 4x_1^2 + 4x_2^2$ $f_2(\mathbf{x}) = (x_1 - 5)^2 + (x_2 - 5)^2$ $c_1(\mathbf{x}) \equiv (x_1 - 5)^2 + x_2^2 \leq 25$ $c_2(\mathbf{x}) \equiv (x_1 - 8)^2 + (x_2 + 3)^2 \geq 7.7$
SRN	$x_1 \in [-20, 20]$ $x_2 \in [-20, 20]$	$f_1(\mathbf{x}) = 2 + (x_1 - 2)^2 + (x_2 - 2)^2$ $f_2(\mathbf{x}) = 9x_1 - (x_2 - 1)^2$ $c_1(\mathbf{x}) \equiv x_1^2 + x_2^2 \leq 225$ $c_2(\mathbf{x}) \equiv x_1 - 3x_2 + 10 \leq 0$
TNK	$x_1 \in [0, \pi]$ $x_2 \in [0, \pi]$	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}) = x_2$ $c_1(\mathbf{x}) \equiv x_1^2 + x_2^2 - 1 - 0.1cos(16arctan\frac{x_1}{x_2}) \geq 0$ $c_2(\mathbf{x}) \equiv (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \leq 0.5$
OSY	$x_1 \in [0, 10]$ $x_2 \in [0, 10]$ $x_3 \in [1, 5]$ $x_4 \in [0, 6]$ $x_5 \in [1, 5]$ $x_6 \in [0, 10]$	$f_1(\mathbf{x}) = -[25(x_1 - 2)^2 + (x_2 - 2)^2 + (x_3 - 1)^2$ $+(x_4 - 4)^2 + (x_5 - 1)^2$ $f_2(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2$ $c_1(\mathbf{x}) \equiv x_1 + x_2 - 2 \geq 0$ $c_2(\mathbf{x}) \equiv 6 - x_1 - x_2 \geq 0$ $c_3(\mathbf{x}) \equiv 2 - x_2 + x_1 \geq 0$ $c_4(\mathbf{x}) \equiv 2 - x_1 + 3x_2 \geq 0$ $c_5(\mathbf{x}) \equiv 4 - (x_3 - 3)^2 - x_4 \geq 0$ $c_6(\mathbf{x}) \equiv (x_5 - 3)^2 + x_6 - 4 \geq 0$

TABLE 4.3: Summary of CONSTR, Two-bar Truss and Welded Beam problems used in the benchmark experiments.

Benchmark Experiments		
Problem Name	Input Space	Objectives $f_k(\mathbf{x})$ and Constraints $c_j(\mathbf{x})$
CONSTR	$x_1 \in [0.1, 10]$ $x_2 \in [0, 5]$	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}) = \frac{(1+x_2)}{x_1}$ $c_1(\mathbf{x}) \equiv x_2 + 9x_1 \geq 6$ $c_2(\mathbf{x}) \equiv -x_2 + 9x_1 \geq 1$
Two-bar Truss Design	$x_1 \in [0, 0.01]$ $x_2 \in [0, 0.01]$ $x_3 \in [1, 3]$	$f_1(\mathbf{x}) = x_1\sqrt{16 + x_3^2} + x_2\sqrt{1 + x_3^2}$ $f_2(\mathbf{x}) = \max(\frac{20\sqrt{16+x_3}}{x_1 x_3}, \frac{80\sqrt{1+x_3^2}}{x_2 x_3})$ $c_1(\mathbf{x}) \equiv \max(\frac{20\sqrt{16+x_3}}{x_1 x_3}, \frac{80\sqrt{1+x_3^2}}{x_2 x_3}) \leq 10^5$
Welded Beam Design	$h \in [0.125, 5]$ $b \in [0.125, 5]$ $l \in [0.1, 10]$ $t \in [0.1, 10]$	$f_1(\mathbf{x}) = 1.10471h^2l + 0.04811tb(14 + l)$ $f_2(\mathbf{x}) = \frac{2.1952}{t^3 b}$ $c_1(\mathbf{x}) \equiv 13600 - \tau(\mathbf{x}) \geq 0$ $c_2(\mathbf{x}) \equiv 30000 - \frac{504000}{t^2 b} \geq 0$ $c_3(\mathbf{x}) \equiv b - h \geq 0$ $c_4(\mathbf{x}) \equiv 64746.022(1 - 0.0282346t)tb^3 - 6000 \geq 0$ $\tau(\mathbf{x}) = \sqrt{\gamma(\mathbf{x})^2 + \epsilon(\mathbf{x})^2 + \frac{l\gamma(\mathbf{x})\epsilon(\mathbf{x})}{\sqrt{0.25(l^2+(h+t)^2)}}}$ $\gamma(\mathbf{x}) = \frac{6000}{\sqrt{2}hl}$ $\epsilon(\mathbf{x}) = \frac{6000(14+0.5l)\sqrt{0.25(l^2+(h+t)^2)}}{2\sqrt{2}hl(\frac{l^2}{12+0.25(h+t)^2})}$

Figure 4.7 and 4.8 show the average results of each method on these experiments with the corresponding error bars. In these experiments, when a particular method outputs an infeasible solution, (*i.e.*, a solution that does not fulfil at least one of the constraints), that result is ignored. To guarantee a fair comparison, we have also recorded the fraction of times that an infeasible solution is returned by each method. If the performance of two methods is similar, it will be preferred the method that gives a lower percentage of infeasible solutions. In practice, we have observed that all the BO methods compared tend to provide a similar fraction of infeasible points. An exception is the random search strategy that systematically tends to recommend infeasible solutions. The complete results are found in the supplementary material.

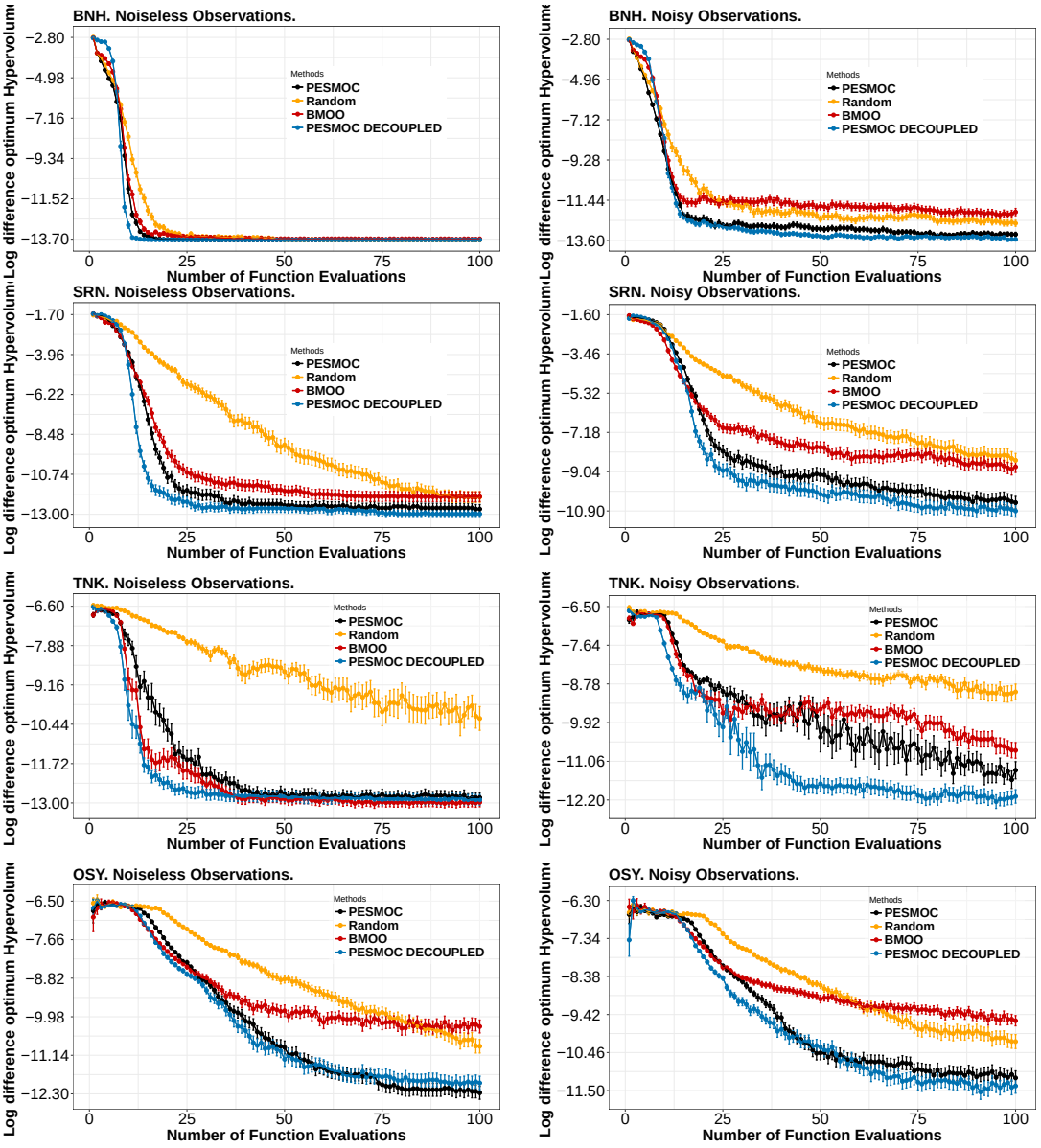

FIGURE 4.7: Results for the problems BNH, SRN, TNK and OSY. Noiseless and noisy settings. The plots show the average log difference w.r.t to the optimal hyper-volume.

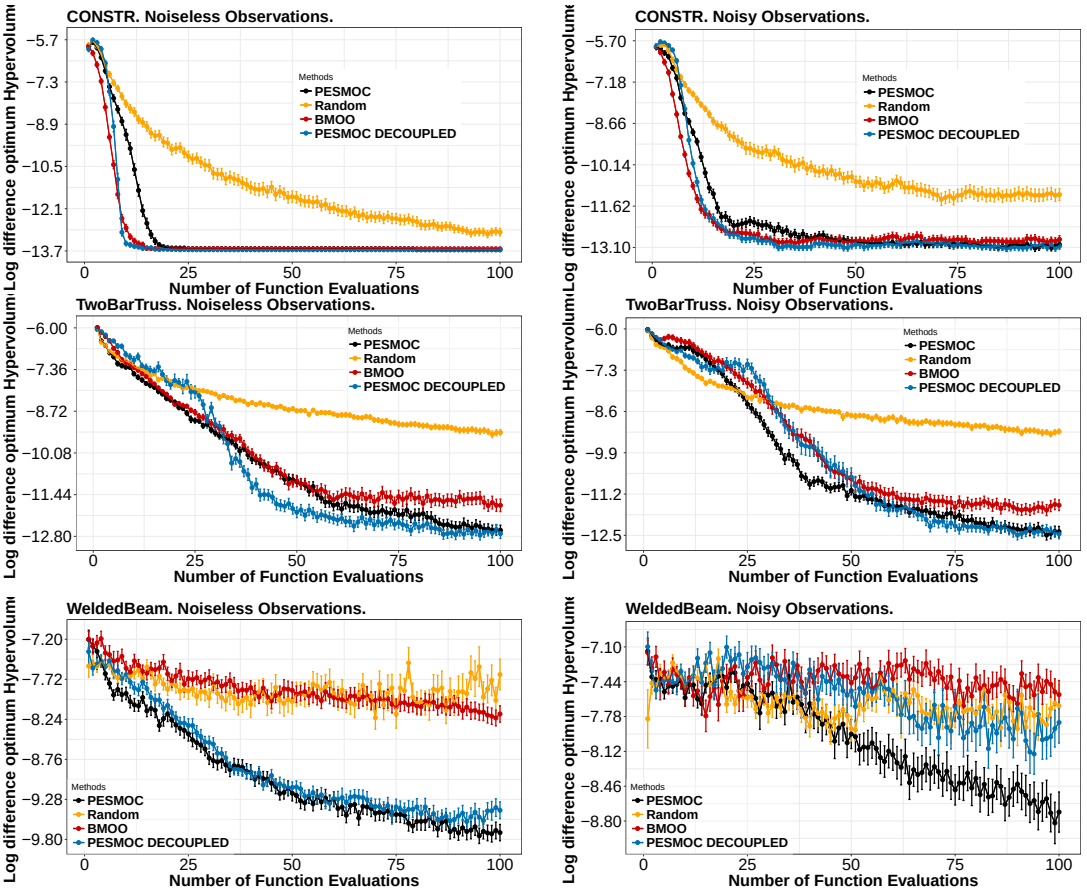

FIGURE 4.8: Results for CONSTR, TwoBarTruss and WeldedBeam. Noiseless and noisy settings. The plots show the average log difference w.r.t to the optimal hyper-volume.

We observe that, on average, PESMOC in the coupled and decoupled setting, outperforms the other methods for multi-objective constrained optimization. BNH is solved pretty fast by all methods, but PESMOC, under a decoupled evaluation setting, obtains better results with a smaller number of evaluations. These differences are also notable in the noisy setting. In this case, PESMOC clearly outperforms BMOO and the random search strategy. In the SRN, TNK and OSY problems, results are more or less the same but with bigger differences among the methods. BMOO tends to systematically perform worse in the presence of noise. Importantly, PESMOC decoupled performs significantly better on TNK. This is because this strategy is able to focus on the evaluation of the most difficult black-box functions. More precisely, in this problem one of the constraints plays a critical role in the identification of the Pareto optimal set and PESMOC decoupled is able to focus on its evaluation. This is a clear example of the benefits of a decoupled evaluation setting. The OSY problem has a higher dimensionality and hence, due to the greedy nature of BMOO that tends to explore too much, PESMOC approaches clearly do better.

The problem CONSTR is very easy to solve so BMOO does a good job on it, leaving PESMOC behind but resulting in the same performance as PESMOC decoupled. TwoBarTruss has the same nature as TNK, with the optimum lying in the frontier of the feasible and infeasible space. Again, PESMOC decoupled explores massively the constraints, solving the problem and giving better results that the other methods

with a smaller number of evaluations. In the noise scenario, however, both PESMOC approaches tie. The last problem reported is WeldedBeam, where both PESMOC approaches outperform the other methods. In the noisy scenario PESMOC under a coupled evaluation setting wins. We believe that model misspecification and the influence of noise may affect negatively the decoupled approach in certain scenarios.

4.4.4 Finding an Optimal Ensemble of Decision Trees

We compare the different methods on a practical problem in which the optimal hyper-parameters of an ensemble of decision trees are optimized. We consider two objectives: the prediction error of the ensemble and its size. These two objectives are conflictive since smaller ensembles will have in general higher error rates and the other way around. The ensemble size is related to the storage requirements and also to the speed of classification, which can play a critical role in real-time prediction systems. The dataset considered is the German Credit dataset, which is extracted from the UCI repository (Dheeru and Karra Taniskidou, 2017). This is a binary classification dataset with $1,000$ instances and 20 attributes. The prediction error is measured using a 10-fold-cross validation procedure that is repeated 5 times to reduce the variance of the estimates. We measure the ensemble size in terms of the logarithm of the sum of the total number of nodes in each of trees of the ensemble.

To get ensembles of decision trees with good prediction properties it is essential to enforce diversity among the ensemble classifiers (Dietterich, 2000). In particular, if all the decision trees of the ensemble are equal, there is no expected gain from aggregating their predictions. However, too much diversity in the ensemble can also lead to a poor prediction performance. For example, if the predictions made are completely random, one cannot obtain improved results by aggregating the individual classifiers. Therefore, we consider here several mechanisms to encourage diversity in the ensemble, and let the amount of diversity be specified in terms of adjustable parameters.

To build the ensemble we employ decision trees in which the best split at each node corresponds to the attribute that decreases the most the data impurity among a randomly chosen set of attributes (we use the DecisionTree implementation provided in the python package scikit-learn), and the number of random attributes is an adjustable parameter. This is the approach followed in random forest (Breiman, 2001). Each tree is trained on a random subset of the training data of a particular size, which is another adjustable parameter. This approach is known in the literature as subbagging (Bühlmann and Yu, 2002). We consider also an extra method to introduce diversity known as class-switching (Martínez-Muñoz and Suárez, 2005). In class-switching, the labels of a random fraction of the training data are changed to a different class. The final ensemble prediction is computed by majority voting.

More precisely, the adjustable parameters of the ensemble are: the number of decision trees built (between 1 and $1,000$), the number of random chosen attributes considered at each split in the building process of each tree (between 1 and 20), the minimum number of samples required to split a node (between 2 and 200), the fraction of randomly selected training data used to build each tree (between 0.5 and 1.0), and the fraction of training instances whose labels are changed after the sub-sampling process (between 0.0 and 0.7).

A problem of classification ensembles is that computing predictions can take much longer than using a single classifier. The reason for this is that one has to query all the ensemble classifiers about the class label of each test instance. A potential way of

accelerating predictions is to use a dynamic ensemble pruning technique (Hernández-Lobato et al., 2009). Assume that for a test instance we have queried only a fraction of the ensemble classifiers. It is possible to estimate the probability that the majority vote decision of the ensemble is not changed by the votes of the remaining classifiers. If this probability exceeds a particular threshold (*e.g.*, 99%), the querying process can be early stopped and the current majority class can be returned as the final ensemble prediction. Therefore, we introduce as a constraint of the optimization problem, that the average speed up factor of the classification process given by the previous dynamic ensemble pruning technique is at least 25%. We have carefully chosen this value to guarantee that the constraint is active at the optimal solution. In practice, the methods compared rarely provide infeasible solutions. If this is the case, we simply ignore those recommendations.

Note that the setting described is suited for the decoupled version of PESMOC since both objectives and the constraint can be evaluated separately. In particular, the total number of nodes is estimated by building only once the ensemble without leaving any data aside for validation, as opposed to the cross-validation approach used to estimate the ensemble error, which requires to build several ensembles on subsets of the data, to then estimate the prediction error on the data left out for validation. Similarly, evaluating the constraint involves building a lookup table whose entries indicate, for each different number of classifiers queried so far, how many votes of the most common class are needed to early stop the prediction process. This table is expensive to build and is different for each ensemble size. See (Hernández-Lobato et al., 2009) for further details.

We report in Figure 4.9 the results obtained for each method after 100 and 200 evaluations of the corresponding black-box functions. This figure shows the average Pareto front obtained by each method across the 100 different repetitions of the experiments. The Pareto front is simply given by the objective values associated to the recommendation made by each method. In general, and assuming minimization, the higher the volume of points that is above this set of points in the objective values space the better the performance of a method, as estimated by the hyper-volume metric. We observe that PESMOC outperforms BMOO and the random search strategy. Furthermore, PESMOC decoupled obtains better results that PESMOC. More precisely, PESMOC and PESMOC decoupled find better solutions in the sense that the ensembles obtained have a lower size and a smaller prediction error. Last, we note that BMOO is able to find the ensembles of the smallest size, but with higher levels of error, in a smaller number of evaluations.

We also show in Table 4.4 the average hyper volume of the solutions provided by each method. In general, a higher hyper-volume implies that the method gives better results. The values obtained agree with the previous figure. Namely, PESMOC decoupled outperforms the other methods, followed closely by PESMOC in a coupled setting, BMOO and the random search strategy. After 200 evaluations the differences in the hyper-volume between PESMOC decoupled and the other methods become bigger. This is probably a consequence of PESMOC decoupled performing more evaluations of the most complicated black-box function.

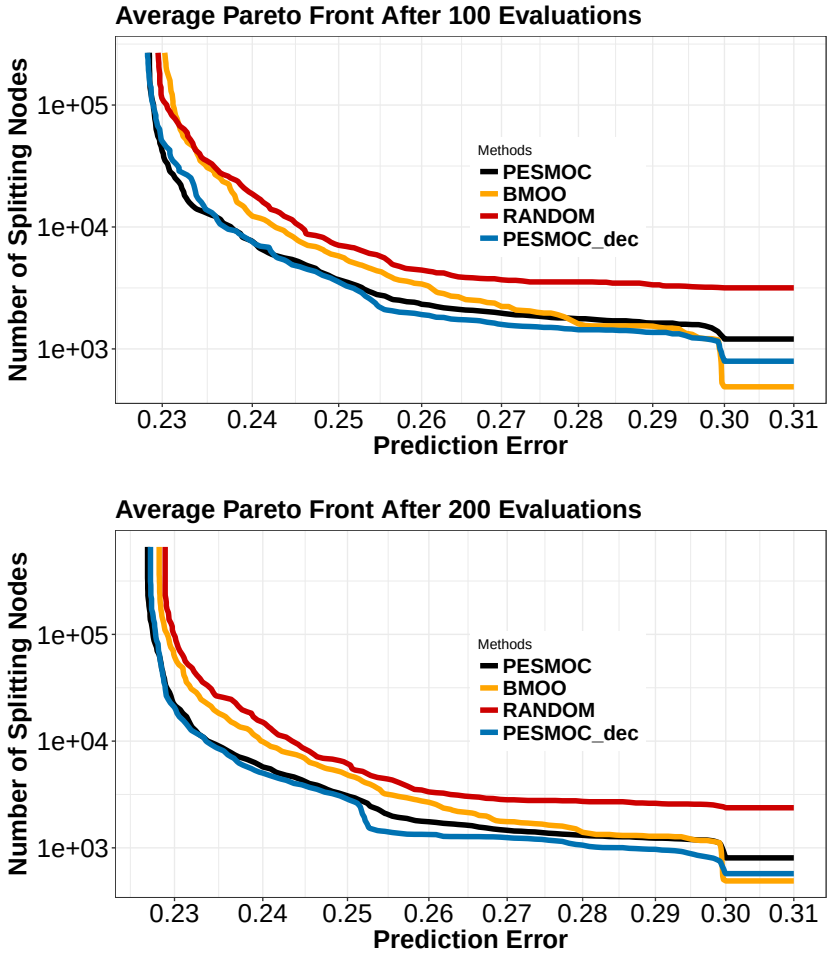

FIGURE 4.9: Results of each method on the problem of finding an optimal ensemble of classification trees. The Pareto frontier is shown for each method. The volume of points above the frontier (hyper-volume) represents the quality of the solution. A wider volume is always better.

TABLE 4.4: Average hyper-volume of each method on the task of finding an optimal classification ensemble. Larger hyper-volumes means better quality. PESMOC c. means PESMOC coupled and PESMOC d. means PESMOC decoupled.

# Eval.	PESMOC c.	PESMOC d.	BMOO	Random
100	0.309 ± 0.001	$\mathbf{0.311 \pm 0.002}$	0.293 ± 0.001	0.265 ± 0.002
200	0.325 ± 0.001	$\mathbf{0.338 \pm 0.001}$	0.309 ± 0.001	0.279 ± 0.001

In the problem described, we expect the prediction error to be the black-box function with the most important role in solving the optimization problem. Probably, it is more difficult to model than the the ensemble size or the speed-up factor due to the dynamic pruning technique. To check this hypothesis we record for PESMOC decoupled the number of times that each black-box function is evaluated. The average results obtained are shown in Figure 4.10. This figure shows, for each iteration of the optimization process, the average number of evaluations of each black-box function performed by PESMOC decoupled. We can see that the previous hypothesis is validated by the plot. Namely,

the prediction error is evaluated more frequently than the other black-box functions. This also explains the better results obtained by PESMOC decoupled. In particular, this technique is able to focus on the evaluation of the most important black-box function. Of course, the prediction error takes more time to evaluate than the other black-box functions, so PESMOC decoupled also takes a bit more time than the other techniques. In any case, this result illustrates the potential benefits of a decoupled evaluation strategy, which chooses not only at which point to perform the evaluation, but also which black-box function should be evaluated each time.

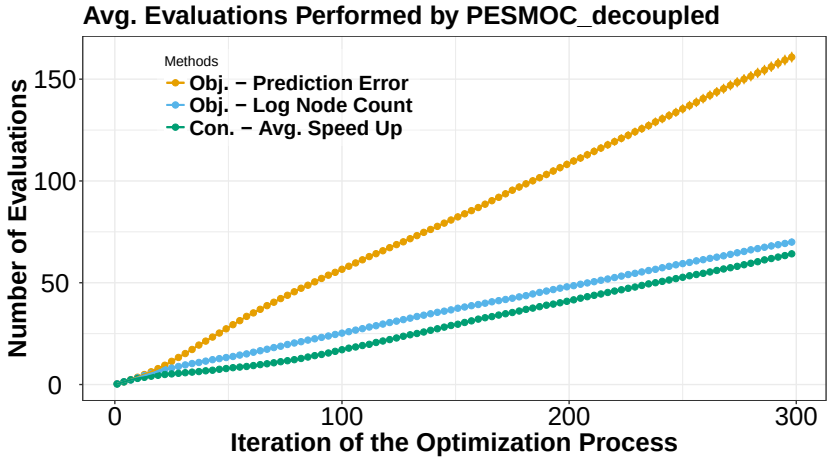

FIGURE 4.10: Evaluations of each black-box function made by PESMOC decoupled in the problem of finding an optimal ensemble of decision tree classifiers. The error objective is black-box function chosen most frequently for evaluation.

4.4.5 Finding an Optimal Deep Neural Network

In this section we evaluate the performance of the different methods on the task of finding an optimal deep neural network on the MNIST dataset (LeCun et al., 2010). This dataset contains a training set of 60,000 instances. The objectives that we consider for this problem include minimizing the prediction error of the neural network on a validation dataset of 10,000 instances (extracted from the original training set) and minimizing the time that such a neural network will take for making predictions on such a set. Note that these are conflictive objectives in the sense that most probably minimizing the prediction error on the validation set will require bigger neural networks with a larger number of hidden units and layers. Of course, these neural networks will require longer prediction times. Conversely, the minimization of the prediction time will probably involve using neural networks of smaller size whose prediction performance will be worse.

Besides this, we also consider that we may be interested in codifying such a neural network into a chip. This can be interesting for example if we would like to use that neural network in an electronic device such as an smart-phone. Motivated by this scenario we propose to constrain the optimization problem in such a way that the area of the resulting deep neural network, after being codified into a chip, is below one squared millimeter. We have carefully chosen this value to guarantee that the constraint is active at the optimal solution. To measure this area we use the hardware simulator Aladdin

(Shao et al., 2014), which given a computer program describing the operations carried out by the deep neural network, outputs an estimate of the area of a chip implementing those operations.

In practice, the methods compared rarely provide infeasible solutions. If this is the case, we simply ignore those recommendations. To train the deep neural network we use the Keras library (Chollet, 2015). Prediction time is measured on the validation set of $10,000$ training instances. The prediction time is normalized by the smallest possible prediction time, which corresponds to a neural network of a single layer with 5 hidden units.

Importantly, the different black-box functions involved in the optimization problem just described can be evaluated separately in a decoupled way. The reason for this is that the prediction time and the chip area does not need specific values for the neural network weights and biases. These can simply be initialized randomly. These two black-box functions only depend on the particular architecture of the neural network (the number of layers and the number of hidden units on each layer). Therefore, the problem described is adequate for PESMOC decoupled. The specific steps involved in measuring the different black-boxes are displayed in Figure 4.11.

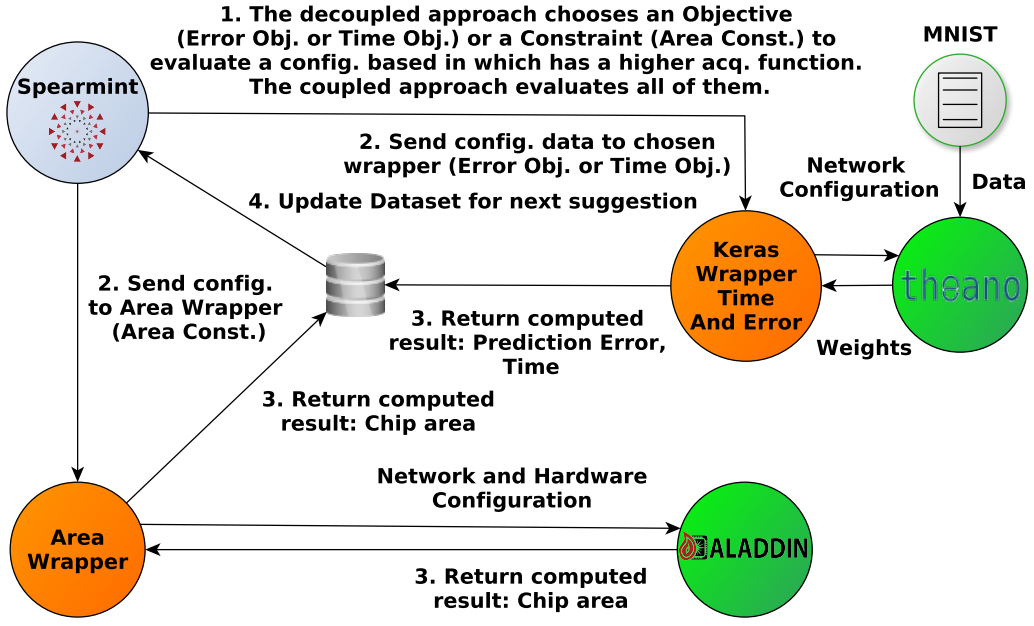

FIGURE 4.11: Diagram showing the architecture of systems that we have used for the deep neural network experiments. The different steps involved in the evaluation of each black-box function are displayed here.

The input parameters that we consider for optimization in this problem are: The logarithm of the ℓ_1 and ℓ_2 weight regularizers; the dropout probability; the logarithm of the initial learning rate; the number of hidden units per layer; and the number of hidden layers. We have also considered two variables that have an impact in the hardware implementation of the neural network. Namely, the logarithm (in base 2) of the array partition factor and the loop unrolling factor. See (Shao et al., 2014) for further details.

A summary of the parameters considered, their potential values, and their impact in each black-box function (prediction error, time and chip area) is displayed in Table 4.5.

TABLE 4.5: Parameter space of the deep neural network experiments. PE = Prediction error. T = Time. CA = Chip area.

Parameter	**Min**	**Max**	**Step**	**Black-box**
Hidden Layers	1	3	1	PE/T/CA
Neurons per Layer	5	300	1	PE/T/CA
Learning rate	e^{-20}	1	ϵ	PE
Dropout rate	0	0.9	ϵ	PE
ℓ_1 penalty	e^{-20}	1	ϵ	PE
ℓ_2 penalty	e^{-20}	1	ϵ	PE
Memory partition	1	32	2^x	CA
Loop unrolling	1	32	2^x	CA

In these experiments we evaluated the performance of each method after 50 and 100 evaluations of the black-boxes. Furthermore, the training of the deep neural networks is carried out using ADAM with the default parameter values (Kingma and Ba, 2014). The loss function is the categorical cross-entropy. The last layer of the neural network contains 10 units and a soft-max activation function. All other layers use Re-Lu as the activation function. Finally, each neural network is trained during a total of 150 epochs using mini-batches of size $4,000$ instances.

The average results obtained across 100 repetitions of the experiments can be shown in Figure 4.12 after 50 and 100 evaluations of the black-boxes. This figure shows the average Pareto frontier obtained by each method. As in the previous experiments PESMOC decoupled outperforms the others methods. PESMOC is the second best method, giving solutions with a best trade-off between prediction error and time ratio (prediction time normalized with respect to the smallest possible prediction time), under the constraint that the chip area is below the specified value. BMOO also gives better results than the random search strategy which is the worst performing method. These experiments show strong empirical evidence supporting that PESMOC is a competitive strategy for constrained multi-objective optimization. We also provide the average hyper-volume of the solutions found by each method after 50 and 100 evaluations of the black-boxes. These results are displayed in Table 4.6. We observe that PESMOC decoupled outperforms the rest of the methods. This strategy finds solutions that, on average, have a significantly higher hyper-volume than any of the other methods. Furthermore, PESMOC is able to find solutions that are slightly better than those obtained by BMOO.

TABLE 4.6: Average hyper-volume of each method on the task of finding an optimal neural network on the MNIST dataset. Larger hyper-volumes means better quality. PESMOC c. means PESMOC coupled and PESMOC d. means PESMOC decoupled.

# Eval.	PESMOC c.	PESMOC d.	BMOO	Random
50	47.230 ± 0.079	$\mathbf{47.608 \pm 0.056}$	46.104 ± 0.267	44.886 ± 0.135
100	47.621 ± 0.054	$\mathbf{48.069 \pm 0.039}$	47.304 ± 0.083	45.714 ± 0.093

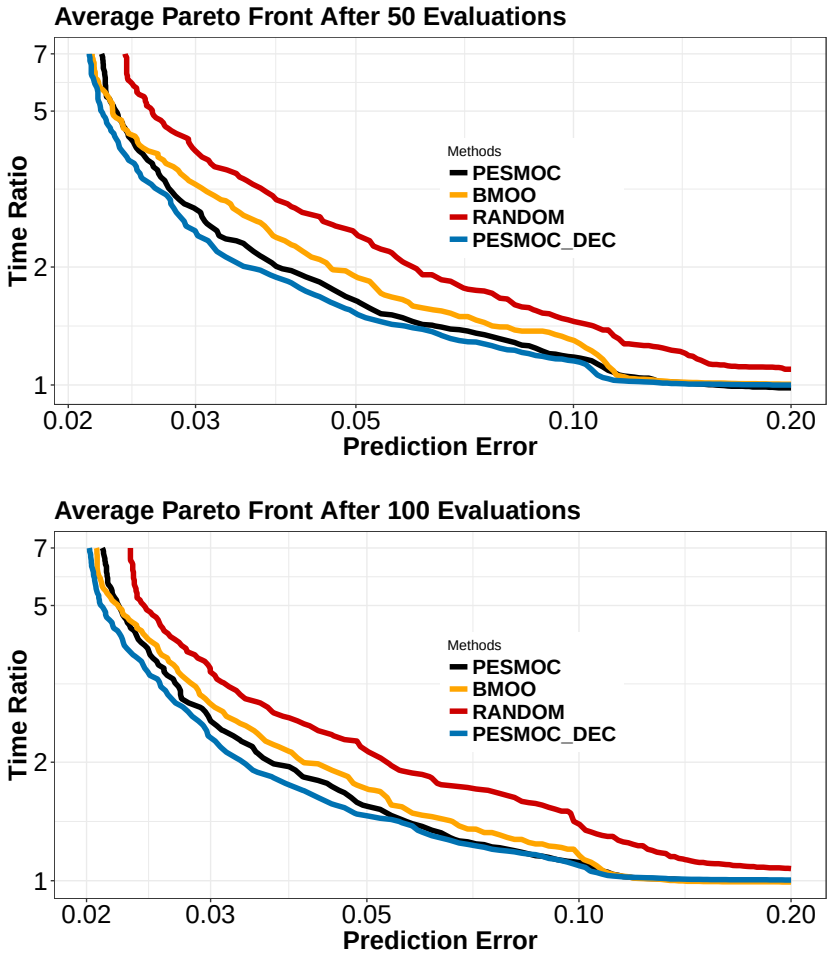

FIGURE 4.12: Results of each method on the problem of finding an optimal neural network on the MNIST dataset. The Pareto frontier is shown for each method. The volume of points above the frontier (hyper-volume) represents the quality of the solution. A wider volume is always better.

We also analyze in these experiments which black-box function is evaluated more frequently by PESMOC decoupled. For this, we record the number of evaluations of each black-box made by this strategy as a function of the total evaluations made. The average results obtained across the 100 repetitions of the experiments are shown in Figure 4.13. We observe that PESMOC decoupled tends to evaluate a significantly higher number of times the prediction error of the neural network. This also explains the better results obtained by this strategy which is able to focus on the evaluation of the black-box

function that is most difficult to model or that plays a critical role in the optimization problem. Of course, the prediction error takes more time to evaluate than the other black-box functions, so PESMOC decoupled also takes a bit more time than the other techniques in this problem. In any case, this result illustrates again the potential benefits of a decoupled evaluation strategy, which can be used to choose in an intelligent way which black-box function should be evaluated next at each iteration of the optimization process.

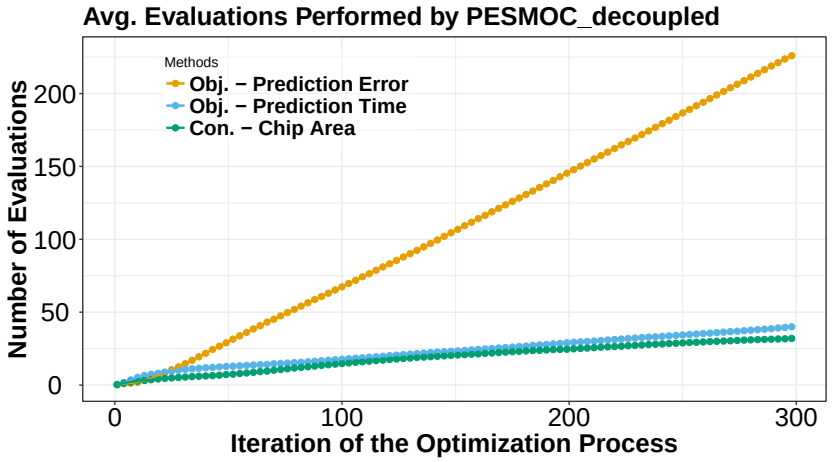

FIGURE 4.13: Evaluations of each black-box function made by PESMOC decoupled in the problem of finding an optimal neural network on the MNIST dataset. The error is black-box function that is chosen most frequently for evaluation by PESMOC decoupled.

4.5 Conclusions

We have described an information-based approach that can be used to address a wide range of Bayesian optimization problems, including multiple objectives and several constraints. Motivated by the lack of methods that are available to solve these problems with an adequate exploration-exploitation balance, PESMOC has been presented. At each iteration, PESMOC evaluates the objective functions and the constraints at an input location that is expected to reduce the entropy of the posterior distribution of the Pareto set in the feasible space the most. The computation of the expected reduction of the entropy of such a random variable is intractable. Nevertheless, we have described how the required computations can be approximated using expectation propagation (Minka, 2001a).

Importantly, in the proposed approach the acquisition function can be expressed as a sum of a different acquisition per black-box function (objective or constraint). This means that PESMOC allows for a decoupled evaluation setting. In this scenario one is not only interested in finding which is the next point at which the black-boxes should be evaluated, but also in finding what black-box function or subset of these should be evaluated next. For this, one simply has to optimize the individual acquisition functions and compare their corresponding values. Other related methods from the literature do not allow for such a setting since the utility function they are based on (the improvement of the hyper-volume metric) requires the evaluation of all the black-box functions.

We have illustrated in a wide range of experiments, including synthetic, benchmark and real-world problems the benefits of PESMOC. Furthermore, we have compared results in these experiments with a state-of-the-art method for constrained multi-objective Bayesian optimization, BMOO (Féliot et al., 2017), which is based on the expected improvement of the hyper-volume, and with a baseline method that explores the input space uniformly at random. These experiments show that PESMOC is able to obtain better results in terms of the hyper-volume of the recommendations made. More precisely, it provides estimates of the Pareto set in the feasible space that are more accurate with a smaller number of evaluations. Furthermore, PESMOC in a decoupled setting is able to provide significantly better results in several of these problems. This is very useful in practical situations in which the objectives and the constraints are very expensive to evaluate.

A more general scenario involves the evaluation of several points in parallel. If the scenario also involves the simultaneous optimization of several objectives and constraints, this scenario can be defined as the batch constrained multi-objective scenario. In the next chapter of the thesis, we are going to present an enhancement of PESMOC to tackle the batch constrained multi-objective scenario. In this scenario we also have to optimize several objectives and constraints but now we have several resources to evaluate suggestions in parallel. The method that we are going to present is able to suggest several points for evaluation at the same time at every iteration. By contrast PESMOC can only suggest a single point for evaluation at each iteration. It is, hence, an improved version of PESMOC that will require additional methodologies and more approximations. These approximations will be carried out using again EP.

Chapter 5

Parallel Predictive Entropy Search For Multi-Objective Bayesian Optimization With Constraints

As we have seen in the previous chapter, real-world problems often involve the optimization of several objectives under multiple black-boxes. We have also described how BO can recommend a valid Pareto set at the end of its execution. A limitation of the described methodology, however, is that current BO methods, as PESMOC, for these problems, choose a point at a time at which to evaluate the black-boxes. If the expensive evaluations can be carried out in parallel (as when a cluster of computers is available), this results in a waste of resources. In this chapter, we introduce PPESMOC, Parallel Predictive Entropy Search for Multi-objective Optimization with Constraints, an extension of PESMOC for solving the problems described. PPESMOC selects, at each iteration, a batch of input locations at which to evaluate the black-boxes, in parallel, to maximally reduce the entropy of the problem's solution. To our knowledge, this is the first batch method for constrained multi-objective BO. In this chapter we present empirical evidence in the form of synthetic, benchmark and real-world experiments that illustrate the effectiveness of PPESMOC.

5.1 Introduction

A limitation of BO methods for constrained multi-objective optimization like PESMOC, which was described in the previous chapter, is that they choose a point at a time at which to evaluate the black-boxes. Assume that a cluster of computers or some other resource is available to perform the evaluation of the black-boxes at several points in parallel. If only a single point is evaluated each time, this results in a a waste of resources and leads to sub-optimal optimization results. The problem described can be solved by using BO methods that suggest not only a single point at which to evaluate the black-boxes, but a batch or collection of points of adjustable size (Azimi et al., 2012; Bergstra et al., 2011; González et al., 2016; Shah and Ghahramani, 2015).

To the best of our knowledge, no parallel BO method has been proposed to deal with the optimization of multiple objectives under several constraints. Only sequential methods exist. Therefore, the literature about BO is missing important methods to

address BO problems with the characteristics described. In this chapter, we propose a BO method that can precisely address the problems described and can suggest, at each iteration, a batch of points at which to evaluate all the black-boxes in parallel. The method proposed is based on an extension of the PESMOC method described in the previous chapter and an extension of sequential constrained multi-objective BO (Shah and Ghahramani, 2015). More precisely, the acquisition function that we consider receives as an input a batch of candidate input locations at which to perform the evaluation of the black-boxes in parallel and estimates the expected reduction in the entropy of the solution of the optimization problem. The difference with PESMOC is that here, the acquisition function evaluates a batch of points instead of a single one.

We have carried out extensive experiments to evaluate the performance of PPESMOC in synthetic, benchmark and real-world optimization problems. Furthermore, we have compared results with a base-line which chooses the points to evaluate at random and with a simple method that applies iteratively a sequential BO method for constrained multi-objective BO (as many times as the batch size). This last method introduces virtual observations (fantasies) to avoid choosing many times the same point. The results show that PPESMOC performs better than a random search strategy and similarly or better than sequential base-lines. The advantage of the proposed approach is, however, that its cost scales much better with respect to the batch size than the sequential base-lines.

5.2 Parallel Bayesian Optimization

Any sequential BO strategy can be transformed into a batch one by iteratively applying the sequential strategy B times. To avoid choosing similar points each time, one can simply hallucinate the results of the already chosen pending evaluations (Snoek et al., 2012). For this, the acquisition function is simply updated from $\alpha(\mathbf{x}|\mathcal{D})$ to $\alpha(\mathbf{x}|\mathcal{D} \bigcup (\mathbf{x}_i, \mathbf{h}_i), \forall \mathbf{x}_i \in \mathcal{P})$, where $\mathcal{D}$ are the data collected so far, $\mathcal{P}$ is the set of pending evaluations, and $\mathbf{h}_i$ denotes the hallucinated evaluation result for the pending evaluation $\mathbf{x}_i$. A simple approach is to update the surrogate model after choosing each batch point by setting $\mathbf{h}_i$ equal to the mean of the predictive distribution given by the GPs (Desautels et al., 2014). Of course, this strategy, to which we refer to as parallel sequential, has the disadvantage of requiring the optimization of the acquisition B times, and also updating the GPs using hallucinated observations. This is expected to lead to extra computational cost than in PPESMOC.

PPESMOC is a generalization of PESMOC, which was described in the previous chapter. PESMOC is the current state-of-the-art for solving constrained multi-objective BO problems. Nevertheless, PESMOC is a sequential BO method that can only suggest one point at a time to be evaluated. It cannot suggest a batch of points as PPESMOC. PESMOC also works by choosing the next candidate point as the one that is expected to reduce the most the entropy of the Pareto set in the feasible space. The required computations are also approximated using the expectation propagation algorithm (Minka, 2001a). Notwithstanding, the extension of PPESMOC over PESMOC is not trivial. In particular, in PPESMOC the acquisition function involves additional non-Gaussian factors (one per each point in the batch) and requires the computation of its gradients. These are not needed in PESMOC as the input dimensionality in that method is smaller, *i.e.*, D vs. $B \times D$.

5.3 Parallel Predictive Entropy Search for Multi-Objective Bayesian Optimization with Constraints

Consider N observations $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$ of the black-boxes obtained so far. We define $\mathbf{X}_{N+1} = \{\mathbf{x}_1, \ldots, \mathbf{x}_B\}$ as the batch of B points where the black-boxes should be evaluated at the next iteration. In this chapter, we describe how PPESMOC can be used to identify such a batch of points by maximizing an acquisition function.

5.3.1 Modeling the Black-boxes Using Gaussian Processes

As in PESMOC, we model each objective $f_k(\cdot)$ and constraint $c_j(\cdot)$ using a Gaussian process (GP) (Rasmussen, 2003). We assume independent GPs for each black-box function, objective or constraint. Consider the observations of a particular black-box function $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where $y_i = f(\mathbf{x}_i) + \epsilon_i$, with $f(\cdot)$ the black-box function and ϵ_i some Gaussian noise. A GP gives a distribution for the potential values of $f(\cdot)$ at a new set of input points $\mathbf{X}^\star = (\mathbf{x}_1^\star, \ldots, \mathbf{x}_B^\star)^{\mathrm{T}}$ of size B. Let $\mathbf{f}^\star = (f(\mathbf{x}_1^\star), \ldots, f(\mathbf{x}_B^\star))^{\mathrm{T}}$. The predictive distribution for $\mathbf{f}^\star$ is Gaussian. $p(\mathbf{f}^\star|\mathbf{y}) = \mathcal{N}(\mathbf{f}^\star|\mathbf{m}(\mathbf{X}^\star), \mathbf{V}(\mathbf{X}^\star))$, where $\mathbf{y} = (y_1, \ldots, y_N)^{\mathrm{T}}$ and the mean and covariances are, respectively:

$$\mathbf{m}(\mathbf{X}^\star) = \mathbf{K}_\star^{\mathrm{T}}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}\,, \qquad \mathbf{V}(\mathbf{X}^\star) = \mathbf{K}_{\star,\star} - \mathbf{K}_\star^{\mathrm{T}}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{K}_\star\,. \tag{5.1}$$

In Equation (5.1) σ^2 is the variance of the Gaussian noise; $\mathbf{K}_\star$ is a $N \times B$ matrix with the prior covariances between $\mathbf{f}^\star$ and each $f(\mathbf{x}_i)$; and $\mathbf{K}$ is a $N \times N$ matrix with the prior covariances among each $f(\mathbf{x}_i)$. That is $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, for some covariance function $k(\cdot, \cdot)$. Finally, $\mathbf{K}_{\star,\star}$ is a $B \times B$ matrix with the prior covariances for each entry in $\mathbf{f}^\star$.

5.3.2 Specification of the Acquisition Function

We choose $\mathbf{X}_{N+1}$ as the batch of points that maximizes the expected reduction in the entropy of the Pareto set in the feasible space, $\mathcal{X}^\star$. This is a popular strategy that has shown good empirical results in other optimization settings including in the PESMOC acquisition function presented in the previous chapter (Garrido-Merchán and Hernández-Lobato, 2019b; Hernández-Lobato et al., 2016, 2014; Shah and Ghahramani, 2015). Therefore, the PPESMOC acquisition function, $\alpha(\cdot)$, is:

$$\alpha(\mathbf{X}) = \mathrm{H}[p(\mathcal{X}^*|\mathcal{D})] - \mathbb{E}_{p(\mathbf{Y}|\mathcal{D},\mathbf{X})}[\mathrm{H}[p(\mathcal{X}^*|\mathcal{D} \cup (\mathbf{X}, \mathbf{Y})]]\,, \tag{5.2}$$

where $\mathbf{X}$ is the candidate batch of B points at which to evaluate the black-boxes; $\mathbf{Y}$ is a matrix with the set of B noisy evaluations associated to $\mathbf{X}$, for each black-box function; $\mathrm{H}[p(\mathbf{x})] = -\int p(\mathbf{x}) \log p(\mathbf{x}) d\mathbf{x}$ is the differential entropy of the distribution $p(\mathbf{x})$; the expectation is with respect to the posterior predictive distribution of $\mathbf{Y}$ at the candidate batch $\mathbf{X}$, given the data we have observed so far, $\mathcal{D}$; finally, $p(\mathcal{X}^*|\mathcal{D})$ is the probability distribution of potential Pareto sets $\mathcal{X}^\star$ given the data we have observed so far $\mathcal{D}$. The distribution $p(\mathbf{Y}|\mathcal{D}, \mathbf{X})$, is given by the product of the predictive distributions of each GP, as indicated in Equation (5.1), for each black-box function (the level of Gaussian noise, $\mathbf{I}\sigma^2$, has to be added to each covariance matrix). Namely, $p(\mathbf{Y}|\mathcal{D}, \mathbf{X}) = \prod_{k=1}^{K} p(\mathbf{y}_k^o|\mathcal{D}, \mathbf{X}) \prod_{j=1}^{J} p(\mathbf{y}_j^c|\mathcal{D}, \mathbf{X})$, where $\mathbf{y}_k^o$ and $\mathbf{y}_j^c$ are B-dimensional vectors with the potential observations of each black-box function, objective or constraint, for each point in the batch $\mathbf{X}$. Recall that an independent GP is modeling each black-box function.

Note that Equation (5.2) involves the entropy of $\mathcal{X}^\star$ which can be very difficult to compute. To simplify the computation of the acquisition function, we use the same trick based on the symmetry of mutual information that we carried out in the PESMOC acquisition function. For this, we observe that Equation (5.2) is the mutual information between $\mathcal{X}^\star$ and $\mathbf{Y}$, $\mathrm{I}(\mathcal{X}^\star, \mathbf{Y})$. Since the mutual information is symmetric, *i.e.*, $\mathrm{I}(\mathcal{X}^\star, \mathbf{Y}) = \mathrm{I}(\mathbf{Y}, \mathcal{X}^\star)$, we swap the roles of $\mathcal{X}^\star$ and $\mathbf{Y}$ obtaining:

$$\alpha(\mathbf{X}) = \mathrm{H}[p(\mathbf{Y}|\mathcal{D}, \mathbf{X})] - \mathbb{E}_{p(\mathcal{X}^\star|\mathcal{D})}[\mathrm{H}[p(\mathbf{Y}|\mathcal{D}, \mathbf{X}, \mathcal{X}^\star)]], \tag{5.3}$$

where $p(\mathbf{Y}|\mathcal{D}, \mathbf{X}, \mathcal{X}^\star)$ is the predictive distribution for the values of the black-boxes at $\mathbf{X}$, given the observed data $\mathcal{D}$, and given that the solution of the optimization problem, *i.e.*, the Pareto set in the feasible space, is given by $\mathcal{X}^\star$. Furthermore, the expectation is with respect to $p(\mathcal{X}^\star|\mathcal{D})$. Namely, the posterior distribution of $\mathcal{X}^\star$ given the data we have observed so far $\mathcal{D}$.

Importantly, the first term in Equation (5.3) can be evaluated analytically since it is just the entropy of the predictive distribution, $\mathrm{H}[p(\mathbf{Y}|\mathcal{D}, \mathbf{X})]$, which is a factorizing $K + J$ dimensional multivariate Gaussian. In particular,

$$\mathrm{H}[p(\mathbf{Y}|\mathcal{D}, \mathbf{X})] = 0.5((K + J)B \log(2\pi e) + \sum_{k=1}^{K} \log |\mathbf{V}_k^o(\mathbf{X})| + \sum_{j=1}^{J} |\mathbf{V}_j^c(\mathbf{X})|) \tag{5.4}$$

where $\mathbf{V}_k^o(\mathbf{X})$ and $\mathbf{V}_j^c(\mathbf{X})$ are the covariance matrices of the predictive distribution for each black-box function (objective or constraint, respectively) given by Equation (5.1), plus the corresponding additive Gaussian noise, $\mathbf{I}\sigma^2$.

Moreover, the expectation in Equation (5.3) can be approximated by a Monte Carlo average. More precisely, one can generate random samples of the black-box functions using a random-feature approximation of each GP as we described in the previous chapter. These samples can then be easily optimized to generate a sample from $p(\mathcal{X}^\star|\mathcal{D})$. Because the samples of the black-box function are cheap to evaluate, this optimization process has little cost and can be done using, *e.g.*, a grid of points. In practice, we use a finite Pareto set approximated by 50 points. A problem, however, is evaluating the second term that appears in Equation (5.3). Namely, the entropy of $p(\mathbf{Y}|\mathcal{D}, \mathbf{X}, \mathcal{X}_s^\star)$, for a particular sample of $\mathcal{X}^\star$, $\mathcal{X}_s^\star$. Such a distribution is intractable. As in the case of PESMOC, we resort to expectation propagation to approximate its value (Minka, 2001a).

5.3.3 Approximating the Conditional Predictive Distribution

Assume both $\mathcal{X}$ and $\mathcal{X}^\star$ have finite size and that $\mathcal{X}^\star$ is known. Later on, we will show how to approximate $\mathcal{X}$ with a finite size set. Let $\mathbf{F}$ and $\mathbf{C}$ be a matrix with the actual objective and constraint values associated to $\mathcal{X}$. Then,

$$p(\mathbf{Y}|\mathcal{D}, \mathbf{X}, \mathcal{X}^\star) = \int p(\mathbf{Y}|\mathbf{X}, \mathbf{F}, \mathbf{C}) p(\mathcal{X}^\star|\mathbf{F}, \mathbf{C}) p(\mathbf{F}|\mathcal{D}) p(\mathbf{C}|\mathcal{D}) d\mathbf{F} d\mathbf{C}\,, \tag{5.5}$$

where $p(\mathbf{Y}|\mathbf{X}, \mathbf{F}, \mathbf{C}) = \prod_{b=1}^{B} \prod_{k=1}^{K} \delta(y_b^k - f_k(\mathbf{x}_b)) \prod_{j=1}^{J} \delta(y_b^j - c_j(\mathbf{x}_b))$, with y_b^k the evaluation corresponding to the k-th objective associated to the batch point $\mathbf{x}_b$, y_b^j the evaluation corresponding to the j-th constraint associated to the batch point $\mathbf{x}_b$, $\delta(\cdot)$ a Dirac's delta function and B the batch size. We have assumed no additive Gaussian noise. In the case of noisy observations, one simply has to replace the delta functions with Gaussians with the corresponding variance, σ^2.

In Equation (5.5) $p(\mathbf{F}|\mathcal{D})$ and $p(\mathbf{C}|\mathcal{D})$ denote the posterior predictive distribution for the objectives and constraints, respectively. Note that we assume independent GPs. Therefore, these distributions factorize across objectives and constraints. They are Gaussians with parameters given in Equation (5.1). Last, in Equation (5.5) $p(\mathcal{X}^\star|\mathbf{F},\mathbf{C})$ is an informal probability distribution that takes value different from zero, only for a valid Pareto set $\mathcal{X}^\star$. More precisely, $\mathcal{X}^\star$ has to satisfy that $\forall \mathbf{x}^\star \in \mathcal{X}^\star$, $\forall \mathbf{x}' \in \mathcal{X}$, $c_j(\mathbf{x}^\star) \geq 0$, $\forall j$, and if $c_j(\mathbf{x}') \geq 0$, $\forall j$, then $\exists k$ s.t. $f_k(\mathbf{x}^\star) < f_k(\mathbf{x}')$. Namely, each point of the Pareto set has to be better than any other feasible point in at least one of the objectives. These conditions can be summarized as:

$$p(\mathcal{X}^\star|\mathbf{F},\mathbf{C}) \propto \prod_{\mathbf{x}^\star \in \mathcal{X}^\star} \left(\left[\prod_{j=1}^{J} \Phi_j(\mathbf{x}^\star) \right] \left[\prod_{\mathbf{x}' \in \mathcal{X}} \Omega(\mathbf{x}', \mathbf{x}^\star) \right] \right) \tag{5.6}$$

where $\Phi_j(\mathbf{x}^\star) = \Theta(c_j(\mathbf{x}^\star))$, with $\Theta(\cdot)$ the Heaviside step function, using the convention that $\Theta(0) = 1$. Furthermore,

$$\Omega(\mathbf{x}', \mathbf{x}^\star) = \left[\prod_{j=1}^{J} \Theta(c_j(\mathbf{x}')) \right] \Psi(\mathbf{x}', \mathbf{x}^\star) + \left[1 - \prod_{j=1}^{J} \Theta(c_j(\mathbf{x}')) \right] \cdot 1\,, \tag{5.7}$$

where $\Psi(\mathbf{x}', \mathbf{x}^\star) = 1 - \prod_{k=1}^{K} \Theta(f_k(\mathbf{x}^\star) - f_k(\mathbf{x}'))$. The goal of $\prod_{j=1}^{J} \Phi_j(\mathbf{x}^\star)$ in Equation (5.6) is to guarantee that every point in $\mathcal{X}^\star$ is feasible. Otherwise, $p(\mathcal{X}^\star|\mathbf{F},\mathbf{C})$ takes value zero. Similarly, $\Omega(\mathbf{x}', \mathbf{x}^\star)$ can be understood as follows: $\prod_{j=1}^{J} \Theta(c_j(\mathbf{x}')$ checks that $\mathbf{x}'$ is feasible. If $\mathbf{x}'$ is infeasible, we do not care and simply multiply everything by 1. Otherwise, $\mathbf{x}'$ has to be dominated by $\mathbf{x}^\star$. That is checked by $\Psi(\mathbf{x}', \mathbf{x}^\star)$. This last factor takes value one if $\mathbf{x}^\star$ dominates $\mathbf{x}'$ and zero otherwise. Summing up, the r.h.s. of Equation (5.6) takes value 1 only if $\mathcal{X}^\star$ is a valid Pareto set.

Critically, in Equation (5.5) all the factors that appear in the r.h.s are Gaussian, except for $p(\mathcal{X}^\star|\mathbf{F},\mathbf{C})$. The non-Gaussian factors contained in this distribution are approximated by Gaussians using expectation propagation (EP) (Minka, 2001a). Each $\Phi_j(\mathbf{x}^\star)$ factor is approximated by a univariate Gaussian that need not be normalized. Namely, $\Phi_j(\mathbf{x}^\star) \approx \tilde{\mathcal{N}}(c_j(\mathbf{x}^\star|\tilde{m}_j^{\mathbf{x}^\star}, \tilde{v}_j^{\mathbf{x}^\star}))$. The parameters of this Gaussian are tuned by EP. Similarly, each $\Omega(\mathbf{x}', \mathbf{x}^\star)$ is approximated by a product of K bivariate Gaussians and J univariate Gaussians that need not be normalized. That is,

$$\Omega(\mathbf{x}', \mathbf{x}^\star) \approx \prod_{k=1}^{K} \tilde{\mathcal{N}}(\boldsymbol{\upsilon}|\tilde{\mathbf{m}}_k^{\mathbf{x}^\star,\mathbf{x}'}, \tilde{\mathbf{V}}_k^{\mathbf{x}^\star,\mathbf{x}'}) \prod_{j=1}^{J} \tilde{\mathcal{N}}(c_j(\mathbf{x}')|\tilde{m}_j^{\mathbf{x}'}, \tilde{v}_j^{\mathbf{x}'})), \tag{5.8}$$

where $\boldsymbol{\upsilon} = (f_k(\mathbf{x}^\star), f_k(\mathbf{x}'))^{\mathrm{T}}$. The parameters of these Gaussians are also adjusted by EP. The approximate factors are refined iteratively until their parameters do not change. This ensures that they look similar to the corresponding exact factors.

In our experiments, and when running EP, we replace the set $\mathcal{X}$ in Equation (5.6) by a finite set given by $\{\mathbf{x}_n\}_{n=1}^{N} \bigcup \mathbf{X} \bigcup \mathcal{X}^\star$. Namely, the union of all points that have already being evaluated, the candidate batch $\mathbf{X}$ and the current Pareto set $\mathcal{X}^\star$ that has been sampled from $p(\mathcal{X}^\star|\mathcal{D})$ when using a Monte Carlo approximation of the expectation in the r.h.s. of Equation (5.3). These are the input points we have so far. Finally, the factors that belong to the batch $\mathbf{X}$ where the acquisition is going to be evaluated are only refined once, as in the PESMOC case.

5.3.4 PPESMOC's Acquisition Function

After EP has converged, the conditional predictive distribution in Equation (5.5) is approximated by replacing each non-Gaussian factor by the corresponding Gaussian approximation obtained by EP. Because all factors are then Gaussian, and the Gaussian family is closed under the product operation, their product can be easily evaluated, resulting in another Gaussian distribution for $p(\mathbf{Y}|\mathcal{D}, \mathbf{X}, \mathcal{X}^\star)$. Consider S Monte Carlo samples of $\mathcal{X}^\star$ to approximate the expectation in the r.h.s. of Equation (5.3). Let $\mathbf{V}_k^o(\mathbf{X}; s)^{\text{CPD}}$ and $\mathbf{V}_j^c(\mathbf{X}; s)^{\text{CPD}}$ denote the covariance matrices of the Gaussian approximation of $p(\mathbf{Y}|\mathcal{D}, \mathbf{X}, \mathcal{X}^\star)$ for each objective k and constraint j, for sample $\mathcal{X}_s^\star$. The PPESMOC's acquisition is simply given by the difference in the entropy before and after conditioning to $\mathcal{X}^\star$. Namely,

$$\alpha(\mathbf{X}) = \sum_{k=1}^{K} \log |\mathbf{V}_k^o(\mathbf{X})| + \sum_{j=1}^{J} \log |\mathbf{V}_k^c(\mathbf{X})| \\ - \frac{1}{S} \sum_{s=1}^{S} \left[\sum_{k=1}^{K} \log |\mathbf{V}_k^o(\mathbf{X}; s)^{\text{CPD}}| + \sum_{j=1}^{J} \log |\mathbf{V}_k^c(\mathbf{X}; s)^{\text{CPD}}| \right] . \tag{5.9}$$

The cost of computing Equation (5.9), assuming a constant number of iterations of EP until convergence, is in $\mathcal{O}((K + J)N^3 + (K + J)B^3)$, where N is the number of points observed so far, and B is the batch size. This cost is a consequence of the GP inference process and having to compute the determinant of matrices of size $B \times B$. Importantly, the gradients of Equation (5.9) w.r.t $\mathbf{X}$ can be computed using automatic differentiation tools as Autograd (Maclaurin et al., 2015). This is key to guarantee that the acquisition can be optimized using, *e.g.*, quasi-Newton methods (L-BFGS), to find $\mathbf{X}_{N+1}$. Appendix C has further details about the computation of the acquisition and its gradients. We implemented PPESMOC in the software for BO Spearmint.

We illustrate the PPESMOC's acquisition function in two scenarios of a 1-dimensional problem (for the sake of visualization) with batch size $B = 2$ in Fig. 5.1 (left) and (right). This problem has two objectives and two constraints. The observations obtained so far are displayed with a blue cross (previous evaluated batch). Each axis corresponds to the potential values (in the interval $[0, 1]$) for each one of the two points in the batch. Fig. 5.1 displays the contour curves of the acquisition function. We remark here some of its properties: (i) It is symmetric w.r.t the diagonal, meaning that the order of the points in the batch does not affect its value. (ii) In the neighborhood of the observed point the acquisition value is low, meaning that the acquisition favors unexplored regions. (iii) In the diagonal the acquisition takes lower values, meaning that it favors diversity in the batch. These are expected properties of a batch BO acquisition function. In these experiments the number of Monte Carlo samples S is set equal to 10. This is also the number of samples used for the GP hyper-parameters.

5.3.5 Quality of the Approximation to the Acquisition Function

As described previously, the acquisition function of the proposed method, PPESMOC, is intractable and needs to be approximated. The exact evaluation requires computing an expectation that has no closed form solution and computing the conditional predictive distribution of the probabilistic models given some Pareto set $\mathcal{X}^\star$. In Section 5.3.4 we propose to approximate these quantities using Monte Carlo samples and expectation

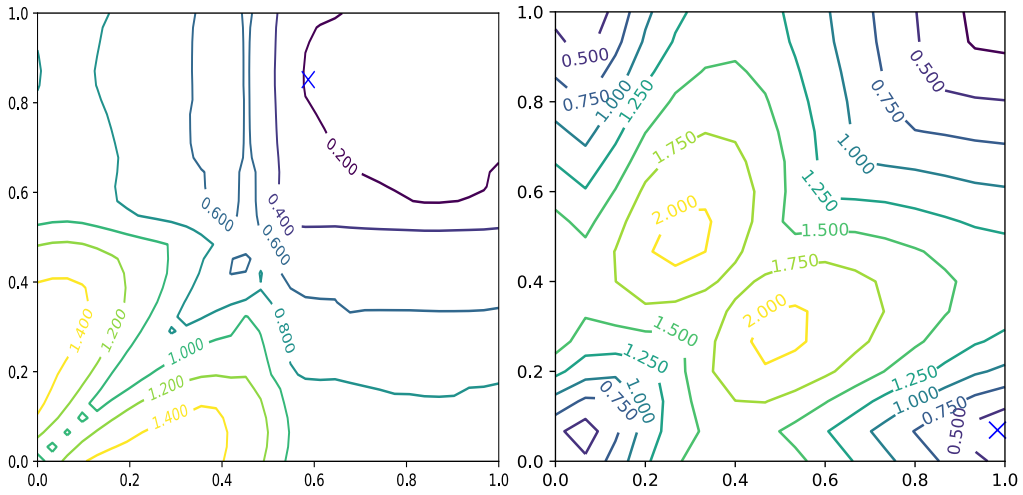

FIGURE 5.1: Acquisition function visualization. Each figure corresponds to a different repetition of the optimization problem. The batch size is $B = 2$. The problem is in one dimension. Each axis corresponds to different values for one of the 2 points in the batch, taking values in the interval $[0, 1]$. Blue crosses show already evaluated locations.

propagation, respectively. In this section we check the accuracy of this approximation to see if it resembles the actual acquisition function. For this, we consider a simple one dimensional problem with a batch of two points, two objectives and one constraint generated from a GP prior. In this simple setting, it is possible to compute a more accurate estimate of the acquisition function using a more expensive sampling technique, combined with a non-parametric estimator of entropy (Singh et al., 2003). More precisely, we discretize the input space and generate a sample of the Pareto set $\mathcal{X}^\star$ by optimizing a sample of the black-box functions. This sample is generated as in the PPESMOC approximation. We then generate samples of the black-box functions and keep only those that are compatible with $\mathcal{X}^\star$ being the solution to the optimization problem. This process is repeated $100,000$ times. Then, a non-parametric method is used to estimate the entropy of the predictive distribution at each possible batch of the input space before and after the conditioning. The difference in the entropy at each input location gives a more accurate estimate of the acquisition function of PPESMOC. Of course, this approach is too expensive to be used in practice for solving optimization problems.

We consider first a coupled evaluation setting. Figure 5.2 shows a comparison between the two estimates of the exact acquisition function. In both plots, the acquisition function values are displayed for all the batch combinations of the input space. The one described above (left, exact) and the one suggested in Section 5.3.4 (right, approximate). We observe that both estimates of the acquisition function take higher values in regions with high uncertainty and promising predictions. Similarly, both estimates take lower values in regions with low uncertainty. Importantly, both acquisition functions are pretty similar in the sense that they take high and low values in the same regions of the input space. Therefore, both acquisition functions are extremely correlated. This empirical result supports that the approximation proposed in this chapter is an accurate estimate of the actual acquisition function.

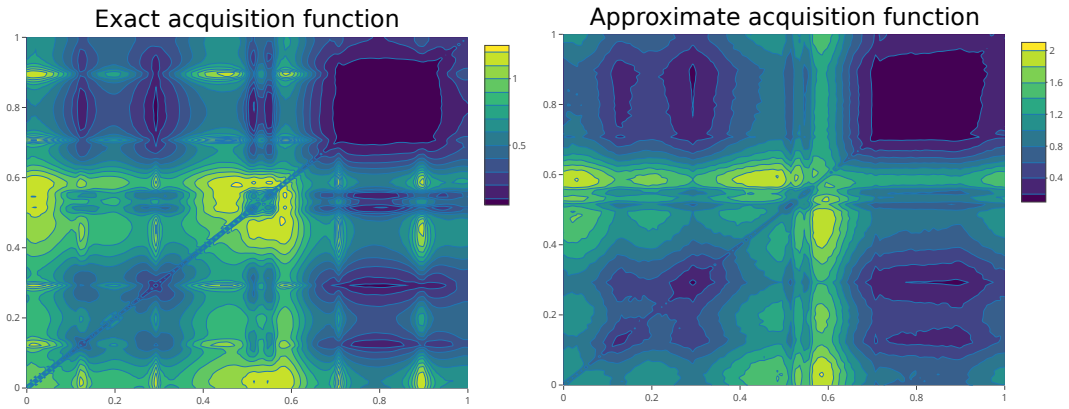

FIGURE 5.2: Acquisition function visualization. The batch size is $B = 2$. The problem is in one dimension. Each axis corresponds to different values for one of the 2 points in the batch, taking values in the interval $[0, 1]$.

5.4 Related Work

In this section we review other works from the literature that describe related batch BO methods and BO methods that can address multi-objective problems with constraints. Nevertheless, to our knowledge PPESMOC is the only batch BO method for constrained multi-objective problems. PPESMOC is related to parallel predictive entropy search (PPES) (Shah and Ghahramani, 2015). At each iteration, PPES also selects a batch of points maximizing the expected information gain about the global maximizer of the objective. The computations are also approximated using expectation propagation (EP). The main difference is that PPES is limited to single-objective and un-constrained optimization problems, unlike PPESMOC, which can address multiple objectives and several constraints. This is a non-trivial extension of PPES. In particular, several objectives and constraints require the use of several GPs, not only one. Furthermore, including constraints and several objectives leads to more complicated non-Gaussian factors that need to be approximated using EP. Therefore, the EP update operations of PPESMOC are significantly more complicated than those of PPES. Solving multi-objective problems is also more complicated than solving a single-objective problem. The solution of the later is the Pareto set, a set of potentially infinite size.

Other batch BO methods form the literature include local penalization. This is an heuristic that penalizes the acquisition function in each neighborhood of already selected points (González et al., 2016). The advantage of this method is that it can be used with arbitrary acquisition functions. A limitation, however, is that it requires to fix the amount of penalization, which depends on the Lipschitz constant of the objective. Such a constant needs to be estimated from data. Furthermore, González et al. (2016) only addresses unconstrained single-objective problems.

Hybrid batch Bayesian optimization dynamically switches, based on the current state, between sequential and batch evaluation policies with variable batch sizes (Azimi et al., 2012). This strategy uses expected improvement as the acquisition function. However, it can only address unconstrained single-objective problems. A batch BO approach can also be implemented via a multi-objective ensemble of multiple acquisition functions (Lyu et al., 2018). In each iteration, a multi-objective optimization of multiple acquisition functions is carried out. A sample of points from the Pareto set is then selected as

the batch of points to evaluate. Even though this strategy can address multi-objective problems, it cannot deal with constraints in the optimization process. Other strategies for batch BO that do not address the constrained multi-objective setting include (Daxberger and Low, 2017; Desautels et al., 2014; Gupta et al., 2018; Kathuria et al., 2016).

5.5 Experiments

We evaluate the performance of PPESMOC and compare results with two base-lines. Namely, a strategy that chooses at each iteration a random batch of points at which to evaluate the black-boxes. We also compare results with two parallel sequential methods (see Section 5.4) that use the acquisition function of PESMOC and BMOO, respectively. We refer to these methods as PS_PESMOC and PS_BMOO. In each experiment, we report average results and error bars across 100 repetitions. We measure the logarithm of the relative difference in absolute value of the hyper-volumes of the recommendation and the the problem's solution. The solution is found via exhaustive search in synthetic problems. In real-world problems we use the best recommendation obtained by any method. We use a Matérn covariance function for the GPs. The GP hyper-parameters are sampled from the posterior using 10 slice samples, as in (Snoek et al., 2012). The predictive distribution and acquisition functions are averaged over the samples. In PPESMOC, the number of samples S of $\mathcal{X}^\star$ is also set to 10 as it was done for PESMOC. In each method, at each iteration, we output a recommendation obtained by optimizing the GP means. For this, we use a grid of points. Finally, to recommend only feasible solutions with high probability, we perform the same operation as the one described in the previous chapter.

5.5.1 Synthetic Experiments

We compare the performance of each method when the objectives and the constraints are sampled from a GP prior. The problem considered has 2 objectives and 2 constraints in a 4-dimensional input space. We consider a noiseless and a noisy scenario. In this last case, the evaluations are contaminated with Gaussian noise with variance equal to 0.1. We report results for different batch sizes. Namely, $4, 8, 10$ and 20 points. We allow for 100 evaluations. In the case that the recommendation produced contains an infeasible point, we simply set the hyper-volume of the recommendation equal to zero. For the optimization of the PPESMOC and parallel sequencial acquisition functions, we select an initial random point from the grid and launch L-BFGS from it. If a high amount of computational resources are available, we suggest to compute PPESMOC and parallel sequencial acquisition functions in a grid and select the initial point for the L-BFGS optimization algorithm as the one that maximizes the respective acquisition function.

The results obtained are displayed in Figure 5.3 and Figure 5.4 for the noiseless and noisy case, respectively. We observe that PS_PESMOC and PPESMOC and better than the other methods. The random search strategy gives the worst results followed by PS_BMOO. The results for the noise and the noiseless scenario are similar. We can observe that, for every scenario, PPESMOC outperforms the other methods.

The disadvantage of PS_PESMOC (and also PS_BMOO) is that it has a bigger computational cost with respect to the batch size B than PPESMOC. More precisely, it requires updating the GP models and optimizing the acquisition B times. PPESMOC is computationally cheaper. We add empirical evidence for this claim by showing, in Table 5.1, the median of the time used by each method to determine the next batch of points to evaluate. PPESMOC consumes a computational time that grows less with respect to

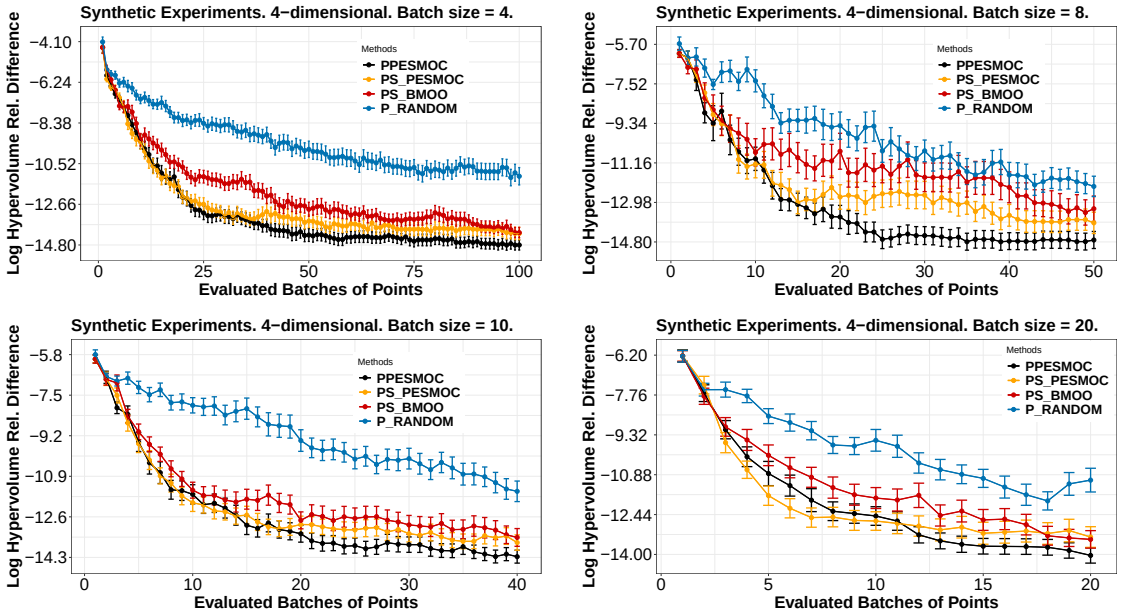

FIGURE 5.3: Average results for the synthetic experiments in the noiseless evaluation setting.

TABLE 5.1: Median time in seconds to choose the next batch of points to evaluate of PPESMOC and the parallel sequential approaches.

Method	B=4	B=8	B=10	B=20	B=50
PPESMOC	697.3±27.5	913.9±28.1	960.7±27.8	1044.8±30.7	**1273.9±30.1**
PS_PESMOC	**191.3±7.0**	**346.2±6.1**	**406.2±6.5**	**799.8±26.6**	1960.3±31.6
PS_BMOO	379.4±12.3	551.7±22.8	594.2±19.9	895.9±27.9	1874.2±42.4

the batch size B than parallel sequential approaches. By contrast, PS_PESMOC, as well as PS_BMOO, requires, on average, more and more computation time as the batch size B increases.

5.5.2 Benchmark Experiments

We carry out extra experiments in which the black-boxes are not sampled from a GP. For this, we consider 6 classical constrained multi-objective optimization problems (Chafekar et al., 2003). We consider two scenarios. A noiseless scenario and a noisy scenario, where the black-box evaluations are contaminated with additive Gaussian noise. The variance of the noise is set to 1% of the range of potential values of the corresponding black-box. The batch size considered is $B = 4$. The average results obtained in these experiments, for each method and each scenario, are displayed in Figure 5.5, that shows plots of the noiseless case, and Figure 5.6, showing the noisy setting. We observe that, most of the times, PPESMOC outperforms or performs similarly to the other methods in every scenario, noiseless and noisy, as it was expected.

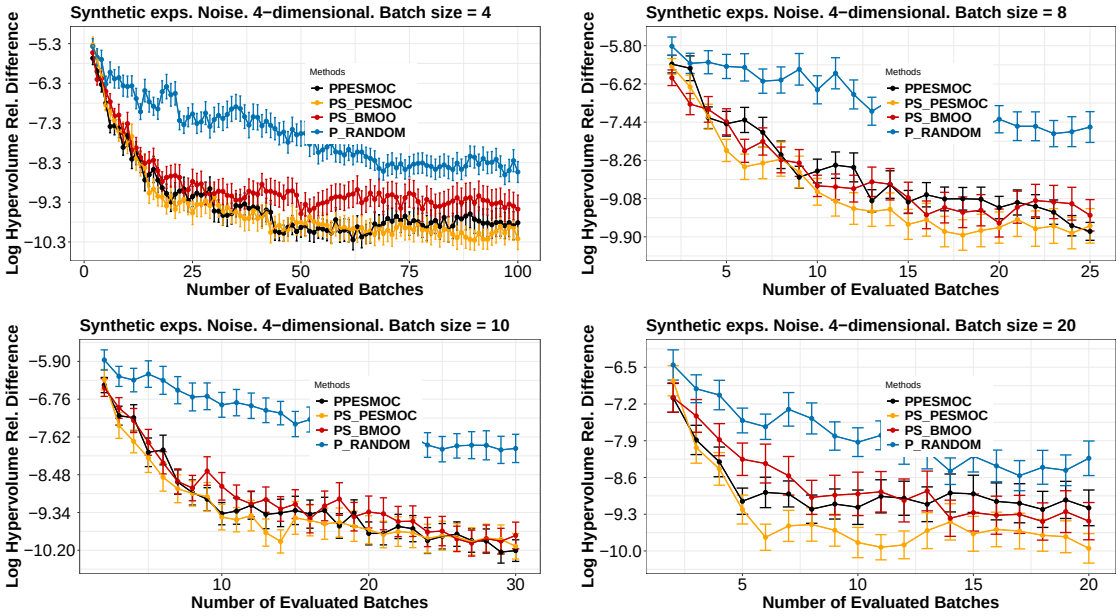

FIGURE 5.4: Average results for the synthetic experiments in the noisy evaluation setting.

5.5.3 Real-world Experiments

We compare each method in the task of finding an optimal ensemble gradient-boosting ensemble. We consider two objectives: the prediction error of the ensemble and its size. These objectives are conflictive since smaller ensembles will have in general higher error rates and the other way around. We introduce as a constraint of the problem, that the average speed up factor of the classification process given by a dynamic ensemble pruning technique is at least 25% as we did in the previous chapter. We have carefully chosen this value to guarantee that the constraint is active at the optimal solution. The dataset considered is the German credit dataset extracted from the UCI repository (Dua and Graff, 2017). This is a binary classification dataset with $1,000$ instances and 20 attributes. The prediction error is measured using 10-fold-cross validation, repeated 5 times. The ensemble size is the logarithm of the sum of the total number of nodes in the trees of the ensemble.

The adjustable parameters of the ensemble are: the number of trees (between 1 and $1,000$), the number of random attributes considered for split in each tree (between 1 and 20), the minimum number of samples required to split a node (between 2 and 200), the fraction of randomly selected training data used to build each tree (between 0.5 and 1.0), and the fraction of training instances whose labels are changed after the sub-sampling process (between 0.0 and 0.7).

Table 5.2 shows the average hyper-volume obtained in this task, for each method, after 25 and 50 evaluations using a batch size of 4. Figure 5.7 (left) shows also the average Pareto front obtained by each method. The Pareto front is simply given by the objective values associated to the recommendation made by each method. The higher the volume of points that is above this set of points in the objective values the better the performance of a method. We observe that PPESMOC slightly outperforms the other approaches. As described in PESMOC, we also evaluate each method on the task of

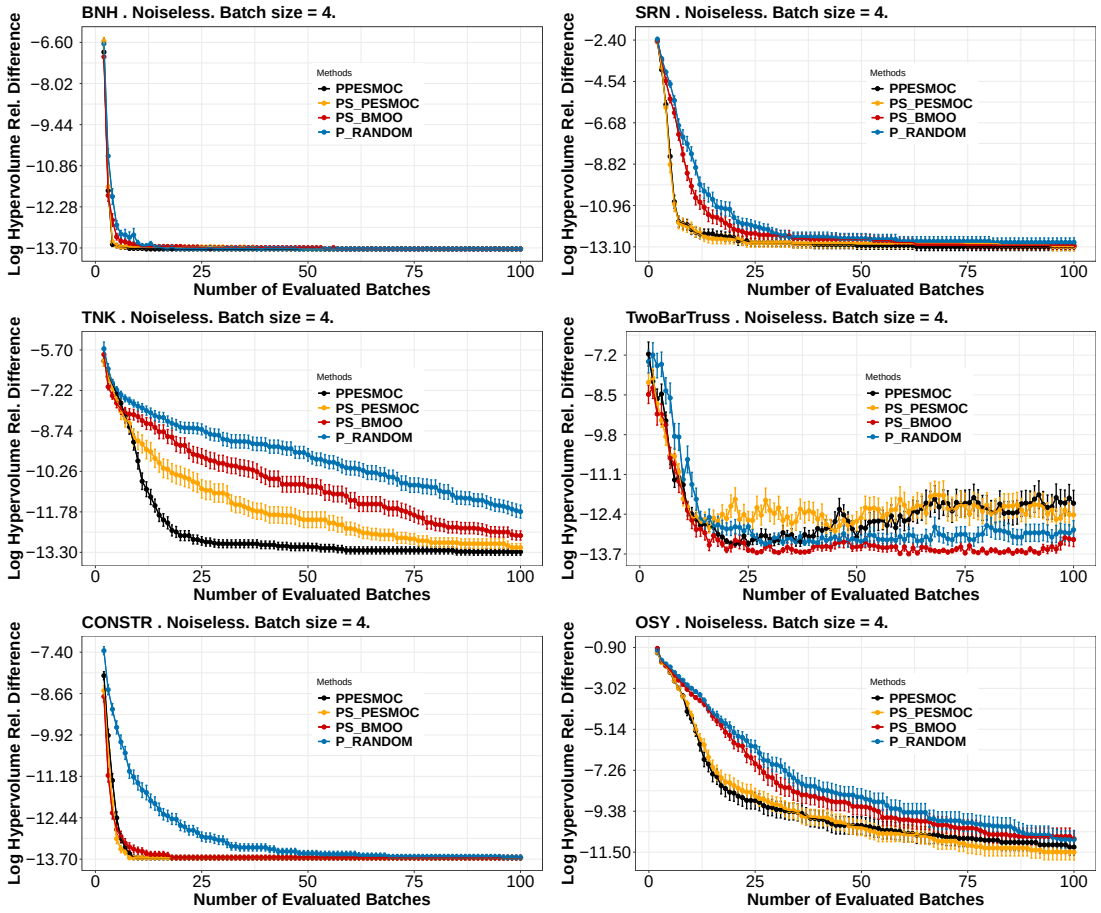

FIGURE 5.5: Logarithm of the relative difference between the hyper-volume of the recommendation obtained by each method and the hyper-volume of the actual solution. We report results after each evaluation of the black-box functions. Benchmark functions corrupted by noise.

PPESMOC	PS_PESMOC	PS_BMOO	P_RANDOM
0.231±0.005	0.228±0.005	0.228±0.008	0.218±0.006

TABLE 5.2: Average hyper-volume in the task of finding an optimal ensemble of trees.

finding an optimal deep neural network (DNN) on the MNIST dataset (LeCun et al., 2010). The objectives are the prediction error of the DNN on a validation dataset of 10,000 instances (extracted from the original training set) and the time that such a DNN will take for making predictions. These are conflictive objectives in the sense that minimizing the prediction error will often lead to bigger DNN with a bigger prediction times. We are also interested in codifying such a DNN into a chip. Thus, we constrain the problem by enforcing that the area of the resulting DNN, after being codified into a chip, is below 1 mm^2. We have carefully chosen this value to guarantee that the constraint is active at the optimal solution. To measure the chip area we use the Aladdin simulator, which given a computer program describing the operations of the DNN, outputs an estimate of the area of a chip implementing those operations (Shao et al., 2014). To train the DNN we use the Keras library. Prediction time is normalized by the smallest possible prediction time, which corresponds to a DNN of a single layer with 5 hidden units.

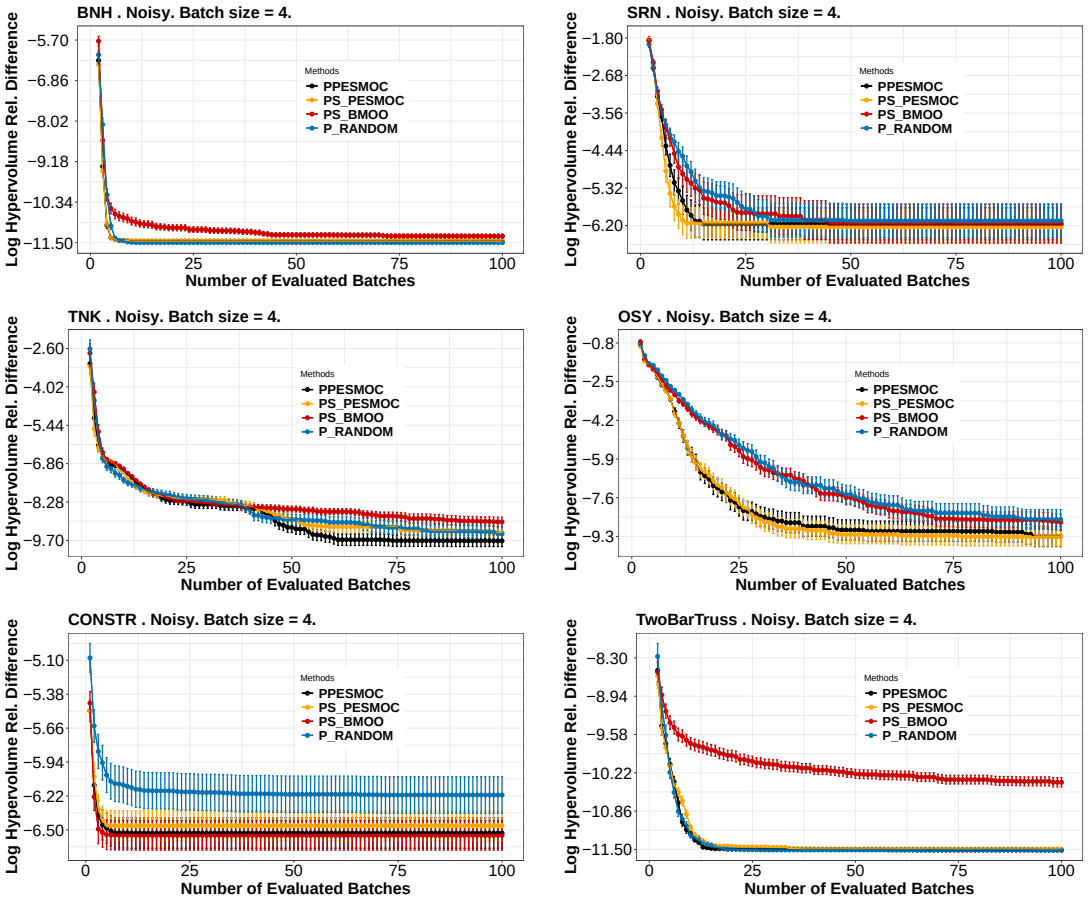

FIGURE 5.6: Average results for the problems BNH, SRN, TNK and OSY, CONSTR and TwoBar Truss. Noisy setting.

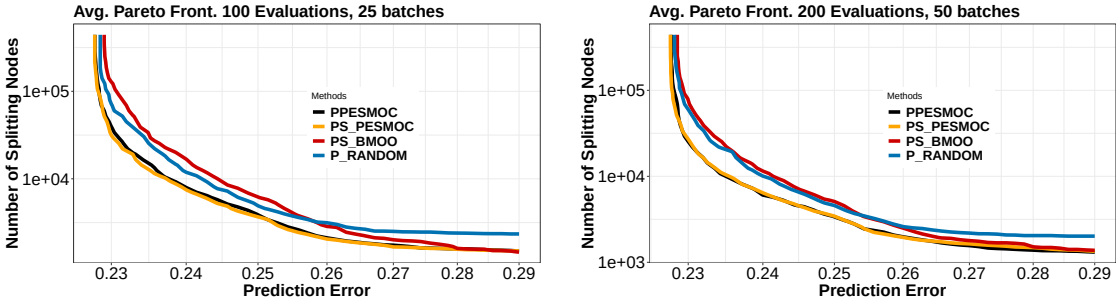

FIGURE 5.7: Average Pareto front in the task of finding an optimal ensemble for 100 (left) and 200 evaluations (right).

The input parameters to be optimized are: The logarithm of the ℓ_1 and ℓ_2 weight regularizers; the dropout probability; the logarithm of the initial learning rate; the number of hidden units per layer; and the number of hidden layers. We have also considered two variables that have an impact in the hardware implementation of the DNN. Namely, the logarithm (in base 2) of the array partition factor and the loop unrolling factor.

We report the performance after 15 and 25 evaluations using a batch size $B = 4$. The DNN is trained using ADAM with the default parameters. The loss function is the

TABLE 5.3: Hypervolume dominated by the proposed methods in the experiment. Larger hypervolume means better quality.

PPESMOC	PS_PESMOC	PS_BMOO	P_RANDOM
25.55±0.03	25.44±0.05	25.01±0.05	24.61±0.03

cross-entropy. The last layer of the DNN contains 10 units and a soft-max activation function. All other layers use Re-Lu as the activation function. Finally, each DNN is trained during a total of 150 epochs using mini-batches of size $4,000$.

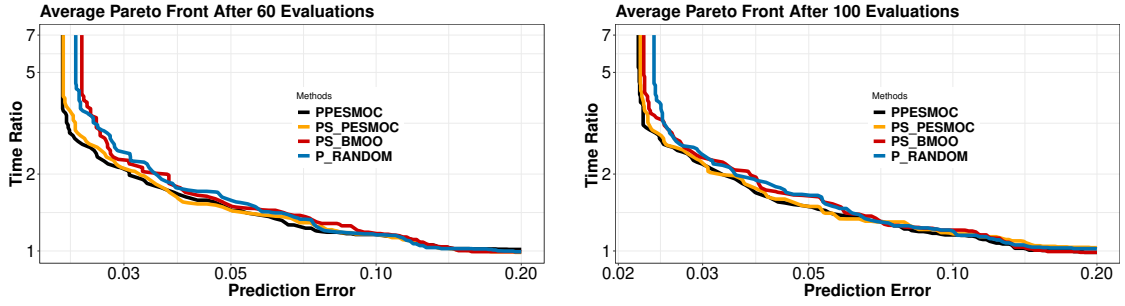

FIGURE 5.8: Average Pareto front in the task of finding an optimal neural network for 60 (left) and 100 evaluations (right).

The average Pareto front obtained by each method is shown in Figure 5.8. Table 5.3 shows the average hyper-volume of each method. Here, PPESMOC outperforms by little PS_PESMOC. PS_BMOO gives worse results than PS_PESMOC. The random search strategy is the worst performing method. We can see in the frontier how the difference between the PPESMOC and PS_PESMOC method is slight. PS_PESMOC method suggests configurations with less prediction error and PPESMOC suggests neural networks with less energy consumption.

5.6 Conclusions

In this chapter we have described PPESMOC, the first method to address batch Bayesian optimization problems with several objectives and constraints. More precisely, PPESMOC suggests, at each iteration, at batch of points at which the objectives and constraints should be evaluated in parallel. We have compared the performance of PPESMOC on several optimization problems, including synthetic, benchmark and real-world problems. Furthermore, we have compared results with two simple base-lines. Namely, a random exploration strategy and two methods derived from the literature about sequential Bayesian optimization, PS_PESMOC and PS_BMOO. We have observed that PPESMOC performs well in general, giving similar and sometimes better results than PS_PESMOC and PS_BMOO. The main advantage is, however, that the PPESMOC scales much better with respect to the batch size. Unlike PPESMOC, the sequential strategies require repeating an iterative process as many times as the batch size. This process includes hallucinating observations, re-fitting the underlying GP models, and optimizing a sequential acquisition function. This leads to a prohibitive computational cost for large batch sizes.

Chapter 6

Dealing With Categorical And Integer-Valued Variables In Bayesian Optimization With Gaussian Processes

In the context of Bayesian optimization, GPs assume input variables taking values in the real line. When this is not the case, for example when some of the input variables take categorical or integer values, the practitioner has to introduce extra approximations to tackle these variables. Consider a BO optimization problem with non-real input variables and a suggested input location at which to evaluate the objective taking values in the real line. Before doing the evaluation of the objective, a common approach is to use a one hot encoding approximation for categorical variables, or to round to the closest integer, in the case of integer-valued variables. We show that this methodology can lead to problems in the Bayesian optimization process and describe a more principled approach to account for input variables that are categorical or integer-valued. We illustrate in both synthetic and real experiments the utility of our approach, which significantly improves the results of standard BO methods using Gaussian processes on problems with categorical or integer-valued variables.

6.1 Introduction

Many problems involve the optimization of a function with no analytical form. An example is tuning the parameters of the control system of a robot to maximize locomotion speed (Lizotte et al., 2007). There is no closed-form expression to describe the function that, given specific values for these parameters, returns an estimate of the corresponding speed. A practical experiment with the robot or a computer simulation will have to be carried out for this purpose. Moreover, the time required for such an evaluation can be high, which means that in practice one can only perform a few evaluations of the objective. Importantly, these evaluations may be noisy and hence different for the same input parameters. The noise can simply be related to the environmental conditions in which the robot's experiment is performed. When a function has the characteristics described it is called a black-box function $f(\mathbf{x})$ over some bounded domain $\mathcal{X} \in \mathbb{R}^d$, where d is the dimensionality of the input space. Examples of problems involving the optimization of black-box functions include automatic tuning machine learning hyper-parameters (Snoek,

2013), finding optimal control parameters in robotics (Lizotte et al., 2007), optimal weather sensor placement (Garnett et al., 2010) or the optimization search strategies (Cornejo-Bueno et al., 2018).

Bayesian Optimization (BO) methods are popular for optimizing black-box functions (Shahriari et al., 2015) with the characteristics described (Mockus et al., 1978). More formally, BO methods optimize a real-valued function $f(\mathbf{x})$ over some bounded domain $\mathcal{X}$. The objective function is assumed to lack an analytical expression (which prevents any gradient computation), to be very expensive to evaluate, and the evaluations are assumed to be noisy (*i.e.*, rather than observing $f(\mathbf{x})$ we observe $y = f(\mathbf{x}) + \epsilon$, with ϵ some additive noise). The goal of BO methods is to reduce the number of objective evaluations that need to be performed to solve the optimization problem. For this, they iteratively suggest, in a careful and intelligent way, an input location at which the objective that is being optimized should be evaluated each time. For this, at each iteration $N = 1, 2, 3, \ldots$ of the optimization process, BO methods fit a probabilistic model, typically a Gaussian process (GP) (Rasmussen, 2003), to the collected observations of the objective $\{y_i\}_{i=1}^{N}$. The uncertainty about the potential values of the objective are provided by the predictive distribution of the GP. This uncertainty is used to generate an acquisition function $\alpha(\cdot)$, whose value, at each input location, indicates the expected utility of evaluating $f(\cdot)$ there. The next point $\mathbf{x}_{N+1}$ at which to evaluate $f(\cdot)$ is the one that maximizes $\alpha(\cdot)$. After collecting this observation, the process is repeated. When enough data has been collected, the GP predictive mean value for $f(\cdot)$ can be optimized to find the solution of the problem.

The key to BO success is that evaluating the acquisition function $\alpha(\cdot)$ is very cheap compared to the evaluation of the objective $f(\cdot)$. This is so because the acquisition function only depends on the GP predictive distribution for $f(\cdot)$ at a candidate point $\mathbf{x}$. Thus, $\alpha(\cdot)$ can be maximized with very little cost. BO methods hence spend a small amount of time thinking very carefully where to evaluate next the objective function with the aim of finding its optimum with the smallest number of evaluations. This is a useful strategy when the objective function is very expensive to evaluate and it can save a lot of computational time.

A problem, however, of GPs is that these probabilistic models assume that the input variables take real-values. If this is not the case and, for example, some of the variables can take categorical or integer values, extra approximations in the BO method have to be introduced to address this issue. In the case of integer-valued variables, the approximations often involve simply doing some rounding to the closest integer after optimizing the acquisition function. In the case of a categorical variable, one simply uses a one-hot encoding. This involves adding as many extra input variables as different categories this variable can take. Then, after optimizing the acquisition function, the extra variable that is largest is set equal to one and all the others equal to zero. This is the approach followed, for example, in the popular software for BO Spearmint (`https://github.com/HIPS/Spearmint`).

We show here that the approaches described for handling categorical and integer-valued variables may make the BO method fail. These problems can be overcome by doing the rounding (to the closest integer or the corresponding one-hot encoding) inside the wrapper that evaluates the objective. Nevertheless, this will make the objective constant in some regions of the input space, *i.e.*, those rounded to the same integer value, in the case of integer-valued variables, or those that lead to the same one-hot encoding, in the case of categorical variables. This constant behavior of the objective will be ignored by the GP model. To overcome this, we introduce a transformation

of the input variables that will lead to an alternative covariance function for the GP model. With this covariance function, the GP will correctly describe the objective as constant in particular regions of the input space, leading to better modeling results and, in consequence, to better optimization results.

Practical examples of optimization problems involving a mix between real, categorical and integer-valued variables include finding the optimal hyper-parameters of a machine learning system (Snoek, 2013). Specifically, in a deep neural network, we may want to adjust the learning rate, the number of layers and the activation function. These two last variables can only take integer and categorical values, respectively, while the learning rate can take real values. Similarly, in a gradient boosting ensemble of decision trees (Friedman, 2001) we may try to adjust the learning rate and the maximum depth of the trees, which can only take integer values. Our experiments show that the proposed approach for dealing with a mix of real, categorical, and integer-valued variables in BO methods leads to improved results over standard techniques and other alternatives from the literature.

The rest of the chapter is organized as follows: Section 6.2 gives a short introduction to BO and Gaussian processes. Section 6.3 describes the proposed approach to deal with categorical and integer-valued variables in BO methods using GPs. Section 6.4 reviews related methods from the literature. Section 6.5 describes synthetic and real-world experiments that show that the proposed approach has advantages over standard methods for BO and related techniques. Finally, Section 6.6 gives the conclusions of this chapter.

6.2 Background on Gaussian Processes and Bayesian Optimization

BO methods rely on a probabilistic model for the black-box function being optimized. This model must generate a predictive distribution for the potential values of the objective at each point of the input space. This predictive distribution is used to guide the search, by focusing only on regions of the input space that are expected to deliver the most information about the solution of the optimization problem. Most commonly used models are Gaussian processes (GPs) (Rasmussen, 2003), Random Forests implemented in Auto-WEKA (Thornton et al., 2013), Student's-T processes (Shah et al., 2014) or deep neural networks (Snoek et al., 2015). In this chapter, we will focus on the use of GP, but the same ideas can be implemented in a Student's-T process.

A GP is defined as a prior distribution over functions. When using a GP as the underlying model, the assumption made is that the black-box function $f(\cdot)$ to be optimized has been generated from such a prior distribution, which is characterized by a zero mean and a covariance function $k(\mathbf{x}, \mathbf{x}')$. That is $f(\cdot) \sim \mathcal{GP}(0, k(\cdot, \cdot))$. The particular characteristics of $f(\cdot)$, *e.g.*, smoothness, additive noise, amplitude, etc., are specified by the covariance function $k(\mathbf{x}, \mathbf{x}')$ which computes the covariance between $f(\mathbf{x})$ and $f(\mathbf{x}')$. A typical covariance function employed in the context of BO is the Matérn function, in which the ν parameter is set equal to 3/2 (Snoek, 2013). This covariance function is:

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2(1 + \frac{\sqrt{3}r}{\ell}) \exp(-\frac{\sqrt{3}r}{\ell}), \tag{6.1}$$

where r is the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}'$. Namely, $|\mathbf{x} - \mathbf{x}'|$. Note that $k(\cdot, \cdot)$ only depends on r. This particular covariance function and others that share this property are known as *radial basis functions* (RBFs). ℓ is simply a hyper-parameter known as length-scale, which controls the smoothness of the GP. Most of the times a different length scale ℓ_j is used for each dimension j. σ^2 is the amplitude parameter, which controls the range of variability of the GP samples. Finally, ν is a hyper-parameter related to the number of times that the GP samples can be differentiated. Another popular covariance function is the squared exponential. In this case $k(\cdot, \cdot)$ is given by:

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{r^2}{2\ell^2}\right). \tag{6.2}$$

Assume that we have already evaluated the objective at N input locations. Let the corresponding data be summarized as $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $y_i = f(\mathbf{x}_i) + \epsilon_i$, with ϵ_i some additive Gaussian noise with variance σ_0^2. A GP provides a predictive distribution for the potential values of $f(\cdot)$ at different regions of the input space. This distribution is Gaussian and is characterized by a mean $\mu(\mathbf{x})$ and a variance $\sigma^2(\mathbf{x})$. Namely, $p(f(\mathbf{x}^\star)|\mathbf{y}) = \mathcal{N}(f(\mathbf{x}^\star)|m(\mathbf{x}^\star), \sigma^2(\mathbf{x}^\star))$, where the mean and variance are respectively given by:

$$\mu(\mathbf{x}) = \mathbf{k}_*^T (\mathbf{K} + \sigma_0^2 \mathbf{I})^{-1} \mathbf{y}\,, \tag{6.3}$$

$$\sigma^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}_*^T (\mathbf{K} + \sigma_0^2 \mathbf{I})^{-1} \mathbf{k}_*\,. \tag{6.4}$$

In the previous expression $\mathbf{y} = (y_1, \ldots, y_{t-1})^{\mathrm{N}}$ is a vector with the objective evaluations observed so far; $\mathbf{k}_*$ is a vector with the prior covariances between $f(\mathbf{x})$ and each y_i; σ_0^2 is the variance of the additive Gaussian noise; $\mathbf{K}$ is a matrix with the prior covariances among each $f(\mathbf{x}_i)$, for $i = 1, \ldots, N$; and $k(\mathbf{x}, \mathbf{x})$ is the prior variance at the candidate location $\mathbf{x}$. All these quantities are simply obtained by evaluating the covariance function $k(\cdot, \cdot)$ on the corresponding input values. See Rasmussen (2003) for further details.

BO methods use the previous predictive distribution to determine at which point $\mathbf{x}_{N+1}$ the objective function has to be evaluated. Once this new observation has been collected, the GP model is updated with the new data and the process repeats. After collecting enough data like this, the GP posterior mean given by (6.3) can be optimized to provide an estimate of the solution of the optimization problem. Notwithstanding, a GP has some hyper-parameters that need to be adjusted during the fitting process. These include the variance of the additive Gaussian noise σ_0^2, but also any potential hyper-parameter of the covariance function $k(\cdot, \cdot)$. These can be, *e.g.*, the amplitude parameter and the length-scales (Rasmussen, 2003). Instead of finding point estimates for these hyper-parameters, an approach that has shown good empirical results is to compute an approximate posterior distribution for them using slice sampling (Snoek, 2013). The previous Gaussian predictive distribution described in (6.3) and (6.4) is then simply averaged over the hyper-parameter samples to obtain the final predictive distribution of the probabilistic model.

The key for BO success is found in the acquisition function $\alpha(\cdot)$. This function uses the predictive distribution given by the GP to compute the expected utility of performing an evaluation of the objective at each input location. The next point at which the objective has to be evaluated is simply $\mathbf{x}_{N+1} = \arg\max_{\mathbf{x}} \alpha(\mathbf{x})$. Because this function only depends on the predictive distribution given by the GP and not on the actual objective $f(\cdot)$, the maximization of $\alpha(\cdot)$ is very cheap. A popular acquisition function is expected improvement (EI) (Jones et al., 1998). EI is given by the expected value of the

utility function $u(y) = \max(0, \nu - y)$ under the GP predictive distribution for y, where $\nu = \min(\{y_i\}_{i=1}^N)$ is the best value observed so far, assuming minimization. Therefore, EI measures in expectation how much we will improve on the current best found solution by performing an evaluation at each candidate point. The EI acquisition function is given by the following expression:

$$\alpha(\mathbf{x}) = \sigma(\mathbf{x})(\gamma(\mathbf{x})\Phi(\gamma(\mathbf{x}) + \phi(\gamma(\mathbf{x}))\,, \tag{6.5}$$

where $\gamma(\mathbf{x}) = (\nu - \mu(\mathbf{x}))/\sigma(\mathbf{x})$ and $\Phi(\cdot)$ and $\phi(\cdot)$ are respectively the c.d.f. and p.d.f. of a standard Gaussian distribution.

Another popular acquisition function for BO is Predictive Entropy Search (PES) (Hernández-Lobato et al., 2014). PES is an information-theoretic method that chooses the next input location $\mathbf{x}_{N+1}$ at which the objective function has to be evaluated as the one that maximizes the information about the global maximum $\mathbf{x}^\star$ of the optimization problem. The information about this maximum is given in terms of the differential entropy of the random variable $\mathbf{x}^\star$. This random variable is characterized by the corresponding posterior distribution $p(\mathbf{x}^\star|\mathcal{D}_N)$. PES simply chooses $\mathbf{x}_{N+1}$ as the point that maximizes the expected reduction in the differential entropy of $\mathbf{x}^\star$. The PES acquisition function is:

$$\alpha(\mathbf{x}) = H[p(\mathbf{x}^\star|\mathcal{D}_N)] - \mathbb{E}_y[H[p(\mathbf{x}^\star|\mathcal{D}_N \cup (\mathbf{x}, y))]\,, \tag{6.6}$$

where the expectation w.r.t. y is given by the predictive distribution of the GP at $\mathbf{x}$.

A problem is, however, that evaluating (6.6) in closed form is intractable. This expression has to be approximated in practice. In Hernández-Lobato et al. (2014), the authors use the fact that the previous expression is simply the mutual information between $\mathbf{x}^\star$ and y, $I(\mathbf{x}^\star; y)$, which is symmetric. Therefore one can swap the roles of y and $\mathbf{x}^\star$ in (6.6). This greatly simplifies the evaluation of the acquisition function and an efficient approximation based on the expectation propagation algorithm is possible. PES has been compared to other acquisition functions showing improved optimization results. In particular, it shows a better trade-off between exploration and exploitation than EI.

6.3 Dealing with Categorical and Integer-valued Variables

In the framework described, the objective function $f(\cdot)$ is assumed to have input variables taking values on the real line. This is so, because in a GP the variables introduced in the covariance function $k(\cdot, \cdot)$ are assumed to be real. A problem may arise when some of the input variables can only take values in a closed subset of a discrete set, such as the integers, or when some of the input variables are categorical. In this second case a typical approach is to use a one-hot encoding of categorical variables. That is, the number of input dimensions is extended by adding extra variables, one per potential category. The only valid configurations are those in which one of the extra variables takes value one (*i.e.*, the extra variable corresponding to the active category), and all other extra variables take value zero. For example, consider a categorical input dimension x_j taking values in the set $\mathbb{C} = \{\textit{red}, \textit{green}, \textit{blue}\}$. We will replace dimension j in $\mathbf{x}$ with three extra dimensional variables taking values $(1, 0, 0), (0, 1, 0)$ and $(0, 0, 1)$, for each value in $\mathbb{C}$, respectively.

When not all input variables take real values, a standard GP will ignore that only some input variables configurations are valid and will place some probability mass on points at which $f(\cdot)$ cannot be evaluated. These incorrect modeling assumptions about $f(\cdot)$

may have a negative impact on the optimization process. Furthermore, the optimization of $\alpha(\cdot)$ will give candidate points $\mathbf{x}_{N+1}$ in which integer-valued or categorical variables will be assigned invalid values. In practice, some mechanism must be implemented to transform real values into integer or categorical values before the evaluation can take place. Importantly, if this is not done with care, some problems may appear.

6.3.1 Naive and Basic Approaches

As described before, if the problem of interest considers some categorical or integer-valued variables, $f(\cdot)$ cannot be evaluated at all potential input locations. It can only be evaluated at those input locations that are compatible with the categorical or integer-valued variables. A naive approach to account for this is to (i) optimize $\alpha(\cdot)$ assuming all variables take values in the real line, and (ii) replace all the values for the integer-valued variables by the closest integer, and replace all categorical variables with the corresponding one-hot encoding in which only one of the extra input variables takes value one and all the others take value zero. In this second case, the active variable is simply chosen as the one with the highest value among the extra input variables. More precisely, let $\mathbb{Q}_k$ be the set of extra input dimensions of $\mathbf{x}$ corresponding to the categorical input variable k and let $j \in \mathbb{Q}_k$. Then, we simply set $x_j = 1$ if $x_j > x_i \; \forall i \in \mathbb{Q}_k$ and $i \neq j$. Otherwise, $x_j = 0$. This is the approach followed by the popular software for BO Spearmint (`https://github.com/HIPS/Spearmint`).

The first row of Figure 6.1 shows, for an integer-valued input variable, that the naive approach just described can lead to a mismatch between the points in which the acquisition takes high values, and where the actual evaluation is performed. Importantly, this can produce situations in which the BO method always evaluates the objective at a point where it has already been evaluated. This may happen simply because the next and following evaluations are performed at different input locations from the one maximizing the acquisition function. More precisely, since the evaluation is performed at a different point, it may not reduce at all the uncertainty about the potential values of the objective at the point maximizing the acquisition function. Of course, in the case of categorical input variables this mismatch between the maximizer of the acquisition function and the actual point at which the objective is evaluated will also be a problem. For this reason, we discourage the use of this approach.

The previous problem can be easily solved. In the case of integer-valued variables, one can simply do the rounding to the closest integer value inside the wrapper that evaluates the objective. In the case of categorical variables, a similar approach can be followed inside the wrapper using one-hot encoding. Namely, (i) look at which extra input variable has the largest value, (ii) set that input variable equal to one, and (iii) set all other extra input variables equal to zero. This basic approach is shown in the second row of Figure 6.1 for the integer-valued case. Here, the points at which the acquisition takes high values and the points at which the objective is evaluated coincide. Thus, the BO method will tend to always perform evaluations at different input locations, as expected. This will avoid the problem described before, in which the BO method may get stuck. The problem is, however, that the actual objective is constant in the intervals that are rounded to the same integer value. This constant behavior is ignored by the GP, which can lead to sub-optimal optimization results. The same behavior is expected in the case of categorical input variables.

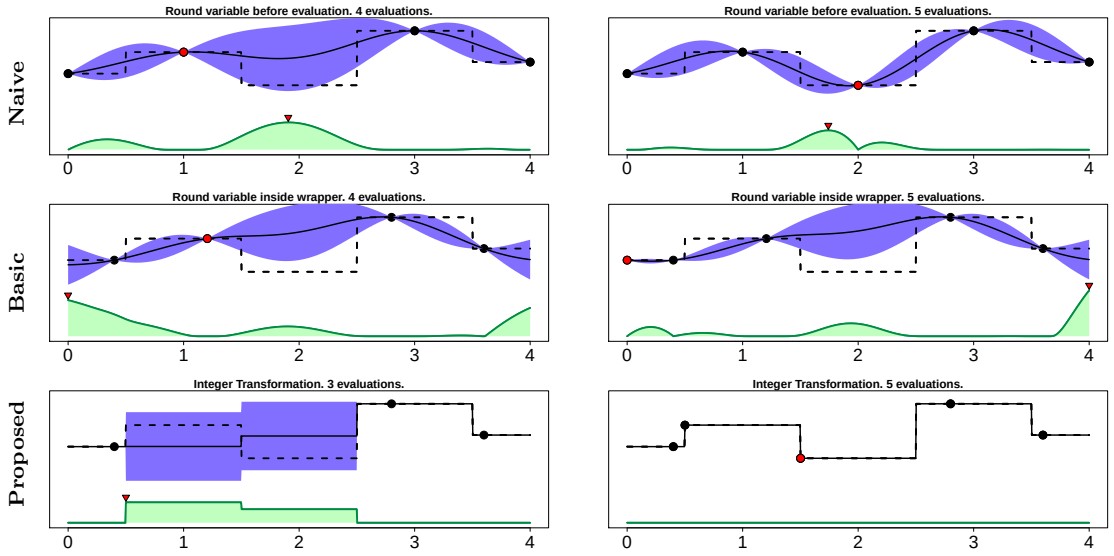

FIGURE 6.1: Different methods for dealing with integer-valued variables. At the top of each image, we show a GP fit to the data (posterior mean and 1-std confidence interval, in purple) that models a 1-dimensional objective taking values in the set $\{0, 1, 2, 3, 4\}$ (dashed line). To display the objective we have rounded the real values at which to do the evaluation to the closest integer. Below the GP fit, it is shown the acquisition function whose maximum is the recommendation for the next new evaluation. Each column shows similar figures before and after evaluating a new point, respectively. The proposed approach leads to no uncertainty about the objective after two evaluations. Best seen in color.

6.3.2 Proposed Approach

We propose here a method to alleviate the problems of the basic approach described in Section 6.3. For this, we consider that the objective should be constant in those regions of the input space that lead to the same input variable configuration on which the actual objective has to be evaluated. This property can be easily introduced into the GP by modifying the covariance function $k(\cdot, \cdot)$. Covariance functions are often stationary and only depend on the distance between the input points (Rasmussen, 2003). If the distance between two points is zero, the values of the function at both points will be the same (the correlation is equal to one). Based on this fact, we suggest to transform the input points to $k(\cdot, \cdot)$, obtaining an alternative covariance function $k'(\cdot, \cdot)$:

$$k'(\mathbf{x}_i, \mathbf{x}_j) = k(T(\mathbf{x}_i), T(\mathbf{x}_j)) \,, \tag{6.7}$$

where $T(\mathbf{x})$ is a transformation in which all non real input variables of $f(\cdot)$ in $\mathbf{x}$ are modified as follows:

- The input variables corresponding to an integer-valued input variable are rounded to the closest integer value.
- All extra input variables corresponding to the same categorical input variable are assigned zero value unless they take the largest value among the corresponding group of extra variables. If they take the largest value, they are assigned value one.

Essentially $T(\cdot)$ does the same transformation on $\mathbf{x}$ as the one described in Section 6.3.1 for the basic approach inside the wrapper that evaluates the objective. Importantly,

however, this transformation takes place in the covariance function of the GPs, which will allow for a better modeling of the objective.

The beneficial properties of $k'(\cdot,\cdot)$ when used for BO are illustrated in the third row of Figure 6.1 for the case of an integer-valued input variable. We can see that the GP model correctly identifies that the objective function is constant inside intervals of real values that are rounded to the same integer. The uncertainty is also the same in those intervals, and this is reflected in the acquisition function. Furthermore, after performing a single measurement in each interval, the uncertainty about $f(\cdot)$ goes to zero. This better modeling of the objective is expected to be reflected in a better performance of the optimization process. The same behavior is expected in the case of categorical variables.

In the case of integer-valued variables, the transformation $T(\mathbf{x})$ will round all integer-valued variables values in $\mathbb{R}$ to the closest integer $k \in \mathbb{Z}$. The set of integer values, $\mathbb{Z}$, has a notion of order. That is, for all $z \in \mathbb{Z}$, we can define operators of order that involve two values: $<, >, \leq$ and $\geq$, such that $z_i < z_j$, $z_j > z_i$, $z_i \leq z_j$ and $z_j \geq z_i$, having that $z_i, z_j \in \mathbb{Z}$. This order will be preserved by the resulting transformation. More precisely, assume an integer input variable and that $T(\mathbf{x})$ and $T(\mathbf{x}')$ only differ in the value of such integer input variable. The prior covariance between $f(\mathbf{x})$ and $f(\mathbf{x}')$ under $k(T(\mathbf{x}), T(\mathbf{x}'))$ will be higher the closer the corresponding integer values of $T(\mathbf{x})$ and $T(\mathbf{x}')$ are one from another. Therefore, the GP will be able to exploit the smoothness in the objective $f(\cdot)$ when solving the optimization problem.

In the case of categorical variables (*e.g.*, variables that can take values such as *red*, *green*, *blue*) there is no notion of order. That is, the operators $<, >, \geq$ and $\leq$ have no meaning nor purpose. One cannot compare two different values c_1, c_2 of any categorical-valued set $\mathbb{C}$ according to these operators. However, what does exist in a categorical set is a notion of equality or difference, given by the operators $=, \neq$. The proposed transformation is able to preserve this notion of no order and notion of equal or different. More precisely, assume a single categorical variable and that $T(\mathbf{x})$ and $T(\mathbf{x}')$ only differ in the values of the corresponding extra variables associated to that categorical variable. The prior covariance between $f(\mathbf{x})$ and $f(\mathbf{x}')$ under $k(T(\mathbf{x}), T(\mathbf{x}'))$ will be the same as long as $T(\mathbf{x})$ and $T(\mathbf{x}')$ encode a different value for the categorical variable. Of course if $T(\mathbf{x})$ and $T(\mathbf{x}')$ encode the same value for the categorical variable, the covariance will be maximum.

It is straight-forward to show that the proposed transformation generates a valid kernel function. In particular, a kernel is valid if we can find an embedding $\phi(\cdot)$ such that $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathrm{T}} \cdot \phi(\mathbf{x})$ (Shawe-Taylor et al., 2004). Assume that the original kernel is valid and hence $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathrm{T}} \cdot \phi(\mathbf{x})$ for some embedding $\phi(\cdot)$. Then $k(T(\mathbf{x}), T(\mathbf{x}')) = \phi(T(\mathbf{x}))^{\mathrm{T}} \cdot \phi(T(\mathbf{x}))$, and the embedding of the resulting kernel is simply given by $\phi(T(\cdot))$.

6.3.2.1 Visualization of the Proposed Transformation

Figure 6.2 illustrates the modeling properties of the proposed transformation in the case of a real and an integer-valued variable (6.7). It shows the mean and standard deviation of the posterior distribution of a GP given some observations. It compares results with a standard GP that does not use the proposed transformation. In this case, the data has been sampled from a GP using the covariance function in (6.7) with $k(\cdot,\cdot)$ the squared exponential covariance function (Rasmussen, 2003). One dimension takes continuous values and the other dimension takes values in $\{0, 1, 2, 3, 4\}$. Note that the posterior distribution captures the constant behavior of the function in any interval of values that

are rounded to the same integer, only for the integer dimension (top). A standard GP (corresponding to the basic approach in Section 6.3) cannot capture this shape (bottom).

Figure 6.3 illustrates the proposed transformation for the categorical case and a single variable that can only take two values, *e.g.*, *True* and *False*. Using one-hot encoding, these two values will be represented as $(0, 1)$ and $(1, 0)$, respectively. In the naive approach described before, this categorical variable will be replaced by two real variables taking values in the range $[0, 1]$. Notwithstanding, any combination of values in which the first component is larger than the second will lead to the configuration value $(1, 0)$. Conversely, any combination of values in which the second component is larger will lead to the configuration value $(0, 1)$. Therefore, the corresponding objective will be constant in those regions of the input space that lead to the same configuration. This behavior is illustrated by Figure 6.3 (top), in which the posterior distribution of the GP is plotted given two observations. In this case we use the proposed transformation of the covariance function. Note that the uncertainty goes to zero after just having a single observation corresponding to the *True* value and a single observation corresponding to the *False* value. This makes sense, because the objective is constant in all those regions of the input space that lead to the same configuration of the extra variables introduced in the input space. In Figure 6.3 (bottom) we show that a standard GP cannot model this behavior, and the posterior distribution of the mean is not constant in those regions of the input space that lead to the same configuration for the categorical variable. Furthermore, the posterior standard deviation is significantly different from zero, unlike in the proposed approach. Summing up, Figure 6.3 shows that by using the proposed covariance function, we are better modeling the objective function, which in the end will be translated in better optimization results.

6.3.3 Optimization of the Acquisition Function

A consequence of the transformation described in the previous section is that the acquisition function will be flat in some regions of the input space, depending on the number of integer and categorical variables. This behavior is illustrated in Figure 6.1 for one integer-valued variable. Often, the typical approach to optimize the acquisition is to evaluate it first on a grid of points to search for a good candidate point at which to start a gradient-based search using, *e.g.*, L-BFGS. This is the approach employed by the BO software Spearmint. However, if the acquisition function is not smooth, this may be sub-optimal. Assume that we want to optimize a function with D binary categorical inputs, with large D, for example, 30. The best point after evaluating the acquisition function on a grid is extremely unlikely to be the best among the 2^D choices, and the gradient-based optimization of the acquisition function will not leave the starting point, since the acquisition function is expected to be flat at the starting point.

To overcome this problem, we consider a block coordinate ascent optimization methodology which iterates between optimizing the non-real variables (integer-valued and categorical) and real variables, similar to the one-exchange neighborhood (OEN) strategy described in (Hutter, 2009; Lévesque et al., 2017). Our methodology consists in using the grid and the L-BFGS methods to optimize all real variables and then, the OEN strategy to optimize the transformed integer and categorical variables. The OEN strategy is a greedy method. Under it, one iteratively evaluates for each non-real dimension, the corresponding neighbors of that dimension, and if some improvement is made in terms of the acquisition function, that new value is kept as the best one. This process is repeated until no further progress is made. Of course, to evaluate the quality of each neighbor, for

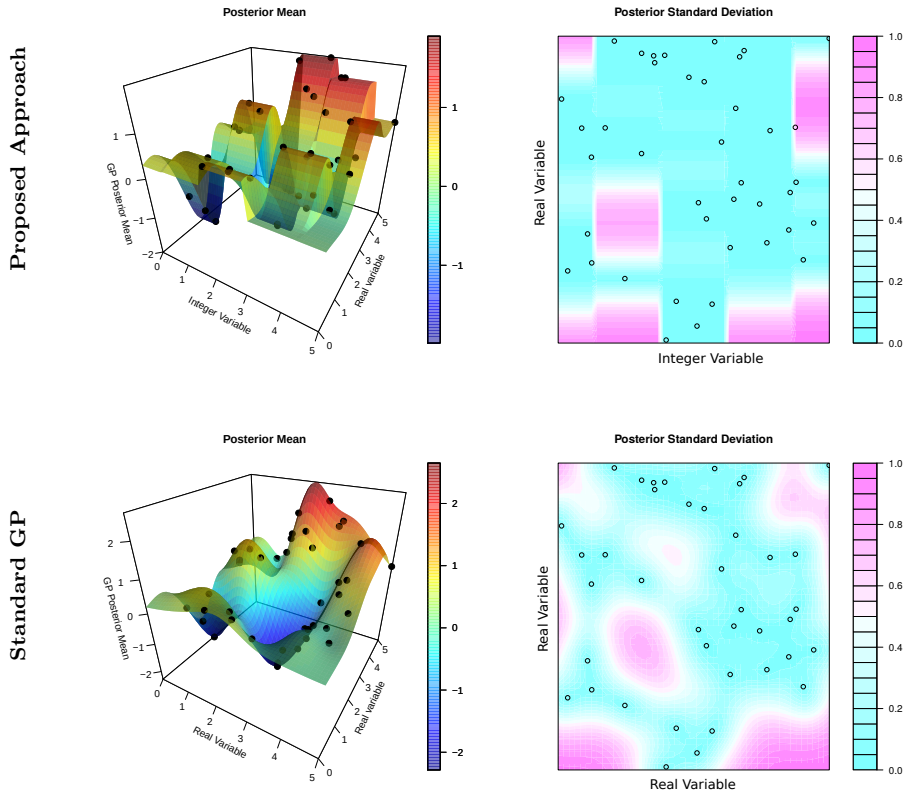

FIGURE 6.2: (top) Posterior mean and standard deviation of a GP model over a 2-dimensional space in which the first dimension can only take 5 different integer values and when the covariance function in (6.7) is used. Note that the second dimension can take any real value. (bottom) Same results for a GP model using a covariance function without the proposed transformation. Best seen in color.

a given integer-valued or categorical dimension, the real variables have to be optimized. For that task, we use the L-BFGS method.

Of course, at this point one may ask whether the proposed transformation of the GP covariance function is beneficial at all, and if simply optimizing the acquisition function as described here could be enough. To answer this question, we have also implemented block coordinate ascent optimization methodology without transforming the integer-valued and categorical variables in the GP covariance function. We compared in a toy problem the results given by both approaches, the proposed approach and the alternating optimization methodology alone, which we refer to as OEN optimization only.

Figure 6.4 shows the evaluations performed by the proposed approach and by OEN optimization only in a 2-dimensional optimization problem with one real variable and one integer-valued variable taking 5 different values. The contour curves show the value of the acquisition function. We show results after 10, 20 and 30 evaluations. We observe that the proposed approach performs a more evenly evaluation of the input space. By contrast, the OEN optimization only strategy, which does not make use of the proposed transformation, tends to concentrate all evaluations in a particular region of the input space. This is a consequence of using a model (*i.e.*, a GP without the proposed transformation) that ignores that the actual objective will take the same value

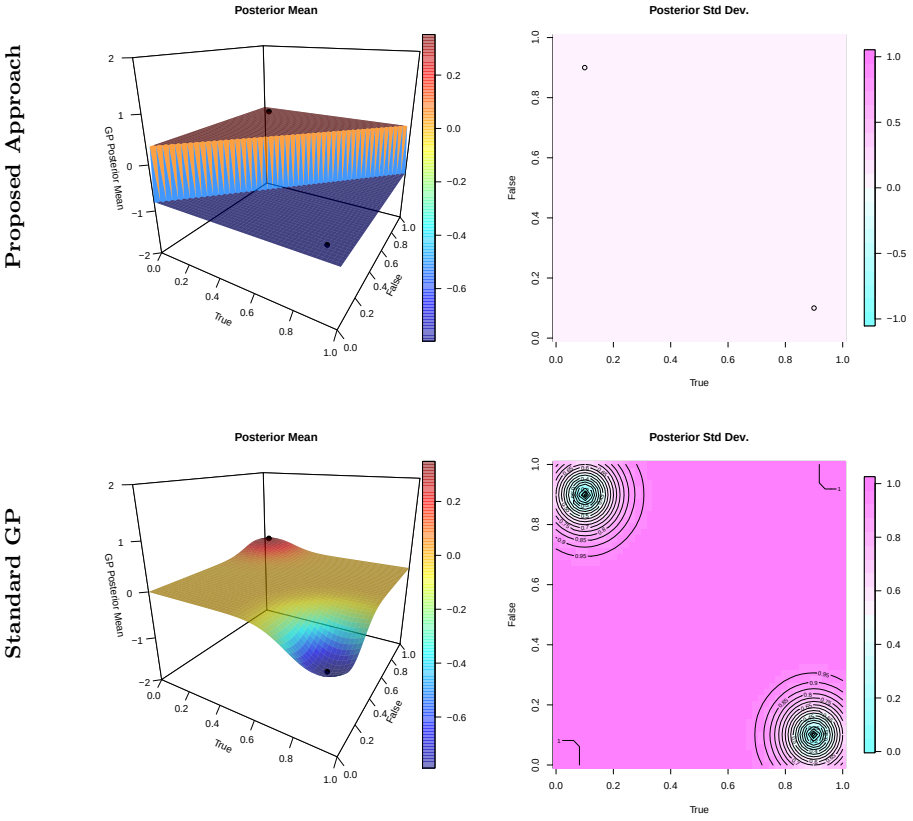

FIGURE 6.3: (top) Posterior mean and standard deviation of a GP model over a 1-dimensional binary variable. The covariance function in (6.7) is used. Same results for a GP model using a covariance function without the proposed transformation. Best seen in color.

in those regions of the input space that lead to the same integer value. In any case, our experiments of Section 6.5 show that OEN optimization only often performs better than the basic approach described in Section 6.3, which simply uses a grid of points combined with L-BFGS to optimize the acquisition function with respect to all input variables, independently of whether they take real, integer or categorical values.

6.4 Related Work

We describe here two approaches that can be used as an alternative to BO methods using GPs when categorical and/or integer-valued variables are present in a black-box optimization problem. These are Sequential model-based optimization for general algorithm configuration (SMAC) (Hutter et al., 2011) and the Tree-structured Parzen Estimator Approach (TPE) (Bergstra et al., 2011). Both can naturally handle integer and categorical-valued variables. SMAC is present in the popular machine learning tool AutoWeka (Thornton et al., 2013). TPE is used in the HyperOpt tool (Bergstra et al., 2013).

SMAC uses a random forest as the underlying surrogate model of the black-box objective (Breiman, 2001). The predictive distribution given by this model is used to

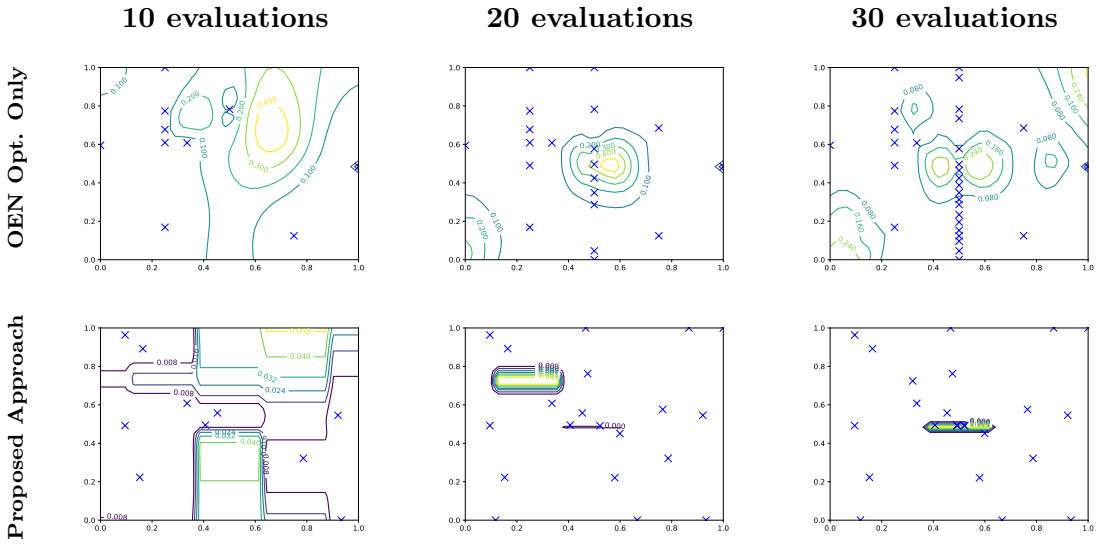

FIGURE 6.4: Evaluations plotted by crosses performed by the OEN optimization only method (top) and the proposed approach (bottom) of a 2-dimensional sample from a GP where one of the variables is integer-valued with range 5. The contour represents the value of the acquisition function in the input space. From left to right, acquisition function with 10, 20 and 30 evaluations. Best seen in color.

select promising parameter values on which the objective should be evaluated. In random forest T random regression trees are iteratively fit using each time a bootstrap sample of training data. Each bootstrap sample is obtained by drawing with replacement from the observed data N instances. Furthermore, in random forest, at each node, a randomly chosen subset of variables are tested to split the data. This introduces variability in the generated regression trees. Given a candidate test location, the prediction for that point is computed for each of the T trees. The predictive distribution of the model is simply a Gaussian distribution with the empirical mean and variance across the individual tree predictions. Given this predictive distribution, the EI criterion described in Section 6.2 is computed and used to select a new point at which the objective $f(\cdot)$ should be evaluated. The main advantage of random forest is that it has a smaller computational cost than a GP.

The regression trees used by random forest to compute the predictive distribution can naturally consider integer and categorical-valued variables. Therefore this method does not suffer from the limitations described in Section 6.3 for GPs. A problem, however, is that the predictive distribution of random forest is not very good. In particular, it relies on the randomness introduced by the bootstrap samples and the randomly chosen subset of variables to be tested at each node to split the data. This result is confirmed by our experiments, in which BO methods using GPs tend to perform better than SMAC.

In SMAC, the EI criterion is optimized by a simple multi-start local search algorithm. This method considers the ten resulting configurations with locally maximal EI from previous runs, and initiates a local search at each of them. To handle mixed categorical and integer parameter spaces, they use a randomized one-exchange neighbourhood search method. The search is stopped when none of the neighbours improves the EI criterion. The configuration with the highest EI value is chosen as the candidate on which to

evaluate the objective at the next iteration. More details on this method are given in (Hutter et al., 2011).

TPE uses EI as the acquisition function. However, its computation is carried out in a different way, using a different modeling strategy. Whereas standard BO methods fit a discriminative model for $p(y|\mathbf{x})$ directly, TPE follows a generative approach. More precisely, $p(\mathbf{x}|y)$ and $p(y)$ are fit instead. Both approaches are related as $p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$ where $p(\mathbf{x}) = \int p(\mathbf{x}|y)p(y)dy$. To obtain an estimate of $p(\mathbf{x}|y)$, TPE models each dimension with a probability distribution that serves as a prior for that dimension. Then, TPE replaces those distributions with non-parametric densities. TPE redefines $p(\mathbf{x}|y)$ by using two different densities, $\ell(\mathbf{x})$ and $g(\mathbf{x})$. $\ell(\mathbf{x})$ is estimated using the observations in which the evaluation is lower than a chosen value $y^\star$. $g(\mathbf{x})$ is estimated using the rest of observations, respectively. That is,

$$p(\mathbf{x}|y) = \begin{cases} \ell(\mathbf{x}) & \text{if} \quad y \leq y^\star \,, \\ g(\mathbf{x}) & \text{if} \quad y > y^\star \,. \end{cases} \tag{6.8}$$

Importantly, these two densities are obtained using Parzen estimators, a non-parametric density estimator, in the case of continuous random variables. In the case of categorical variables, a categorical distribution is used instead. Similarly, in the case of a variable over the integers, a distribution that considers only this domain is used instead. This can easily account for categorical and integer-valued input variables in TPE. $y^\star$ is simply set as some quantile of the observed y values. An interesting property of this approach is that no specific model for $p(y)$ is necessary. TPE derives a different expression for the EI acquisition function. Namely,

$$\begin{aligned} \alpha(\mathbf{x}) &= \int_{-\infty}^{y^\star} (y^\star - y) p(y|\mathbf{x}) dy \\ &= \int_{-\infty}^{y^\star} (y^\star - y) \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})} dy \propto (\gamma + \frac{g(\mathbf{x})}{\ell(\mathbf{x})}(1-\gamma))^{-1} \,, \end{aligned} \tag{6.9}$$

where we have used that $\gamma = p(y < y^\star)$ and that $p(\mathbf{x}) = \int p(\mathbf{x}|y)p(y)dy = \gamma l(\mathbf{x}) + (1-\gamma)g(\mathbf{x})$. See (Bergstra et al., 2011) for further details. Importantly, both of the models, $\ell(\mathbf{x})$ and $g(\mathbf{x})$, are hierarchical processes that naturally take into account discrete-valued and continuous-valued variables.

The TPE EI criterion is hence maximized simply by choosing points with high probability under $\ell(\mathbf{x})$ and low probability under $g(\mathbf{x})$. More precisely, in TPE, at each iteration, the evaluation is performed at the candiate point with greatest EI of many simulated points sampled from $\ell(\mathbf{x})$ and evaluated according to (proportionally) $\ell(\mathbf{x})/g(\mathbf{x})$. The particular form of $\ell(\mathbf{x})$ makes it easy to draw candidates with a mix between discrete and continous variables.

In the literature there are other approaches for BO with GPs that can account for categorical and integer-valued input variables. For example, (Lévesque et al., 2017; Rainforth et al., 2016) suggest to constrain the optimization of the acquisition function to consider only those values that are valid. This is essentially equivalent to the method OEN optimization only described in Section 6.3.3. This method is expected to give sub-optimal results for the reasons explained in that section.

A related approach to our proposed transformation to deal with categorical variables is the kernel proposed in (Hutter, 2009). In that work it is used a weighted Hamming distance kernel to account for this type of variables. That method can be equivalent to

our methodology when the squared exponential kernel is used. However, as we transform the inputs before feeding them into the kernel, our approach is more general and has the advantage of being able to use any valid kernel for GPs. More over, (Hutter, 2009) does not include any empirical evaluation of the benefits of considering such a kernel, nor it explains how to deal with integer-valued variables.

6.5 Experiments

We carry out several experiments to evaluate the performance of our proposed approach for dealing with both integer and categorical-valued variables in Bayesian optimization. We compare the performance of this method with (i) the basic approach described in Section 6.3. We also compare results with (ii) the basic approach that uses the OEN methodology for optimizing the acquisition function (without performing our suggested transformation in the covariance function of the GP). We refer to such a method as OEN optimization only. Each method has been implemented in the software for BO Spearmint in this branch (`https://github.com/EduardoGarrido90/Spearmint`). Finally, we also compare results in both synthetic and real scenarios with two other methods that do not use GPs as the surrogate model. These methods are the ones described in the related work section. Namely, (iii) SMAC and (iv) TPE, as implemented in the HyperOpt platform.

In each experiment carried out in this section, we report average results and the corresponding standard deviations. The results reported are averages over 100 repetitions of the corresponding experiment. Means and standard deviations are estimated using 200 bootstrap samples of the corresponding estimates. For the GPs we use a Matérn covariance function and estimate the GP hyper-parameters using slice sampling (Murray and Adams, 2010). The acquisition function that we employ in these experiments is PES. The hyper-parameters of each GP (length-scales, level of noise and amplitude) are approximately sampled from their posterior distribution using slice sampling as in Snoek (2013). We generate 10 samples for each hyper-parameter, and the acquisition function of each method is averaged over these samples. In the real world scenarios, we generate 50 samples for each hyper-parameter. For each method, at each iteration of the optimization process, we output a recommendation obtained by optimizing the GPs mean functions in the synthetic experiments. In the real-world experiments we return the best observation. Both SMAC and TPE deliver their recommendation based on the best-observed evaluation in both synthetic and real-world scenarios.

The experiments contained in this section are organized as follows: The first set of experiments are synthetic and the objective is sampled from a GP prior. Then, in order to compare GP Bayesian Optimization with non-GP Bayesian Optimization in scenarios where the function is not obtained from a GP, we consider three real optimization problems: Finding an optimal ensemble of trees on the digits dataset and finding an optimal deep neural network on the digits and MNIST datasets.

6.5.1 Synthetic Experiments

We compare the five methods described before when the objective is sampled from a GP prior. For this, we generate optimization problems involving 4 and 6 dimensions. We also consider two settings for each problem involving noisy and noiseless observations. The variance of the additive Gaussian noise is set equal to 0.01 in the noisy setting. In

each problem the objective is randomly sampled from the corresponding GP prior 100 times and we report average optimization results across the different samples.

The first batch of experiments considers 4 input variables. The first 2 variables take real values and the rest of the variables take 4 and 3 different integer values, in the integer case. In the categorical case, the variables take 3 different categories. In the second batch of experiments we consider 6 input variables. The first 3 variables take real values and the other 3 take 4, 3 and 2 different integer values, in the integer case, and 3 different categories, in the categorical case. The next section considers real, categorical and integer-variables at the same time.

In each setting, we sample the objective from a GP prior using (6.7) as the covariance function. Furthermore, we run each BO method (Basic, Proposed, OEN optimization only, SMAC and TPE) for 50 iterations in the 4 inputs problem and for 100 iterations in the 6 inputs problem. For each method, we report the logarithm of the distance to the minimum value of each objective as a function of the evaluations done. In each of the 100 random repetitions of the experiments we use a different random seed to generate the objective.

The average results of each method are displayed in Figure 6.5 for the 4 input setting. We observe that the proposed approach gives better results than the other methods. In particular, it finds points that are closer to the optimal one with a smaller number of evaluations of the objective, both in the case of integer-valued (noiseless and noisy) and categorical-valued scenarios (noiseless and noisy). Figure 6.6 shows similar results for the 6 input setting.

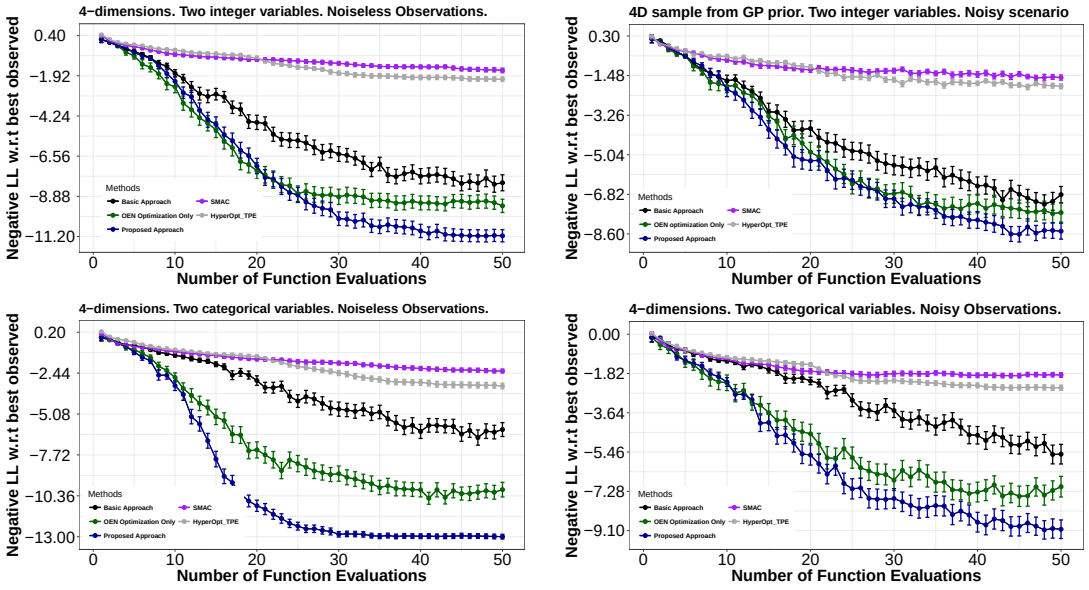

FIGURE 6.5: Average results on the synthetic experiments with 4 dimensions.

We observe that GP based BO outperforms clearly the non-GP based BO, being the proposed approach better than the basic approach or the OEN optimization only method. This last method works better than the basic approach, showing that optimizing the acquisition function with the proposed methodology delivers better results. In the 6-dimensional scenario the difference of performance between the basic approach, OEN optimization only and the proposed approach is slightly higher than in the 4-dimensional scenario. SMAC and TPE also perform worse than the other methods in the 6-dimensional

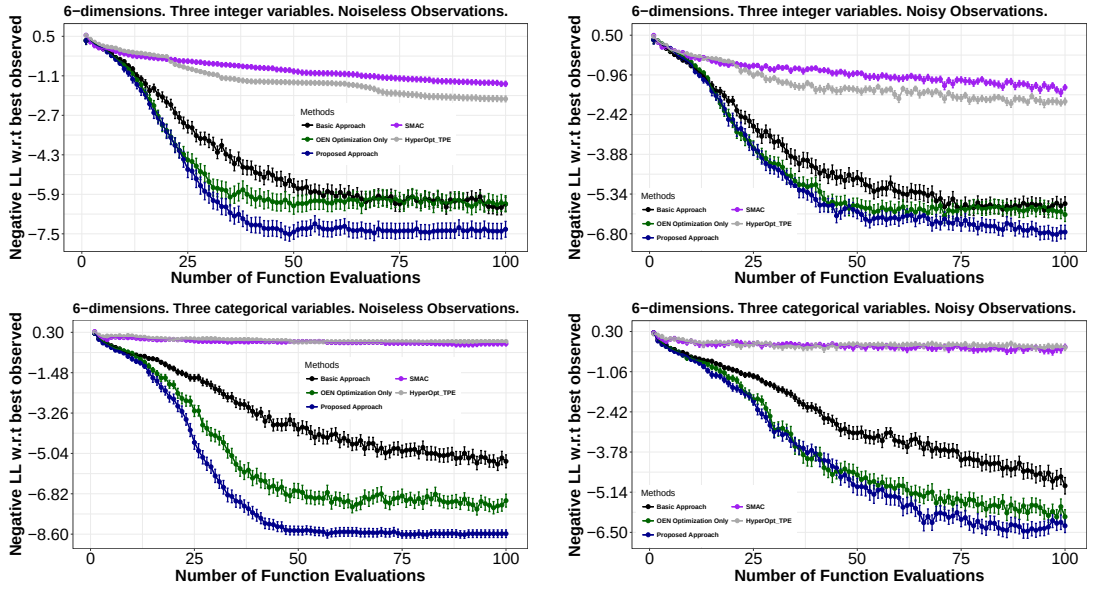

FIGURE 6.6: Average results on the synthetic experiments with 6 dimensions.

case. Finally, in the noisy setting, the methods are more equal but the proposed approach works slightly better. TPE and SMAC also deliver worse results in the noisy setting.

Note that SMAC and TPE do not assume a GP for the underlying model and could be in disadvantage in these experiments. However, we believe it is still interesting to compare results with them in this setting in which the exact solution of the optimization problem can be easily obtained and the level of noise can be controlled. In the following section we carry out experiments in which the actual objectives need not be sampled from a GP, to illustrate the advantages of the proposed approach in a wider range of problems.

6.5.2 Hyper-parameter Tuning of Machine Learning Algorithms

We compare all methods on the practical problem of finding the optimal parameters of a gradient boosting ensemble (Friedman, 2001) and a deep neural network on the digits dataset. This dataset has 1,797 data instances, 10 class labels and 64 dimensions. It has been extracted from the python package scikit-learn (Pedregosa, 2011). Similarly, we also consider finding the optimal hyper-parameters of a deep neural network on the MNIST dataset (LeCun, 1998). This dataset has $60,000$ data instances, 768 dimensions and 10 class labels. In this set of experiments, we use Predictive Entropy Search (PES) as the acquisition function, for both the basic, the proposed approach and the OEN optimization only method.

In the task of finding an optimal ensemble on the digits dataset, the objective that is considered for optimization is the average test log likelihood of the ensemble. This objective is evaluated using a 10-fold cross-validation procedure. Note that model bias can be an issue for all methods in this case, since the actual objective is unknown. We consider a total of 200 evaluations of the objective. A summary of the parameters optimized, their type and their range is displayed on Table 6.1. These parameters are: The logarithm of the learning rate, the maximum depth of the generated trees and the minimum number of samples used to split a node in the tree building process.

Importantly, while the first parameter can take real values, the other two can only take integer values.

TABLE 6.1: Names, types and range of the parameters optimized for the ensemble of trees.

Name	**Type**	**Range**
Log Learning Rate	Real	$[-10, 0]$
Maximum Tree Depth	Integer	$[1, 6]$
Minimum Number of Samples to Split	Integer	$[2, 6]$

In each repetition of the experiment described (there are 100 repetitions) we consider a different 10-fold cross validation split of the data. The average results obtained are displayed in Figure 6.7. This figure shows the average difference, in absolute value, between the test log-likelihood of the recommendation made and the best observed test-log likelihood, for that particular split, in a log scale. We observe that the proposed approach significantly outperforms the basic approach. More precisely, it is able to find parameter values that lead to a gradient boosting ensemble with a better test log likelihood, using a smaller number of evaluations of the objective. Furthermore, the proposed approach also performs better than SMAC, TPE, or the OEN optimization only method.

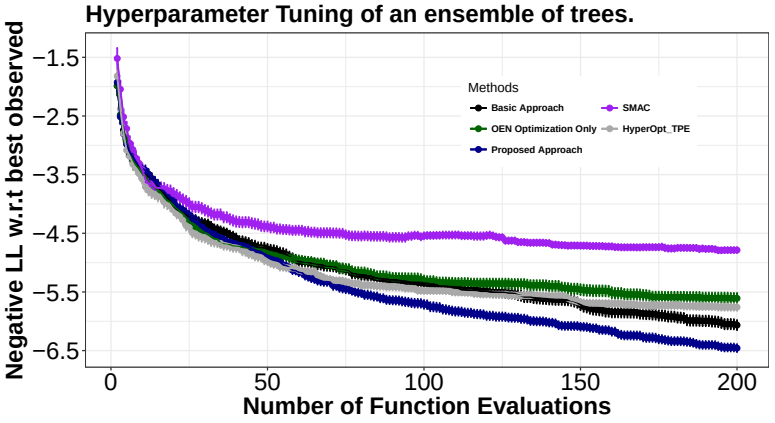

FIGURE 6.7: Average results on the Digits dataset using Gradient Boosting.

In the task of finding an optimal deep neural network on the digits and MNIST dataset, the objective considered is the test log-likelihood of the network. This objective is evaluated using a 10-fold cross-validation procedure in the digits dataset. In the MNIST dataset a validation set of $10,000$ instances, extracted from the training set is used. We consider 125 and 150 evaluations of the objective for the digits and the MNIST dataset, respectively. A summary of the parameters optimized, their type and their range is displayed on Table 6.2. These parameters are: The logarithm of the learning rate, the activation function and the number of hidden layers. The first parameter can take values in the real line. The second and third parameters are categorical and integer-valued. The number of units in each layer, is set equal to 75.

The average results obtained in the two classification problems are displayed in Figure 6.8. The figure shows the average difference, in absolute value, between the test log-likelihood of the recommendation made and the best observed test-log likelihood, in

TABLE 6.2: Name, type and range of the deep neural network parameters optimized.

Name	Type	Range
Log Learning Rate	Real	$[-10, 0]$
Activation Function	Categorical	Linear, Sigmoid, Tanh or ReLU
Number of hidden layers	Integer	$[1, 3]$

a log scale. Again, the proposed approach significantly outperforms the basic approach. More precisely, it is able to find parameter values that lead to a deep neural network with a better test log likelihood on the left-out dataset, using a smaller number of evaluations of the objective. The proposed approach also outperforms SMAC. However, on the MNIST dataset, TPE is only slightly worse than the proposed approach at the end and it outperforms the basic approach. We believe that the better results of TPE obtained in this problem can be a consequence of model bias in the GP that is used to fit the objective in the proposed approach. In both problems, the proposed approach outperforms the OEN optimization only method and the basic approach.

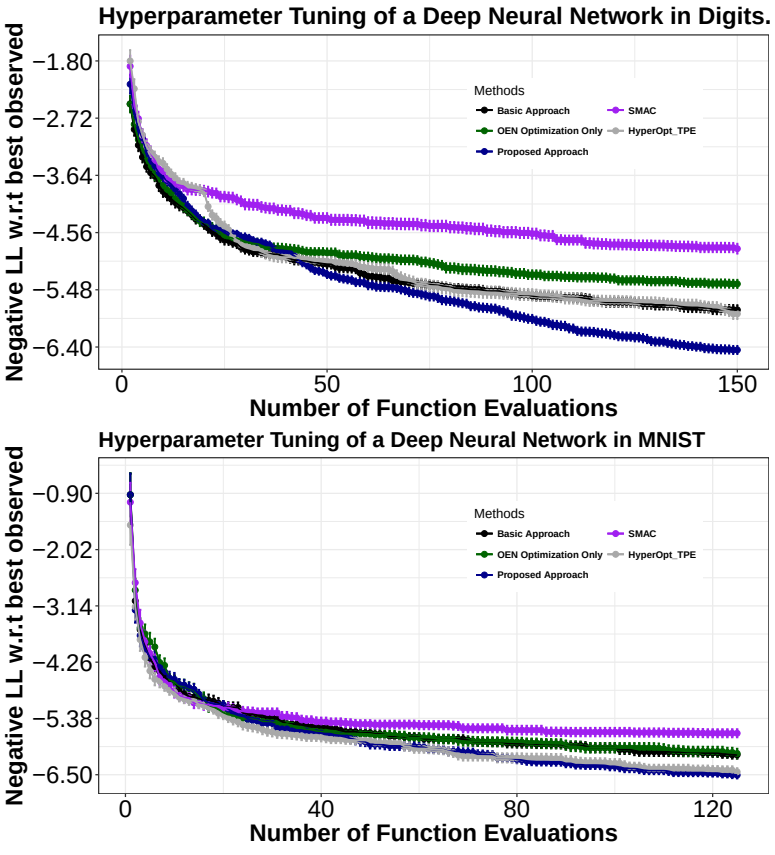

FIGURE 6.8: Average results on the Digits and MNIST dataset using deep neural networks.

6.6 Conclusions

BO methods rely on a probabilistic model of the objective function, typically a Gaussian process (GP), upon which an acquisition function is built. The acquisition function is used to select candidate points, on which the objective should be evaluated, to solve the optimization problem in the smallest number of evaluations. Nevertheless, GPs assume continuous input variables. When this is not the case and some of the input variables take categorical or integer values, one has to introduce extra approximations. A common approach before doing the evaluation of the objective is to use a one-hot encoding approximation for categorical variables, or to round the value to the closest integer, in the case of integer-valued variables. We have shown that this can lead to problems as the BO method can get stuck, always trying to evaluate the same candidate point.

The problem described is a consequence of a mismatch between the regions of the input space that have high acquisition values and the points on which the objective is evaluated. A simple way of avoiding this problem is to do the approximations (a one-hot encoding in the case of categorical variables or the approximation to the closest integer in the case of integer-valued variables) inside the wrapper that is used to evaluate the objective. This technique works in practice, but it has the limitation that it makes the objective constant in those regions of the input space that lead to the same configuration. This constant behavior cannot be modeled by standard GPs.

In this chapter we have proposed to modify the covariance function of the underlying GPs model to account for those regions of the input space in which the objective should be constant. The transformation simply rounds integer-valued variables to the closest integer. In the case of categorical variables in which one-hot encoding has been used, we simply set the largest extra variable equal to one and all the other equal to zero. The consequence of this transformation is that the distance between those points of the input space that lead to the same configuration becomes zero. This enforces maximum correlation between the GP values at those input values, leading to a constant behavior.

The proposed approach has been compared to a basic approach for dealing with categorical and integer-valued input variables in the context BO and GPs. Furthermore, we have also compared results with two other approaches that can be used to solve these optimization problems and that can naturally account for integer and categorical variables. Namely, SMAC and TPE. Several experiments involving synthetic and real-world experiments illustrate the benefits of the proposed approach. In particular, it outperforms the basic approach and SMAC and is most of the times better or at least equivalent to TPE.

The proposed approach also performs better than a strategy that constraints the optimization of the acquisition function to evaluate the objective only at those points that are feasible (*i.e.*, the OEN optimization only method). Such a strategy partially solves the problems of doing a one-hot encoding approximation for categorical variables, or rounding the values to the closest integer, in the case of integer-valued variables. However, we show that it turns out to be sub-optimal in practice, since the GP model considered ignores that the objective becomes constant in those regions of the input space that lead to the same one-hot encoding or the same integer value.

Chapter 7

Bayesian Optimization Of A Hybrid System For Robust Ocean Wave Features Prediction

This chapter describes a Bayesian optimization application on a real case involving the robust prediction of ocean wave features. Specifically, we propose the Bayesian optimization of a hybrid Grouping Genetic Algorithm with an Extreme Learning Machine (GGA-ELM) approach. The system uses data from neighbor stations (usually buoys) in order to predict the significant wave height and the wave energy flux at a goal marine structure facility. The proposed BO methodology has been tested in a real problem involving buoys data in the Western coast of the USA. The results show that BO outperforms the performance of a random search of the hyper-parameters space and the result given by a human expert on the problem.

7.1 Introduction

The accurate prediction of waves features plays a key role in different ocean engineering–related activities, such as safe ship navigation (Liu et al., 2016; Zheng and Sun, 2016), the design of marine structures (Comola et al., 2014; Kim and Suh, 2014), such as oil platforms and harbours, and in marine energy management problems Arinaga and Cheung (2012); Esteban and Leary (2012), like the proper operation of wave energy converters (López et al., 2013), among others. Thus, the topic has a clear impact on human safety, economics and clean energy production. One of the most important features to define the severity of a given ocean wave field is the significant wave height, H_{m_0}. H_{m_0} is usually estimated using in-situ sensors, such as buoys, recording time series of wave elevation information. Buoys provide reliable sea state information that characterizes wave field in a fixed position (i.e. the mooring point). In addition, as buoys are anchored in a hostile media (the ocean), the probability that measuring problems (and therefore missing data) occur in situations of severe weather is very high (Rao and Mandal, 2005). Besides this, marine energy is currently one of the most promising sources of renewable energy, still minor at a global level, but playing a major role in several offshore islands (Bahaj, 2011; Fadaeenejad et al., 2014; Falcao, 2010; Rusu and Soares, 2012). In this case, the accurate estimation of the wave energy flux P is relevant to

characterize the wave energy production from Wave Energy Converters (WECs) facilities (Cuadra et al., 2016).

In this chapter we test a BO methodology to improve the performance of a hybrid prediction system for wave features (H_{m_0} and P) prediction. Specifically, the prediction system is formed by a Grouping Genetic Algorithm for feature selection, and an Extreme Learning Machine for carrying out the final energy flux prediction (Cornejo-Bueno et al., 2016). This hybrid prediction system has a number of parameters that may affect its final performance, and need to be previously specified by the practitioner. Traditionally, these parameters have been manually tuned by a human expert, with experience in both the algorithm and the problem domain. However, it is possible to obtain better results by an automatic fine tuning of the prediction system's parameters. In this case, the parameters of GGA-ELM approach include the probability of mutation in the GGA or the number of neurons in the ELM hidden layer, among others. We propose then to use a Bayesian Optimization (BO) approach to automatically optimize the parameters of the whole prediction system (GGA-ELM), with the aim of improving its performance in wave energy prediction problems. BO has been shown to obtain good results in the task of obtaining good parameter values for prediction systems (Snoek et al., 2012). In the chapter we detail the basic prediction system considered and the BO methodology implemented, along with the improvements obtained in real problems of H_{m_0} and P prediction in the Western coast of the USA.

The rest of the chapter is organized as follows: the next section details the calculation of the features of interest in ocean wave characterization, H_{m_0} and P in this case. Section 7.2 describes the main characteristics of the hybrid system to be optimized, which is formed by a GGA and an ELM for prediction. Section 7.3 presents the real experiments that deal with buoys in the Western coast of the USA whose results show how BO outperforms the random search of the space and the human expert criterion. Finally, Section 7.4 closes the chapter with conclusions and remarks on this research.

7.2 Wave Features of Interest: Calculation of H_{m_0} and P

In the evaluation of marine systems it is essential to previously characterize as accurately as possible the wave features of the zone under study. For example, in a wave energy facility, it is necessary to characterize the amount of wave energy available at a particular location, which is given by features such as H_{m_0} and P. In order to obtain these features, it is necessary to focus on the water surface, and within the framework of the linear wave theory, the vertical wave elevation, $\eta(\mathbf{r}, t)$, at a point $\mathbf{r} = (x, y)$ on the sea surface at time t can be assumed as a superposition of different monochromatic wave components (Borge et al., 2013; Yoshimi, 2010). This model is appropriate when the free wave components do not vary appreciably in space and time (that is, statistical temporal stationarity and spatial homogeneity can be assumed (Yoshimi, 2010)).

In the model described, the concept of "sea state" refers to the sea area and the time interval in which the statistical and spectral characteristics of the wave do not change considerably (statistical temporal stationarity and spatial homogeneity). The features of a given sea state are then the combined contribution of all features from different sources. For example, the "wind sea" occurs when the waves are caused by the energy transferred between the local wind and the free surface of the sea. The "swell" is the situation in which the waves have been generated by winds blowing on another far area (for instance, by storms), and propagate towards the region of observation. Usually, sea states are the composition of these two pure states, forming multi-modal or mixed seas. In a given sea

state, the wave elevation $\eta(\mathbf{r}, t)$ with respect to the mean ocean level can be assumed as a zero-mean Gaussian *stochastic process*, with statistical symmetry between wave maxima and minima. A buoy deployed at point $\mathbf{r}_B$ can take samples of this process, $\eta(\mathbf{r}_B, t_j)$ $j = 1, 2, \cdots, t_{\text{MAX}}$, generating thus a time series of empirical vertical wave elevations. The Discrete Fourier Transform (DFT) of this sequence, using the Fast Fourier Transform (FFT) algorithm, allows for estimating the *spectral density* $S(f)$. Its spectral moments of order n can be computed as follows:

$$m_n = \int_0^\infty f^n S(f) df. \tag{7.1}$$

The Significant Wave Height (SWH) is defined as the average (in meters) of the highest one-third of all the wave heights during a 20-minute sampling period, and it has been widely studied. It can be calculated from the moment of order 0 in Equation (7.1), as follows:

$$H_{m_0} = 4 \cdot (m_0)^{1/2}. \tag{7.2}$$

On the other hand, the wave energy flux is a first indicator of the amount of wave energy available in a given area of the ocean. Wave energy flux P, or power density per meter of wave crest can be computed as

$$P = \frac{\rho g^2}{4\pi} \int_0^\infty \frac{S(f)}{f} df = \frac{\rho g^2}{4\pi} m_{-1} = \frac{\rho g^2}{64\pi} H_{m_0}^2 \cdot T_e, \tag{7.3}$$

where ρ is the sea water density (1025 kg/m^3), g is the acceleration due to gravity, $H_{m_0} = 4\sqrt{m_0}$ is the spectral estimation of the significant wave height, and $T_e \equiv T_{-1,0} = m_{-1}/m_0$ is an estimation of the mean wave period, normally known as the period of energy, which is used in the design of turbines for wave energy conversion (Cahill and Lewis, 2013). Expression (7.3) (with H_{m_0} in meters and T_e in seconds) leads to

$$P = 0.49 \cdot H_{m_0}^2 \cdot T_e, \tag{7.4}$$

measured in kW/m, which helps engineers estimate the amount of wave energy available when planning the deployment of WECs at a given location. The grouping genetic algorithm (GGA) is a type of evolutionary algorithm especially suited to tackle grouping problems, i.e., problems where a number of items must be assigned to a set of predefined groups (Falkenauer, 1993, 1998). The GGA has shown very good performance on different real applications and problems (Agustín-Blas et al., 2011, 2009; Brown and Sumichrast, 2005; De Lit et al., 2000; James et al., 2007a,b). In the GGA, the encoding, crossover and mutation operators of traditional GAs are modified to better deal with grouping problems. In this chapter we use the GGA to obtain a reduced set of features (feature selection) in a context of H_{m_0} and P prediction. We structure the description of the GGA in Encoding, Operators and Fitness Function calculation (Extreme Learning Machine).

7.2.1 Problem Encoding

The GGA is a variable-length genetic algorithm. The encoding is defined by separating each individual in the algorithm into two parts: an *assignment* part, which associates each item to a given group, and a *group* part, which defines the groups that must be taken into account for the individual. In problems where the number of groups is not previously defined, it is straightforward that this is a variable-length algorithm: the

group part varies from one individual to another. In our implementation of the GGA for feature selection, an individual $\mathbf{c}$ has the form $\mathbf{c} = [\mathbf{a}|\mathbf{g}]$. An example of an individual in the proposed GGA for a feature selection problem, with 20 features and 4 groups, is the following:

1 1 2 3 1 4 1 4 3 4 4 1 2 4 4 2 3 1 3 2 | 1 2 3 4

where the group 1 includes features $\{1, 2, 5, 7, 12, 18\}$, group 2 features $\{3, 13, 16, 20\}$, group 3 features {4,9,17,19} and finally group 4 includes features $\{6, 8, 10, 11, 14, 15\}$.

7.2.2 Genetic Operators

In this chapter we use a tournament-based selection mechanism (Yao et al., 1999). This mechanism has been shown to be one of the most effective selection operators, avoiding super-individuals and performing a excellent exploration of the search space. Regarding the crossover operator, we have chosen a modified version of the one initially proposed by Falkenauer (1993, 1998). It follows the process outlined in Figure 7.1:

1. Choose two parents from the current population, at random.
2. Randomly select two points for the crossover, from the "Groups" part of parent 1, then, all the groups between the two cross-points are selected. In the example of Figure 7.1 the two crossover points are G_1 and G_2. Note that, in this case the items of parent1 belonging to group G_1 and G_2 are 1, 2, 4, 5, and 6.
3. Insert the selected section of the "Groups" part into the second parent. After the insertion in the example of Figure 7.1, the assignment of the nodes 1, 2, 4, 5 and 6 of the offspring individual will be those of parent 1, while the rest of the nodes' assignment are those of parent 2. The "Groups" part of the offspring individual is that of parent 2 plus the selected section of parent 1 (8 groups in total, in this case).
4. Modify the "Groups" part of the offspring individual with their corresponding number. In the example, $G = 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 1 \quad 2$ is modified into $G = 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8$. Modify also the assignment part accordingly.
5. Remove any empty groups in the offspring individual. In the example considered, it is found that groups 1, 2, 3, and 6 are empty, so we can eliminate these groups' identification number and rearrange the rest. The final offspring is then obtained.

Regarding mutation operator, we apply a swapping mutation in which two items are interchanged (swapping this way the assignment of features to different groups). This procedure is carried out with a very low probability ($P_m = 0.01$), to avoid increasing of the random search in the process. In the next section we describe the fitness function used to guide the search in the GGA, the ELM neural network, which is a very fast algorithm with excellent performance in prediction problems.

7.2.3 Fitness Function: the Extreme Learning Machine

An ELM is a fast learning method based on the structure of MLPs with a novel way of training feed-forward neural networks (Huang et al., 2006). One of the most important characteristics of the ELM training is the randomness in the process where the network weights are set, obtaining, in this way, a pseudo-inverse of the hidden-layer output matrix. The simplicity of this technique makes the training algorithm extremely fast. Moreover,

it is remarkable the outstanding performance shown when compared to other learning methods. For example, it is usually better than other established approaches such as classical MLPs or SVRs.

The ELM algorithm can be explained as follows: given a training set

$$\mathbb{T} = (\mathbf{x}_i, \boldsymbol{W}_i) | \mathbf{x}_i \in \mathbb{R}^n, \boldsymbol{W}_i \in \mathbb{R}, i = 1, \cdots, l$$

, an activation function $g(x)$ and number of hidden nodes ($\tilde{N}$),

1. Randomly assign inputs weights $\mathbf{w}_i$ and bias b_i, $i = 1, \cdots, \tilde{N}$.
2. Calculate the hidden layer output matrix $\mathbf{H}$, defined as

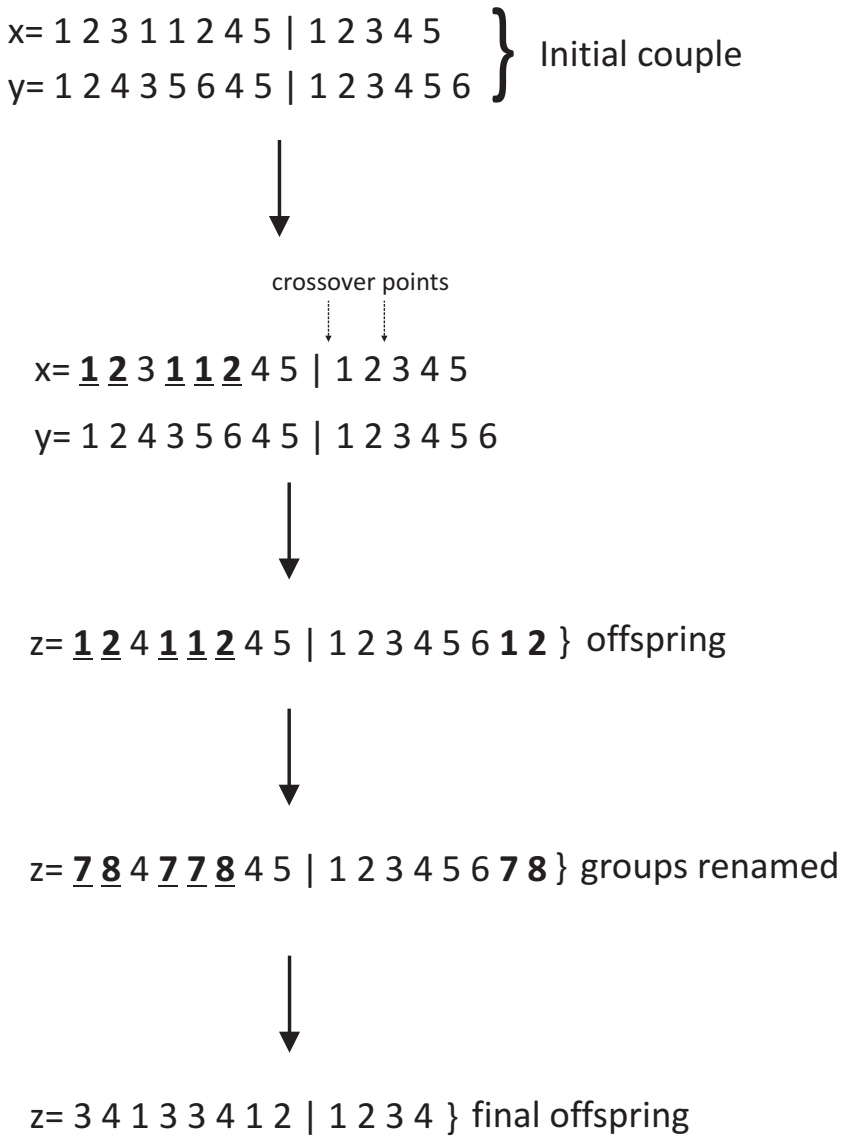

FIGURE 7.1: Outline of the grouping crossover implemented in the proposed GGA.

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{w}_1\mathbf{x}_1 + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}}\mathbf{x}_1 + b_{\tilde{N}}) \\ \vdots & \cdots & \vdots \\ g(\mathbf{w}_1\mathbf{x}_l + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}}\mathbf{x}_N + b_{\tilde{N}}) \end{bmatrix}_{l \times \tilde{N}} \tag{7.5}$$

3. Calculate the output weight vector β as

$$\beta = \mathbf{H}^{\dagger}\mathbf{T}, \tag{7.6}$$

where $\mathbf{H}^{\dagger}$ stands for the Moore-Penrose inverse of matrix $\mathbf{H}$ (Huang et al., 2006), and $\mathbf{T}$ is the training output vector, $\mathbf{T} = [\boldsymbol{W}_1, \cdots, \boldsymbol{W}_l]^T$.

The number of hidden nodes ($\tilde{N}$) is a free parameter of the ELM training, and it can be fixed initially, or in a best convenient way, it must be estimated for obtaining good results as a part of a validation set in the learning process. Hence, scanning a range of $\tilde{N}$ values is the solution for this problem.

These experiments use the Matlab ELM implementation by G. B. Huang, freely available on the Internet (`http://www.ntu.edu.sg/home/egbhuang/elm_codes.html`).

7.3 Experiments

This section presents the experiments carried out in order to show the improvement of performance in the system when it is optimized with the BO techniques shown above. We consider a real problem of wave energy flux prediction ($P = 0.49 \cdot H_s^2 \cdot T_e$ kW/m) from marine buoys (Yoshimi, 2010). Figure 7.2 shows the three buoys considered in this study at the Western coast of the USA, whose data bases are obtained from the National Data Buoy Center. The objective of the problem is to carry out the reconstruction of buoy 46069 from a number of predictive variables from the other two buoys. Thus, 10 predictive variables measured at each neighbor buoy are considered (a total of 20 predictive variables to carry out the reconstruction). Table 7.1 shows details of the predictive variables for this problem. Data for two complete years (1st January 2009 to 31st December 2010) are used, since complete data (without missing values in predictive and objective P) are available for that period in the three buoys. These data are divided into training set (year 2009) and test set (year 2010) to evaluate the performance of the proposed algorithm.

We evaluate the utility of the BO techniques for finding good parameters for the prediction system described in Section 7.2. More precisely, we try to find the parameters that minimize the RMSE of the best individual found by the GGA on a validation set that contains 33% of the total data available. The parameters of the GGA that are adjusted are the probability of mutation $p \in [0, 0.3]$, the percentage of confrontation in the tournament $q \in [0.5, 1.0]$, and the number of epochs $e \in [50, 200]$. On the other hand, the parameters of the ELM that is used to evaluate the fitness in the GGA are also adjusted. These parameters are the number of hidden units $n \in [50, 150]$ and the logarithm of the regularization constant of a ridge regression estimator, that is used to find the weights of the output layer $\gamma \in [-15, -3]$. Note that a ridge regression estimator for the output layer weights allows for a more flexible model than the standard ELM, as the standard ELM is retrieved when γ is negative and large (Albert, 1972).

We compare the BO method with two techniques. The first technique is a random exploration of the space of parameters. The second technique is a configuration specified

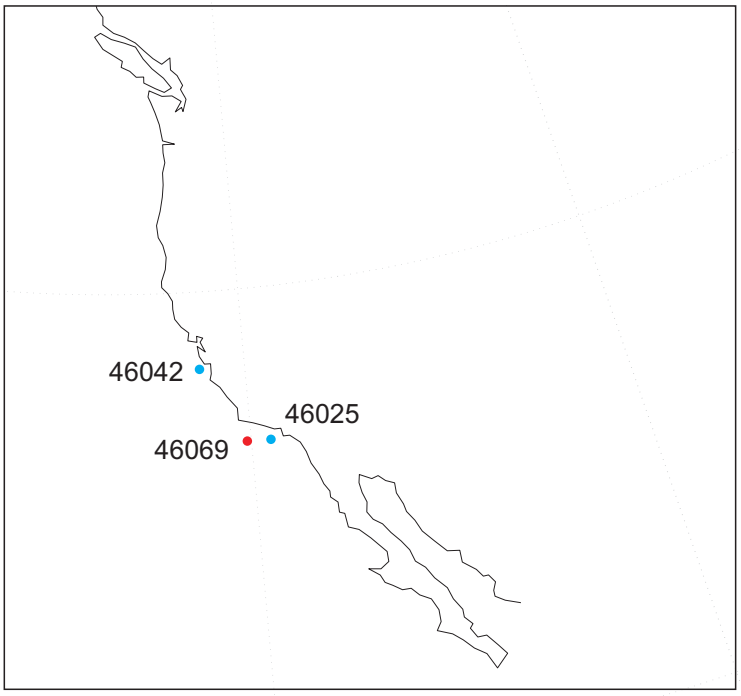

FIGURE 7.2: Western USA Buoys considered in this study. In red buoy where the P prediction is carried out from data at blue ones.

TABLE 7.1: Predictive variables used in the experiments.

Acronym	Predictive variable	units
WDIR	Wind direction	[degrees]
WSPD	Wind speed	[m/s]
GST	Gust speed	[m/s]
WVHT	Significant wave height	[m]
DPD	Dominant wave period	[sec]
APD	Average period	[sec]
MWD	Direction DPD	[degrees]
PRES	Atmospheric pressure	[hPa]
ATMP	Air temperature	[Celsius]
WTMP	water temperature	[Celsius]

by a human expert. Namely, $p = 0.02$, $q = 0.8$, $e = 200$, $n = 150$ and $\gamma = -10$. These are reasonable values that are expected to perform well in the specific application tackled. We set our computational budget to 50 different parameter evaluations for both the BO and the random exploration strategy. After each evaluation, we report the performance of the best solution found. The experiments are repeated for 50 different random seeds and we report average results. All BO experiments are carried out using the acquisition function EI and the software for BO Spearmint.

Fig. 7.3 and 7.4 show the average results obtained and the corresponding error bars for the Wave Energy Flux and the Wave Height optimization. This figure shows the average RMSE of each method (BO and random exploration) on the validation set as a function of the number of configurations evaluated. The performance of the configuration specified by a human expert is also shown. We observe that the BO strategy performs best. After a few evaluations is able to outperform the results of the human expert and it provides results that are similar or better than the ones obtained by the random exploration strategy with a smaller number of evaluations.

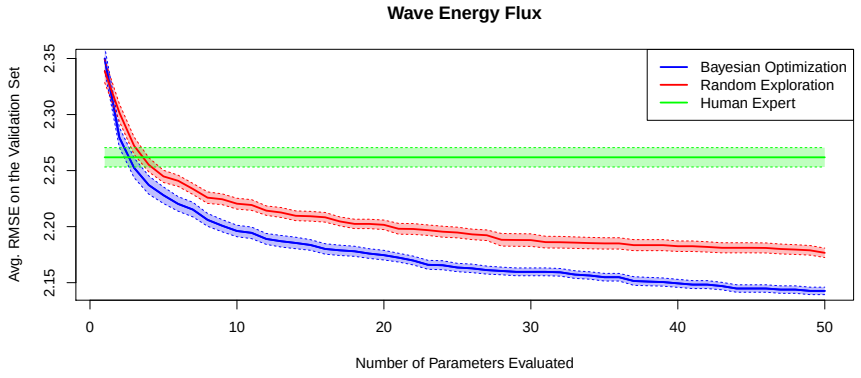

FIGURE 7.3: Average results obtained for the Wave Energy Flux optimization after evaluating the performance of 50 different parameters for the BO technique and a random exploration of the parameter space. The performance a configuration specified by a human expert is also shown for comparison.

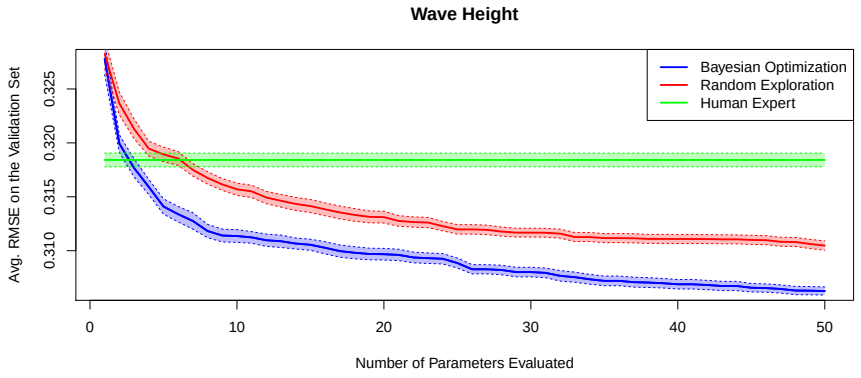

FIGURE 7.4: Wave Height optimization average results of the performance of the 50 different parameter values selected by the BO technique and a random exploration of the parameter space. The plot also shows the performance of the parameter values selected by a human expert.

7.4 Conclusions

In this section we have shown how a hybrid prediction system for wave energy prediction can be improved by means of BO. The prediction system is formed by a grouping genetic algorithm for feature selection, and an Extreme Learning Machine for effective prediction of the target variable, the wave energy flux in this case. After this feature selection process, the final prediction of the wave energy flux is obtained by means of an ELM or a SVR approach. The chapter describes the specific application of BO in the optimization of the GGA-ELM for a real problem of wave energy flux prediction from buoys data in Western California USA. The results show how BO outperforms the results given by a random search of the space and the results given by the criterion of a human expert on the problem.

Chapter 8

Conclusions And Future Work

Bayesian optimization is used to optimize black-box functions, *i.e.*, potentially noise expensive functions with unknown analytical expressions. In particular, examples of black-box optimization include hyper-parameter tuning of machine learning algorithms to optimize an estimation of the generalization error, industry problems such as the optimization of the configuration of a robot or even curious applications such as the optimization of the cooking recipe of a cookie. Concretely, the standard BO scenario concerns the optimization of a single black-box function. However, this thesis has introduced methods that extend the applicability of BO to broader scenarios such as constrained multi-objective scenarios (Chapter 4), parallel BO (Chapter 5) or dealing with integer and categorical-valued variable (Chapter 6). We also include a real application regarding the optimization of a hybrid grouping genetic algorithm applied to an extreme learning machine for robust prediction of ocean wave features in Chapter 7. In those chapters, we have illustrated the usefulness of the proposed methods with toy, synthetic, benchmark or real problems. Finally, in this section, we illustrate the main conclusions of this thesis.

8.1 Conclusions

We provide a list with the main conclusions of this thesis regarding the work that has been shown in the previous chapters.

- First, we developed a method that performs Bayesian optimization in constrained multi-objective scenarios. This method is called Predictive Entropy Search for multi-objective optimization with constraints (PESMOC). Concretely, this method optimizes conflicting black-boxes under the presence of several constraints. In particular, solutions that do not fulfill the constraints are not considered valid. Most importantly, PESMOC iteratively suggests the recommendation that minimizes the expected reduction on the entropy of the Pareto set. In Chapter 4 we showed the usefulness of PESMOC for optimizing several objectives and constraints regarding ensembles and the implementation of deep neural networks in hardware. In those experiments, PESMOC outperforms other methods tackling the constrained multi-objective scenario such as a generalization of the expected improvement acquisition function, that is expected to be greedy, or a random search, that only performs pure exploration.

- As we have already seen, Bayesian optimization suggests a single recommendation point in every iteration of the algorithm. Nevertheless, there are settings where we may have a cluster of nodes available to process suggestions. Unfortunately, standard Bayesian optimization leaves them idle as it can only provides a single suggestion at a time. In order to solve this issue, we generalized the previous constrained multi-objective approach to make it suggest a batch of points for every iteration. In particular, this extension of PESMOC is called Parallel Predictive entropy search for multi-objective optimization with constraints (PPESMOC). In this case, the acquisition function now suggests a batch of points that minimize the expected reduction on the entropy search of the Pareto set. As in the case of PESMOC, we tested PPESMOC on the same experiments of PESMOC but comparing it with greedy versions of PPESMOC where we iteratively create a batch of points using the PESMOC acquisition function. Critically, PPESMOC outperforms or performs similarly to these methods on the proposed experiments. Moreover, it scales the batch size better than the proposed baselines.

- We have studied that BO uses Gaussian processes to model the black-box. GPs assume continuous input real variables. When the problem involves other variables, such as integer-valued or categorical variables, it is common to perform a one hot encoding approximation for categorical variables or to round the integer-valued variables to the closest integer. In this work, we show how such procedures incur in a bad performance of the BO method. To circumvent this issue, we propose a transformation for the input space variables that alleviates the issues that arise in previous procedures. In particular, we include empirical evidence that shows how our transformation outperforms previous approaches dealing with integer-valued and categorical variables.

- Finally, we propose the Bayesian optimization of a hybrid grouping genetic algorithm for attribute selection combined with an extreme learning machine (GGA-ELM) approach for robust prediction of ocean wave features. The contribution of the thesis regarding this work was the design and implementation of the experiments. In Chapter 7, we perform two sets of experiments regarding the prediction of the wave energy flux and wave height optimization. Critically, we showed how BO outperforms the configuration suggested by experts on the field and the performance delivered by random search.

8.2 Future Work

Our proposed methods show how BO can be effectively adapted to cover a wide range of different scenarios with an excellent performance. Moreover, there are a plethora of scenarios that BO can cover in addition to the described settings that could be targeted by future work. Precisely, to conclude this document, we include some ideas as further work that can be studied by future research.

- PESMOC assumes that the black-boxes to be optimized are independent. But this is not necessarily true in real-world problems. The black-boxes can have dependencies. For example, consider the optimization of the number of workers of the company and the benefit. In this example, the benefit is dependent on the hired number of employees. We hypothesize that, by modelling these dependencies, we can improve the performance of the PESMOC method (Shah and Ghahramani,

2016). This work will need to use a multi-output GP to model these dependencies and to propose a new acquisition function, for example extending PESMOC, that takes into account the information provided by this model (Moreno-Muñoz et al., 2018).

- We have used Bayesian optimization with Gaussian processes. Gaussian processes are flexible priors over functions, but may have difficulties modelling non-stationary functions. An extension of GPs, deep Gaussian processes, are designed to be priors over non-stationary functions (Bui et al., 2016; Damianou and Lawrence, 2013). Deep GPs consist of multiple GP mappings organized in several layers. The input of every layer is the output of the previous layer, that consist of several sparse GP. The nodes are sparse GPs to make the deep GP scale to more observations, typically being used from 500 to 5000000 observations. However, Bayesian optimization scenarios do not usually consider more than 300 observations. Hence, if deep GPs are used for BO, they need to be modified to include GPs in their layers as the complexity in the number of observations is not a problem in the BO setting but if we use sparse GPs the performance will suffer. A future line of research is to design a deep GP that can perform well for BO scenarios. In order to do so, we will need to perform hyper-parameter sampling, that is not usually done for deep GPs on regression problems.

- The acquisition function of Bayesian optimization can easily be optimized in a low number of dimensions with a grid search procedure and a local optimizer method such as L-BFGS. In most cases, it is also possible to compute the gradients of the acquisition function. In particular, the number of points of the grid search is implemented to scale linearly with the number of dimensions. However, due to the curse of dimensionality, as we increment the number of dimensions or for parallel acquisition functions such as PPESMOC (whose dimensional complexity is a function of the size of the batch) and the number of dimensions of the input space this methodology will deliver worse and worse results. In order to circumvent this issue, it would be interesting to test evolutionary strategies such as NSGA-II or other metaheuristic strategies to optimize the acquisition function in a high number of dimensions to enhance the results obtained by the proposed methods (Deb et al., 2002).

- An interesting application where we can apply constrained multi-objective BO is to deploy fair ML algorithms (Perrone et al., 2020). Fairness strategies ensure ML algorithms not to incur in discriminations such as racism, machism or ageism. These strategies may be conflicting with the optimization of the estimation of the generalization error. Moreover, they can be considered as black-boxes. Additionally, if we want that these strategies can invalidate certain solutions, we can also implement them as constraints. Therefore, the methods proposed, and more precisely PESMOC, could be used to address the problem described. Further research may analyze the utility of PESMOC for finding fair machine learning models.

Appendix A

Probability Distributions

In this appendix, we define some probability theory concepts that are used in the chapters of the thesis. We specifically describe the fundamentals of probability theory and some ideas of the Gaussian distribution.

A.1 Probability Theory

Let $\mathbf{x} \in \mathbb{R}^N$. Let $X_1, ..., X_N$ be N random variables. The d-dimensional probability density function $p(X_1 = x_1, \cdots, X_N = x_n)$ of the random variables $X_1, ..., X_N$ with support in all $\mathbb{R}^N$ satisfies

$$\int_{\mathbb{R}^N} p(\mathbf{x}) d\mathbf{x} = 1\,. \tag{A.1}$$

For clarity, we abbreviate the notation of probability density functions from $p(X_1 = x_1, \cdots, X_N = x_n)$ to $p(\mathbf{x})$. If $p(\mathbf{x})$ is a probability density function of a N-dimensional real-valued space, then, its cumulative distribution function $F(x_l^1 \leq X_1 \leq x_u^1, \cdots, x_l^N \leq X_N \leq x_u^N)$ is given by:

$$F(x_l^1 \leq X_1 \leq x_u^1, \cdots, x_l^N \leq X_N \leq x_u^N) = \int_{x_l^1}^{x_u^1} \cdots \int_{x_l^N}^{x_u^N} p(\mathbf{x}) d\mathbf{x}\,. \tag{A.2}$$

For clarity, we abbreviate the notation of cumulative distribution functions from $F(x_l^1 \leq X_1 \leq x_u^1, \cdots, x_l^N \leq X_N \leq x_u^N)$ to $F(\mathbf{x})$. Let $X = (Y, Z) \in \mathbb{R}^2$ denote two random variables $Y \in \mathbb{R}, Z \in \mathbb{R}$ with a bivariate probability density function $p(\mathbf{x})$ on $\mathbb{R}^2$. Then, the marginal densities of Y and Z are given by the sum rule of probability:

$$p(y) = \int_{\mathbb{R}} p(y, z) dz, \qquad p(z) = \int_{\mathbb{R}} p(y, z) dy. \tag{A.3}$$

This operation is also known as marginalization. It is usually performed to accumulate the uncertainty of the random variables that are not used for future computations. The conditional density $p(y|z)$, for $p(z) > 0$, is given by the product rule of probability:

$$p(y|z) = \frac{p(y, z)}{p(z)}, \qquad p(y, z) = p(y|z)p(z) = p(z|y)p(y). \tag{A.4}$$

Using Eq. (A.4), we can analytically obtain Bayes theorem:

$$p(y|z) = \frac{P(z|y)P(y)}{P(z)}. \tag{A.5}$$

In Bayes theorem, we define $P(Z)$ as the model evidence, $P(y)$ is the prior distribution, $P(Z|Y)$ is the likelihood function and $P(Y|Z)$ is the posterior distribution. According to the total probability theorem, we have that the model evidence can be computed by the integral of the likelihood times the prior:

$$P(z) = \int P(z|y)P(y)dy\,. \tag{A.6}$$

Let z be substituted by observed data $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1, ..., N\}$ where N is the number of tuples $\mathbf{x}_i, y_i)$ and y be the hyper-parameters $\boldsymbol{\theta}$ of a model $\mathcal{M}$. Bayes theorem can compute the posterior distribution of the hyper-parameters given the data $p(\boldsymbol{\theta}|\mathcal{D})$:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}\,, \tag{A.7}$$

where the likelihood function $p(\mathcal{D}|\boldsymbol{\theta})$ is computed as the product of the likelihood function $p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$ of each of the tuples $(\mathbf{x}_i, y_i)$ as:

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \boldsymbol{\theta})\,. \tag{A.8}$$

$p(\boldsymbol{\theta})$ is the prior over the model hyper-parameters, that can be set as an uninformative prior or to some probability distribution given previous knowledge. Subjective bias is, from our opinion, always going to be present as, although we can set an uninformative prior for the weights, we are conditioning the posterior distribution given our subjective beliefs. We are setting an uninformative prior because we have the subjective belief that it is the best decision for the prior distribution. Hence, we are including a subjective bias in the computation. Finally, $p(\mathcal{D})$ is the model evidence. We can also rewrite Bayes theorem using the theorem of total probability as the following expression:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}}\,. \tag{A.9}$$

Let $\boldsymbol{\Theta}$ be the space of hyper-parameter values. Let every possible value of the hyper-parameters $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ be defined as a hypothesis that the model $\mathcal{M}$ can formulate about the data $\mathcal{D}$. The model evidence $p(\mathcal{D})$ represents the probability of the data given the model $\mathcal{M}$. In other words, how well does the model $\mathcal{M}$ fit the data $\mathcal{D}$ under every possible hypothesis $\boldsymbol{\theta}$ that can generate. It is used for Bayesian model selection. Models with higher marginal likelihood $p(\mathcal{D})$ explain the data better than those with lower marginal likelihood $p(\mathcal{D})$. The marginal likelihood $p(\mathcal{D})$ acts as normalization constant of the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$. The posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ of the hyper-parameters $\boldsymbol{\theta}$ given the data $\mathcal{D}$ represents all the possible hypotheses $\boldsymbol{\theta}$ of a model $\mathcal{M}$ weighted by their probability conditioned on data $\mathcal{D}$. We can update the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ of the hyper-parameters $\boldsymbol{\theta}$ given the data $\mathcal{D}$ every time that new data $\mathbf{x}_i, y_i)$ is available. New data incorporates a new likelihood function $p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$ that affects the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$. In a new iteration, we can state that the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ becomes

the new prior and the process is repeated. Hence, we can say that *iudicium posterium discipulus est prioris* than translated means, the posterior is the student of the prior.

Finally, we can use the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ to make predictions with the model. Let $\mathbf{x}$ be a new data instance. We want to predict the value y associated with the data instance $\mathbf{x}$. Let $y = f(\mathbf{x})$ be the unknown ground truth. A model $\mathcal{M}$ performs a prediction of the value y associated with the data instance $\mathbf{x}$ given the observed data $\mathcal{D}$. As we can consider every possible hypothesis $\boldsymbol{\theta}$ that model $\mathcal{M}$ can compute of the data, we can compute a predictive distribtion $p(f(\mathbf{x})|\mathbf{x}, \mathcal{D})$ of the value y associated with the data instance $\mathbf{x}$. The predictive distribution $p(f(\mathbf{x})|\mathbf{x}, \mathcal{D})$ can be seen as the pondered average of all the possible model predictions $p(f(\mathbf{x})|\mathbf{x}, \boldsymbol{\theta})$ times the probability of the model being configurated with the hyper-parameter values $\boldsymbol{\theta}$ given the data $\mathcal{D}$. This probability is computed with the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$. Hence, the predictive distribution $p(f(\mathbf{x})|\mathbf{x}, \mathcal{D})$ can be computed as the product of $p(f(\mathbf{x})|\mathbf{x}, \boldsymbol{\theta})$ and $p(\boldsymbol{\theta}|\mathcal{D})$ marginalizing the model hyper-parameters $\boldsymbol{\theta}$:

$$p(f(\mathbf{x})|\mathbf{x}, \mathcal{D}) = \int p(f(\mathbf{x})|\mathbf{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta} \,. \tag{A.10}$$

A.2 Gaussian Distribution

Let us first consider the case of univariate Gaussian distributions. Let X be a random variable that is normally distributed. The associated probability density function $\mathcal{N}(x|\mu, \sigma)$ of the normally distributed random variable X is parametrized by a mean μ and its standard deviation σ. The analytical expression of the probability density function associated to $X = x$ is:

$$\mathcal{N}(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \,. \tag{A.11}$$

The multivariate Gaussian distribution is the generalization of the univariate Gaussian distribution in the multi-dimensional scenario. Let $\mathbf{X} = [X_1, ..., X_d]$ be a random variable vector. The multivariate Gaussian distribution corresponds to the joint probability density function of the vector of random variables. It is parametrized by a d-dimensional mean vector $\boldsymbol{\mu}$ and a positive semi-definite square covariance matrix $\boldsymbol{\Sigma}^{d\times d}$. The analytical expression of the probability density function associated to $\mathbf{X} = \mathbf{x}$ is:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} \exp\left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\} , \tag{A.12}$$

where the superscript T means transpose. The analytical expression entropy $H[\mathcal{N}(x|\mu, \sigma)]$ of the univariate Gaussian distribution is:

$$H[\mathcal{N}(x|\mu, \sigma)] = \frac{1}{2} \log(2\pi \mathrm{e} \sigma^2) \,. \tag{A.13}$$

The entropy $H[\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})]$ of the multivariate Gaussian distribution is:

$$H[\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})] = \frac{1}{2} \log[(2\pi \mathrm{e})^d \boldsymbol{\Sigma}] \,. \tag{A.14}$$

The Gaussian distribution is an exponential family member. Exponential family is closed under product and division. Hence, both the multiplication and division operations are closed for multivariate Gaussian distributions. That is, the multiplication of two

Gaussian distributions $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ is another Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. It is defined by the following expression,

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) \propto \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \,, \tag{A.15}$$

where $\boldsymbol{\Sigma} = \left(\boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1}\right)^{-1}$ and $\boldsymbol{\mu} = \boldsymbol{\Sigma}\left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2\right)$. The previous expression represents an unnormalized distribution. The normalization constant of $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is given by the following expression:

$$Z = \sqrt{\frac{|\boldsymbol{\Sigma}|}{(2\pi)^d |\boldsymbol{\Sigma}_1||\boldsymbol{\Sigma}_2|}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2^T\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2 - \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right)\right\} \,, \tag{A.16}$$

where d is the dimensionality of the the Gaussian distribution. Regarding the division of two multivariate Gaussian distributions, we also have equivalent analytical expressions. In particular, the division of Gaussian distributions $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ is also another Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ which is given by the following analytical expression:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)/\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) \propto \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \,, \tag{A.17}$$

where $\boldsymbol{\Sigma} = \left(\boldsymbol{\Sigma}_1^{-1} - \boldsymbol{\Sigma}_2^{-1}\right)^{-1}$ and $\boldsymbol{\mu} = \boldsymbol{\Sigma}\left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2\right)$. As in the case of the product of two Gaussian distributions $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, we can also compute the normalization constant Z of the unnormalized distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The normalization constant Z can be computed via the following expression:

$$Z = \sqrt{\frac{(2\pi)^d |\boldsymbol{\Sigma}||\boldsymbol{\Sigma}_2|}{|\boldsymbol{\Sigma}_1|}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2^T\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2 - \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right)\right\} \,. \tag{A.18}$$

The Gaussian distribution is an exponential family member. Therefore, the Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ can be rewritten as an exponential family representation via the following expression:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \exp(\boldsymbol{\eta}^T\mathbf{u}(\mathbf{x}) - g(\boldsymbol{\eta})) \,, \tag{A.19}$$

where $\boldsymbol{\eta}$ is a vector of natural parameters, $\mathbf{u}(\mathbf{x})$ is a vector function of $\mathbf{x}$ known as the sufficient statistics and $g(\boldsymbol{\eta})$ is the cumulant generating function or log partition function. The log partition function $g(\boldsymbol{\eta})$ guarantees that $\exp(\boldsymbol{\eta}^T\mathbf{u}(\mathbf{x}) - g(\boldsymbol{\eta}))$ integrates to 1. We can compute the sufficient statistics $\mathbf{u}(\mathbf{x})$ and the log partition function $g(\boldsymbol{\eta})$ via the following expressions for the particular case of the Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$:

$$\begin{aligned}
\mathbf{u}(\mathbf{x}) = (&x_1, \ldots, x_d, \\
&x_1^2, x_1x_2, \ldots, x_1x_d, \\
&x_1x_2, x_2^2, \ldots, x_2x_d, \\
&\vdots \\
&x_dx_1, x_dx_2, \ldots, x_d^2\big)^T \,,
\end{aligned}
\qquad
\begin{aligned}
\boldsymbol{\eta} = -\frac{1}{2}\big(&-2\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1} \,, \\
&\Sigma_{11}^{-1}, \Sigma_{12}^{-1}, \ldots, \Sigma_{1d}^{-1}, \\
&\Sigma_{21}^{-1}, \Sigma_{22}^{-1}, \ldots, \Sigma_{1d}^{-1}, \\
&\vdots \\
&\Sigma_{d1}^{-1}, \Sigma_{d2}^{-1}, \ldots, \Sigma_{dd}^{-1}\big)^T
\end{aligned}
\tag{A.20}$$

and $g(\boldsymbol{\eta}) = \frac{1}{2}\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \frac{d}{2}\log(2\pi) + \frac{1}{2}\log(|\boldsymbol{\Sigma}|)$. Let us consider an univariate Gaussian distribution $\mathcal{N}(x|\mu_g, \sigma_g)$. We can transform the parameters μ_g and σ_g of a Gaussian distribution $\mathcal{N}(x|\mu_g, \sigma_g)$ into its natural parameters μ_n and σ_n using the following

expressions:

$$\mu_n = \frac{\mu_g}{\sigma_g}\,, \tag{A.21}$$

$$\sigma_n = \frac{1}{\sigma_g}\,. \tag{A.22}$$

We can also deconvert the natural parameters, μ_n and σ_n, into the Gaussian ones, μ_g and σ_g, using the following expressions:

$$\sigma_g = \frac{1}{\sigma_n}\,, \tag{A.23}$$

$$\mu_g = \mu_n \sigma_g\,. \tag{A.24}$$

Let us now consider a joint Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_n, \boldsymbol{\sigma}_n \mathrm{I})$ where I is the identity matrix. This distribution factorizes as a product of univariate independent Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_n, \boldsymbol{\sigma}_n \mathrm{I}) = \prod_{i=1}^{N} \mathcal{N}(\mu_n, \sigma_n)$. We can convert the parameters of this joint distribution $\mathcal{N}(\boldsymbol{\mu}_n, \boldsymbol{\sigma}_n \mathrm{I})$ using the following analytical expression:

$$\boldsymbol{\mu}_n = \frac{\boldsymbol{\mu}_g}{diag(\boldsymbol{\sigma}_g)}\,, \tag{A.25}$$

$$\boldsymbol{\sigma}_n = \frac{1}{diag(\boldsymbol{\sigma}_g)}\,. \tag{A.26}$$

We can also deconvert the natural parameters, $\boldsymbol{\mu}_n$ and $\boldsymbol{\sigma}_n$, into the Gaussian ones, $\boldsymbol{\mu}_g$ and $\boldsymbol{\sigma}_g$, using the following expressions:

$$\boldsymbol{\sigma}_g = \frac{1}{\boldsymbol{\sigma}_n}\,, \tag{A.27}$$

$$\boldsymbol{\mu}_g = \boldsymbol{\mu}_n \boldsymbol{\sigma}_g\,. \tag{A.28}$$

Transforming the Gaussian parameters, $\boldsymbol{\mu}_g$ and $\boldsymbol{\sigma}_g$, into the natural ones, $\boldsymbol{\mu}_n$ and $\boldsymbol{\sigma}_n$, is interesting as the expressions for the product and division of Gaussian distributions become simpler. Let us return to the case of the univariate Gaussian distribution. The product of two Gaussian distributions $\mathcal{N}(\mu_{n1}, \sigma_{n1})$ and $\mathcal{N}(\mu_{n2}, \sigma_{n2})$ with natural parameters μ_{n1}, σ_{n1} and μ_{n2}, σ_{n2} is a Gaussian distribution $\mathcal{N}(\mu_{n3}, \sigma_{n3})$ with parameters μ_{n3}, σ_{n3}. The parameters μ_{n3}, σ_{n3} are given by the addition of the parameters of the two distributions $\mathcal{N}(\mu_{n1}, \sigma_{n1})$ and $\mathcal{N}(\mu_{n2}, \sigma_{n2})$ that are being multiplied:

$$\mu_{n3} = \mu_{n1} + \mu_{n2}\,, \tag{A.29}$$

$$\sigma_{n3} = \sigma_{n1} + \sigma_{n2}\,. \tag{A.30}$$

Similarly, the division of two Gaussian distributions $\mathcal{N}(\mu_{n1}, \sigma_{n1})$ and $\mathcal{N}(\mu_{n2}, \sigma_{n2})$ with natural parameters, μ_{n1}, σ_{n1} and μ_{n2}, σ_{n2}, is a Gaussian distribution $\mathcal{N}(\mu_{n3}, \sigma_{n3})$ where its natural parameters μ_{n3}, σ_{n3} are given by the substraction of the two distributions $\mathcal{N}(\mu_{n1}, \sigma_{n1})$ and $\mathcal{N}(\mu_{n2}, \sigma_{n2})$ that are being divided:

$$\mu_{g3} = \mu_{g1} - \mu_{g2}\,, \tag{A.31}$$

$$\sigma_{g3} = \sigma_{g1} - \sigma_{g2}\,. \tag{A.32}$$

Let us consider now a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g)$ with parameters $\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g$. We can compute its natural parameters $\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n$ using the following expressions:

$$\boldsymbol{\mu}_n = \boldsymbol{\Sigma}_g^{-1}\boldsymbol{\mu}_g\,, \tag{A.33}$$

$$\boldsymbol{\Sigma}_n = \boldsymbol{\Sigma}_g^{-1}\,. \tag{A.34}$$

As in the case of univariate distributions, we can reconvert the natural parameters into the Gaussian ones using the following expression:

$$\boldsymbol{\Sigma}_g = (\boldsymbol{\Sigma}_n)^{-1}\,, \tag{A.35}$$

$$\boldsymbol{\mu}_g = \boldsymbol{\Sigma}_g\boldsymbol{\mu}_n\,. \tag{A.36}$$

Consider two multivariate Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ with natural parameters $\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1$ and $\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2$. The product of two Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ is another Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3)$ with natural parameters $\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3$ such that $\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3$ are given by:

$$\boldsymbol{\mu}_3 = \boldsymbol{\mu}_1 + \boldsymbol{\mu}_2\,, \tag{A.37}$$

$$\boldsymbol{\Sigma}_3 = \boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2\,. \tag{A.38}$$

Consider two multivariate Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ with natural parameters $\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1$ and $\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2$. The division of two Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ is another Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3)$ with natural parameters $\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3$ such that $\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3$ are given by:

$$\boldsymbol{\mu}_3 = \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\,, \tag{A.39}$$

$$\boldsymbol{\Sigma}_3 = \boldsymbol{\Sigma}_1 - \boldsymbol{\Sigma}_2\,. \tag{A.40}$$

Let $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\mathcal{N}(\mathbf{y}|\boldsymbol{\mu}', \boldsymbol{\Sigma}')$ be two multivariate Gaussian distributions. The Kullback-Leibler (KL) divergence between $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\mathcal{N}(\mathbf{y}|\boldsymbol{\mu}', \boldsymbol{\Sigma}')$ is

$$\mathrm{KL}\left(\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \mathcal{N}(\mathbf{y}|\boldsymbol{\mu}', \boldsymbol{\Sigma}')\right) = \frac{1}{2}\left(\frac{|\boldsymbol{\Sigma}'|}{|\boldsymbol{\Sigma}|}\right) + \mathrm{trace}\left(\boldsymbol{\Sigma}'^{-1}\boldsymbol{\Sigma} - \mathbf{I}\right) + (\boldsymbol{\mu}' - \boldsymbol{\mu})^T\boldsymbol{\Sigma}'^{-1}(\boldsymbol{\mu}' - \boldsymbol{\mu})\,. \tag{A.41}$$

The moments of two Gaussian distributions $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\mathcal{N}(\mathbf{y}|\boldsymbol{\mu}', \boldsymbol{\Sigma}')$ are matched if $\mathrm{KL}\left(\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \mathcal{N}(\mathbf{y}|\boldsymbol{\mu}', \boldsymbol{\Sigma}')\right) = 0$. Let $p(\mathbf{x})$ be an unnormalized probability distribution, the normalization constant Z of $p(\mathbf{x})$ is given by:

$$Z = \int p(\mathbf{x})d\mathbf{x} \tag{A.42}$$

Let $t(\mathbf{x})$ be an arbitrary function of $\mathbf{x}$ and let

$$Z = \int t(\mathbf{x})\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})d\mathbf{x}\,, \qquad p(\mathbf{x}) = \frac{1}{Z}t(\mathbf{x})\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\,. \tag{A.43}$$

In this case, $p(\mathbf{x})$ is a probability distribution, as it has been normalized by the constant Z. The first and second moments of a multivariate Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ are

given by the following expressions:

$$\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu}\,, \tag{A.44}$$

$$\mathbb{E}[\mathbf{x}\mathbf{x}^T] = \boldsymbol{\Sigma} + \boldsymbol{\mu}\boldsymbol{\mu}^T\,. \tag{A.45}$$

The first and second moments of two distributions can be matched via the following expressions:

$$\mathbb{E}_p[\mathbf{x}] = \boldsymbol{\mu} + \boldsymbol{\Sigma}\frac{\partial \log(Z)}{\partial \boldsymbol{\mu}}\,, \tag{A.46}$$

$$\mathbb{E}_p[\mathbf{x}\mathbf{x}^T] - \mathbb{E}_p[\mathbf{x}]\mathbb{E}_p[\mathbf{x}]^T = \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\left(\frac{\partial \log(Z)}{\partial \boldsymbol{\mu}}\left(\frac{\partial \log(Z)}{\partial \boldsymbol{\mu}}\right)^T - 2\frac{\partial \log(Z)}{\partial \boldsymbol{\Sigma}}\right)\boldsymbol{\Sigma}\,. \tag{A.47}$$

In some practical situations, $\partial \log(Z)/\partial \boldsymbol{\Sigma}$ is not robust. In that cases, we can use the second derivative of the mean, $\partial^2 \log(Z)/\partial(\boldsymbol{\mu})^2$, rather than $\partial \log(Z)/\partial \boldsymbol{\Sigma}$. By doing it so, Eq. (2.43) is now given by the following expression:

$$\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}\mathbf{x}^T] - \mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]^T = \boldsymbol{\Sigma}\frac{\partial^2 \log(Z)}{\partial(\boldsymbol{\mu})^2}\boldsymbol{\Sigma} + \boldsymbol{\Sigma}\,. \tag{A.48}$$

Let the normalization constant of a distribution, Z, be defined by Eq. (A.43). Let $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ be the mean and covariance of the multivariate Gaussian distribution that appears in Eq. (A.43) multiplying the factor $t(\mathbf{x})$. Let $\partial \log(Z)/\partial \boldsymbol{\mu}$ be the derivative of Z with respect to $\boldsymbol{\mu}$. Finally, let $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ be:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}. \tag{A.49}$$

Then, we can infer Eq. (A.48) from the following expressions. First, we compute the first derivative, $\partial \log(Z)/\partial \boldsymbol{\mu}$:

$$\begin{aligned}
\frac{\partial \log(Z)}{\partial(\boldsymbol{\mu})} &= \frac{1}{Z}\frac{\partial Z}{\partial \boldsymbol{\mu}}\,,\\
\frac{\partial Z}{\partial \boldsymbol{\mu}} &= \int t(\mathbf{x})\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})(-0.5)2(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(-1)d\mathbf{x}\,,\\
\frac{\partial Z}{\partial \boldsymbol{\mu}} &= \int t(\mathbf{x})\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}d\mathbf{x}\,,\\
\frac{\partial \log(Z)}{\partial(\boldsymbol{\mu})} &= \frac{1}{Z}\int t(\mathbf{x})\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}d\mathbf{x}\,.
\end{aligned} \tag{A.50}$$

Then, we can compute the second derivative, $\partial^2 \log(Z)/\partial(\boldsymbol{\mu})^2$. We need to know that $\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}] = \int t(\mathbf{x}) N(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\mathbf{x} d\mathbf{x}$:

$$
\begin{aligned}
&\frac{\partial^2 \log(Z)}{\partial(\boldsymbol{\mu})^2} = \frac{\partial 1/Z}{\partial \boldsymbol{\mu}} \int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x} - \boldsymbol{\mu})\boldsymbol{\Sigma}^{-1} d\mathbf{x} + \\
&\frac{1}{Z} \frac{\partial \int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} d\mathbf{x}}{\partial \boldsymbol{\mu}}, \\
&\frac{\partial 1/Z}{\partial \boldsymbol{\mu}} = -\frac{1}{Z^2} \frac{\partial Z}{\partial \boldsymbol{\mu}} = -\frac{1}{Z^2} \int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} d\mathbf{x}, \\
&\frac{\partial \int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} d\mathbf{x}}{\partial \boldsymbol{\mu}} = \int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} d\mathbf{x} - \\
&\int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1} d\mathbf{x}, \\
&\frac{\partial^2 \log(Z)}{\partial \boldsymbol{\mu}^2} = -\frac{1}{Z^2} \int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} d\mathbf{x} \int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x} - \boldsymbol{\mu})\boldsymbol{\Sigma}^{-1} d\mathbf{x} + \\
&\frac{1}{Z}[\int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} d\mathbf{x} - \int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1} d\mathbf{x}], \\
&\frac{\partial^2 \log(Z)}{\partial \boldsymbol{\mu}^2} = -\frac{1}{Z^2}[\int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}) d\mathbf{x}][\int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} d\mathbf{x}] + \\
&\frac{1}{Z}[\int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1}[\mathbf{x}\mathbf{x}^T - 2\boldsymbol{\mu}\mathbf{x}^T + \boldsymbol{\mu}\boldsymbol{\mu}^T]\boldsymbol{\Sigma}^{-1} d\mathbf{x} - \int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1} d\mathbf{x}], \\
&\frac{\partial^2 \log(Z)}{\partial \boldsymbol{\mu}^2} = -\frac{1}{Z}[\int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}) d\mathbf{x}] \frac{1}{Z}[\int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} d\mathbf{x}] + \\
&\frac{1}{Z}[\int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1}[\mathbf{x}\mathbf{x}^T - 2\boldsymbol{\mu}\mathbf{x}^T + \boldsymbol{\mu}\boldsymbol{\mu}^T]\boldsymbol{\Sigma}^{-1} d\mathbf{x} - \int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1} d\mathbf{x}], \\
&\frac{\partial^2 \log(Z)}{\partial \boldsymbol{\mu}^2} = -\boldsymbol{\Sigma}^{-1}[\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}] - \boldsymbol{\mu}]^T[\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}] - \boldsymbol{\mu}]\boldsymbol{\Sigma}^{-1} + \\
&\frac{1}{Z}[\int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1}[\mathbf{x}\mathbf{x}^T - 2\boldsymbol{\mu}\mathbf{x}^T + \boldsymbol{\mu}\boldsymbol{\mu}^T]\boldsymbol{\Sigma}^{-1} d\mathbf{x} - \int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1} d\mathbf{x}], \\
&\frac{\partial^2 \log(Z)}{\partial \boldsymbol{\mu}^2} = -\boldsymbol{\Sigma}^{-1}[\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}] - \boldsymbol{\mu}]^T[\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}] - \boldsymbol{\mu}]\boldsymbol{\Sigma}^{-1} + \\
&\frac{1}{Z}[\int t(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1}[\mathbf{x}\mathbf{x}^T - 2\boldsymbol{\mu}\mathbf{x}^T + \boldsymbol{\mu}\boldsymbol{\mu}^T]\boldsymbol{\Sigma}^{-1} d\mathbf{x}] - \boldsymbol{\Sigma}^{-1},
\end{aligned}
\tag{A.51}
$$

where the derivation continues as follows,

$$\begin{aligned}\frac{\partial^2 \log(Z)}{\partial \boldsymbol{\mu}^2} &= -\boldsymbol{\Sigma}^{-1}[\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}] - \boldsymbol{\mu}][\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}] - \boldsymbol{\mu}]^T\boldsymbol{\Sigma}^{-1} + \\ &\boldsymbol{\Sigma}^{-1}[\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}\mathbf{x}^T] - 2\boldsymbol{\mu}\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]^T + \boldsymbol{\mu}\boldsymbol{\mu}^T]\boldsymbol{\Sigma}^{-1} - \boldsymbol{\Sigma}^{-1}\,, \\ &= \boldsymbol{\Sigma}^{-1}[-[\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}] - \boldsymbol{\mu}][\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}] - \boldsymbol{\mu}]^T + \\ &[\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}\mathbf{x}^T] - 2\boldsymbol{\mu}\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]^T + \boldsymbol{\mu}\boldsymbol{\mu}^T]]\boldsymbol{\Sigma}^{-1} - \boldsymbol{\Sigma}^{-1}\,, \\ &= \boldsymbol{\Sigma}^{-1}[-[\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}] - \boldsymbol{\mu}][\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}] - \boldsymbol{\mu}]^T + \\ &[\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}\mathbf{x}^T] - 2\boldsymbol{\mu}\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]^T + \boldsymbol{\mu}\boldsymbol{\mu}^T] - \boldsymbol{\Sigma}]\boldsymbol{\Sigma}^{-1}\,, \\ &= \boldsymbol{\Sigma}^{-1}[-\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]^T + 2\boldsymbol{\mu}\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]^T - \\ &\boldsymbol{\mu}\boldsymbol{\mu}^T + [\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}\mathbf{x}^T] - 2\boldsymbol{\mu}\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]^T + \boldsymbol{\mu}\boldsymbol{\mu}^T] - \boldsymbol{\Sigma}]\boldsymbol{\Sigma}^{-1}\,, \\ &= \boldsymbol{\Sigma}^{-1}[-\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]^T + \mathbb{E}_{p(\mathbf{x})}[\mathbf{x}\mathbf{x}^T] - \boldsymbol{\Sigma}]\boldsymbol{\Sigma}^{-1}\,, \\ &= \boldsymbol{\Sigma}^{-1}[\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}\mathbf{x}^T] - \mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]^T - \boldsymbol{\Sigma}]\boldsymbol{\Sigma}^{-1}\,,\end{aligned} \tag{A.52}$$

where we see that we can finally substitute $\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}\mathbf{x}^T] - \mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]^T$ by:

$$\begin{aligned}\boldsymbol{\Sigma}\frac{\partial^2 \log(Z)}{\partial \boldsymbol{\mu}^2}\boldsymbol{\Sigma} &= \mathbb{E}_{p(\mathbf{x})}[\mathbf{x}\mathbf{x}^T] - \mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]^T - \boldsymbol{\Sigma}\,, \\ \mathbb{E}_{p(\mathbf{x})}[\mathbf{x}\mathbf{x}^T] - \mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]^T &= \boldsymbol{\Sigma}\frac{\partial^2 \log(Z)}{\partial \boldsymbol{\mu}^2}\boldsymbol{\Sigma} + \boldsymbol{\Sigma}\,.\end{aligned} \tag{A.53}$$

Appendix B

Appendix for Chapter 4

We provide, in this Appendix, additional material that complements the exposition of the PESMOC approach introduced in Chapter 4. Concretely, we show the exact computations done by the expectation propagation algorithm to approximate the intractable factors of the conditional predictive distribution involved in the PESMOC acquisition function approximation. We also include a sensitivity analysis of the number of Monte Carlo iterations of the Slice sampling algorithm. Additionaly, we include another sensitivity analysis of the sampled Pareto sets and an analysis of the infeasible solutions in benchmark experiments.

B.1 The Gaussian Approximation to the Conditional Predictive Distribution

Recall from the main manuscript that, in this work, we wish to approximate the Conditional Predictive Distribution of the set defined by the points $\mathcal{X} = \{\{\boldsymbol{x}_n\}_{n=1}^N \cup \mathcal{X}^* \cup \{\boldsymbol{x}\}\}$. This set is, the union between the $\mathcal{N}$ observation points in the input space $\{\boldsymbol{x}_n\}_{n=1}^N$, the $\mathcal{M}$ Pareto Set points $\mathcal{X}^*$ and the candidate point $\{\boldsymbol{x}\}$ to be evaluated. The Gaussian Approximation will then be a multivariate Gaussian Distribution over $\mathcal{N}$+$\mathcal{M}$+1 variables. The Conditional Predictive Distribution is given by the following expression

$$p(\mathbf{y}|\mathcal{D}, \mathbf{x}, \mathcal{X}^\star) \propto \int p(\mathbf{y}|\mathbf{x}, \mathbf{f}, \mathbf{c}) p(\mathcal{X}^\star|\mathbf{f}, \mathbf{c}) p(\mathbf{f}|\mathcal{D}) p(\mathbf{c}|\mathcal{D}) d\mathbf{f} d\mathbf{c} \tag{B.1}$$

, where $p(\mathbf{y}|\mathbf{x}, \mathbf{f}, \mathbf{c}) = \prod_{k=1}^K \delta(y_k - f_k(\mathbf{x})) \prod_{j=1}^J \delta(y_{K+j} - c_j(\mathbf{x}))$ and $p(\mathcal{X}^\star|\mathbf{f}, \mathbf{c})$ is given by

$$p(\mathcal{X}^\star|\mathbf{f}, \mathbf{c}) \propto \prod_{\mathbf{x}^\star \in \mathcal{X}^\star} \left(\left[\prod_{j=1}^J \Phi_j(\mathbf{x}^\star) \right] \left[\prod_{\mathbf{x}' \in \mathcal{X}} \Omega(\mathbf{x}', \mathbf{x}^\star) \right] \right) , \tag{B.2}$$

where $\Phi_j(\mathbf{x}^\star) = \Theta(c_j(\mathbf{x}^\star))$ with $\Theta(\cdot)$ the Heaviside step function, and $\Omega(\mathbf{x}', \mathbf{x}^\star)$ is defined as:

$$\Omega(\mathbf{x}', \mathbf{x}^\star) = \left[\prod_{j=1}^J \Theta(c_j(\mathbf{x}')) \right] \psi(\mathbf{x}', \mathbf{x}^\star) + \left[1 - \prod_{j=1}^J \Theta(c_j(\mathbf{x}')) \right] \cdot 1 , \tag{B.3}$$

where $\psi(\mathbf{x}', \mathbf{x}^\star)$ is defined as

$$\psi(\mathbf{x}', \mathbf{x}^\star) = 1 - \prod_{k=1}^{K} \Theta(f_k(\mathbf{x}^\star) - f_k(\mathbf{x}')) \,. \tag{B.4}$$

The last two probability densities, $p(\mathbf{f}|\mathcal{D})$ and $p(\mathbf{c}|\mathcal{D})$, involved in the Conditional Predictive Distribution are potentially infinite-dimensional Gaussians given by the Gaussian Process predictive distributions for the objectives $\mathbf{f}$ and constraints $\mathbf{c}$ values. As these distributions are Gaussian, they do not need to be approximated.

In order to find a Gaussian Approximation to Eq.(4) it is necessary to perform several steps. First of all, we separate the factors that depend and not depend on $\mathbf{x}$ so that they will be approximated separately. By doing this, the factors that depend on $\boldsymbol{x}$ are refined only once by EP and the other factors are refined iteratively by EP until they change no more.

The factors that depend on $\mathbf{x}$ are the Dirac Delta functions that can be replaced by Gaussians with the corresponding noise variance in the noise case and no variance in the noiseless case. As they are Gaussian Distributions, there is nothing to approximate.

The other factors are the ones that do not depend on $\mathbf{x}$. We define the sampled Pareto Set as $\mathcal{X}^* = \{\boldsymbol{x}_1^*, ..., \boldsymbol{x}_M^*\}$ of size $\mathcal{M}$ and the set of $\mathcal{N}$ observations in the input space as $\hat{\mathcal{X}} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ with the corresponding observations of the k-th objective $\boldsymbol{y}_k$ and of the c-th constraint $\boldsymbol{y}_j$. Then, the values of the posterior distributions of the GPs of the objectives and the constraints at that points are defined by $\boldsymbol{f}_k = (f_k(\boldsymbol{x}_1^*), ..., f_k(\boldsymbol{x}_M^*), f_k(\boldsymbol{x}_1), f_k(\boldsymbol{x}_N))^T$ and $\boldsymbol{c}_j = (c_j(\boldsymbol{x}_1^*), ..., c_j(\boldsymbol{x}_M^*), c_j(\boldsymbol{x}_1), c_j(\boldsymbol{x}_N))^T$. If we define $\boldsymbol{f} = \{\boldsymbol{f}_1, ..., \boldsymbol{f}_K\}$ and $\boldsymbol{c} = \{\boldsymbol{c}_1, ..., \boldsymbol{c}_J\}$, let $q(\boldsymbol{f}, \boldsymbol{c})$ be the distribution that we want to approximate, $p(\mathcal{X}^\star|\mathbf{f}, \mathbf{c})p(\mathbf{f}|\mathcal{D})p(\mathbf{c}|\mathcal{D})$, with the factors that do not depend on $\mathbf{x}$, that is:

$$q(\boldsymbol{f}, \boldsymbol{c}) = \prod_{\mathbf{x}^\star \in \mathcal{X}^\star} \left(\left[\prod_{j=1}^{J} \Phi_j(\mathbf{x}^\star) \right] \left[\prod_{\mathbf{x}' \in \mathcal{X}} \Omega(\mathbf{x}', \mathbf{x}^\star) \right] \right) p(\mathbf{f}|\mathcal{D}) p(\mathbf{c}|\mathcal{D}) d\mathbf{f} d\mathbf{c} \,. \tag{B.5}$$

Because Eq.(10) is not tractable, we approximate the normalized version of $q(\boldsymbol{f}, \boldsymbol{c})$ with a product of Gaussians, the Gaussian Approximation to the Conditional Predictive Distribution. Eq.(10) can be expressed as this normalized product:

$$\begin{aligned} q(\boldsymbol{f}, \boldsymbol{c}) = \frac{1}{Z_q} \left[\prod_{k=1}^{K} \mathcal{N}\left(\boldsymbol{f}_k \mid \boldsymbol{m}_{pred}^{\boldsymbol{f}_k}, \boldsymbol{V}_{pred}^{\boldsymbol{f}_k}\right) \right] \left[\prod_{j=1}^{J} \mathcal{N}\left(\boldsymbol{c}_j \mid \boldsymbol{m}_{pred}^{\boldsymbol{c}_j}, \boldsymbol{V}_{pred}^{\boldsymbol{c}_j}\right) \right] \times \\ \prod_{\mathbf{x}^\star \in \mathcal{X}^\star} \left(\left[\prod_{j=1}^{J} \Phi_j(\mathbf{x}^\star) \right] \left[\prod_{\mathbf{x}' \in \mathcal{X}} \Omega(\mathbf{x}', \mathbf{x}^\star) \right] \right) \,, \end{aligned} \tag{B.6}$$

where $\boldsymbol{m}_{pred}^{\boldsymbol{f}_k}$ and $\boldsymbol{V}_{pred}^{\boldsymbol{f}_k}$ are the mean and covariance matrix of the posterior distributions of $\boldsymbol{f}_k$ given the data in $\mathcal{D}$ and $\boldsymbol{m}_{pred}^{\boldsymbol{c}_j}$ and $\boldsymbol{V}_{pred}^{\boldsymbol{c}_j}$ are the mean and covariance matrix of the posterior distribution of $\boldsymbol{c}_j$ given the data in $\mathcal{D}$. These means and variances are computed

according to the equations 2.22-2.24 provided by Rasmussen (Rasmussen, 2003):

$$\begin{aligned} \boldsymbol{m}_{pred}^{\boldsymbol{f}_k} &= \boldsymbol{K}_*^k \big(\boldsymbol{K}^f + v_f^2 \mathbb{I}\big)^{-1} \boldsymbol{y}^k \,, \\ \boldsymbol{V}_{pred}^{\boldsymbol{f}_k} &= \boldsymbol{K}_{*,*}^k - \boldsymbol{K}_*^k \big(\boldsymbol{K}^k + v_f^2 \mathbb{I}\big)^{-1} \big[\boldsymbol{K}_*^k\big] \,, \end{aligned} \tag{B.7}$$

where $\boldsymbol{K}_*^k$ is an $(N+1) \times N$ matrix with the prior cross-covariances between elements of $\boldsymbol{f}_k$ and $f_{k,1}, ..., f_{k,n}$ and $\boldsymbol{K}_{*,*}^k$ is an $(N+1) \times (N+1)$ matrix with the prior covariances between the elements of $\boldsymbol{f}_k$ and v_k is the standard deviation of the additive Gaussian noise in the evaluations of $\boldsymbol{f}_k$. Following the same reasoning, we have that:

$$\begin{aligned} \boldsymbol{m}_{pred}^{\boldsymbol{c}_j} &= \boldsymbol{K}_*^j \big(\boldsymbol{K}^j + v_j^2 \mathbb{I}\big)^{-1} \boldsymbol{y}^j \,, \\ \boldsymbol{V}_{pred}^{\boldsymbol{c}_j} &= \boldsymbol{K}_{*,*}^j - \boldsymbol{K}_*^j \big(\boldsymbol{K}^j + v_j^2 \mathbb{I}\big)^{-1} \big[\boldsymbol{K}_*^j\big] \,, \end{aligned} \tag{B.8}$$

where $\boldsymbol{K}_*^j$ is an $(N+1) \times N$ matrix with the prior cross-covariances between elements of $\boldsymbol{c}_j$ and $c_{j,1}, ..., c_{j,n}$ and $\boldsymbol{K}_{*,*}^j$ is an $(N+1) \times (N+1)$ matrix with the prior covariances between the elements of $\boldsymbol{c}_j$ and v_j is the standard deviation of the additive Gaussian noise in the evaluations of $\boldsymbol{c}_j$.

The other non-Gaussian factors presented in Eq.(11), $\Phi_j(\mathbf{x}^\star)$ and $\Omega(\mathbf{x}', \mathbf{x}^\star)$, are the problematic ones, as they are not Gaussian Distributions. Hence they will be approximated by Gaussians with EP, as will be described in the next sections.

B.2 Using Expectation Propagation to Approximate the Conditional Predictive Distribution

This section explains how the EP algorithm approximate the previous product of factors, giving a product of Gausssian Distributions which we call the Gaussian Approximation to the Conditional Predictive Distribution, shown in the previous section. As it is a product where different factors are involved, we have to divide the problem in the approximation of the different factors for Gaussian Distributions. These are the $\Phi_j(\boldsymbol{x}^*$ factors and the $\Omega(\mathbf{x}', \mathbf{x}^\star)$ factors, which will be approximated by one-dimensional and two-dimensional Gaussian Distributions respectively.

The factors $\Phi(\boldsymbol{x}^*)$ that represent if a Pareto Set point $\boldsymbol{x}^*$ is feasible evaluated in a certain constraint $c_j(\boldsymbol{x}^*)$, are approximated by a one-dimensional un-normalized Gaussian distribution $\tilde{\Phi}(\boldsymbol{x}^*)$. This distribution is expressed in exponential family form in the next equation:

$$\Phi(\boldsymbol{x}^*) \approx \tilde{\Phi}(\boldsymbol{x}^*) \propto \exp\left\{ -\frac{c_j(\boldsymbol{x}^*)^2 \hat{v}_j^{\boldsymbol{x}^*}}{2} + c_j(\boldsymbol{x}^*) \hat{m}_j^{\boldsymbol{x}^*} \right\}, \tag{B.9}$$

where $\hat{v}_j^{\boldsymbol{x}^*}$ and $\hat{m}_j^{\boldsymbol{x}^*}$ are natural parameters that are going to be adjusted by EP. The variance of the Gaussian Distribution, $\hat{v}_j^{\boldsymbol{x}^*}$, EP factor in every point, $\boldsymbol{x}^*$, for every constraint, c_j will be denoted by $\hat{e}_j$ and the mean EP factor by $\hat{f}_j$. That is, the one-dimensional Gaussian Distribution approximation of $\Phi(\boldsymbol{x}^*)$, in every constraint c_j computed by EP, $\tilde{\Phi}(\boldsymbol{x}^*)$ is defined in every point $\boldsymbol{x}^*$ belonging to $\mathcal{X}^*$, by its mean $\hat{f}_j$ and its variance $\hat{e}_j$. There will be as many Gaussian Distributions as points multiplied by constants.

The factors $\Omega(\cdot, \cdot)$, that represent if a point $\boldsymbol{x}_j$ is not dominated by the other point $\boldsymbol{x}_i$ and it is feasible over all the constraints $\boldsymbol{c}(\boldsymbol{x}_j)$, are approximated by a product of

$\mathcal{J}$ one-dimensional un-normalized Gaussian Distributions where $\mathcal{J}$ are the number of constraints and $\mathcal{K}$ two-dimensional un-normalized Gaussian Distributions where $\mathcal{K}$ are the number of objectives. This product of distributions is expressed by the following equation:

$$\Omega(\boldsymbol{x}', \boldsymbol{x}^*) \approx \tilde{\Omega}(\boldsymbol{x}', \boldsymbol{x}^*) \propto \prod_{k=1}^{K} \exp\left\{ -\frac{1}{2}\boldsymbol{v}_k^T \tilde{\boldsymbol{V}}_k^{\Omega} \boldsymbol{v}_k + (\tilde{\boldsymbol{m}}_k^{\Omega})^T \boldsymbol{v}_k \right\} \times \prod_{j=1}^{J} \exp\left\{ -\frac{c_j(\boldsymbol{x}^*)^2 \tilde{v}_j^{\Omega}}{2} + c_j(\boldsymbol{x}^*) \tilde{m}_j^{\Omega} \right\}, \tag{B.10}$$

where $\boldsymbol{v}_k$ is defined as the vector $(f_k(\boldsymbol{x}'), f_k(\boldsymbol{x}^*))^T$, and $\tilde{\boldsymbol{V}}_k$, $\tilde{\boldsymbol{m}}_k$, $\tilde{v}_j^{\Omega}$ and $\tilde{m}_j^{\Omega}$ are natural parameters adjusted by EP. As the product represents a product of two-dimensional un-normalized Gaussian Distributions, $\tilde{\boldsymbol{V}}_k$ is a 2×2 matrix and $\tilde{\boldsymbol{m}}_k$ is a two-dimensional vector.

For the set of $\mathcal{N}$ observation points in the input space $\hat{\mathcal{X}}$ and the set of $\mathcal{M}$ Pareto Set points $\mathcal{X}^*$, we define the variance of the two-dimensional Gaussian Distribution, $\tilde{\boldsymbol{V}}_k$, EP factor of an observation point $\boldsymbol{x}_i$ with respect to a Pareto Point $\boldsymbol{x}_j$ as $\hat{\boldsymbol{A}}_{ij}$ and the mean EP factor as $\hat{\boldsymbol{b}_{ij}}$. We denote the variance of the one-dimensional Gaussian Distribution, $\tilde{v}_j^{\Omega}$, EP factor in every point $\boldsymbol{x}_j$ as $\hat{ac}_j$ and the mean EP factor by $\hat{bc}_j$.

For the set of $\mathcal{M}$ Pareto Set points $\mathcal{X}^*$, we define the variance of the two-dimensional Gaussian Distribution, $\tilde{\boldsymbol{V}}_k$, EP factor of a point $\boldsymbol{x}_i$ with respect to another Pareto Point $\boldsymbol{x}_j$ as $\hat{\boldsymbol{C}}_{ij}$ and the mean EP factor as $\hat{\boldsymbol{d}}_{ij}$. We denote the variance of the one-dimensional Gaussian Distribution $\tilde{v}_j^{\Omega}$ EP factor in every point $\boldsymbol{x}_j$ as $\hat{cc}_j$ and the mean EP factor by $\hat{dc}_j$.

That is, the approximation $\tilde{\Omega}(\boldsymbol{x}', \boldsymbol{x}^*)$ computed by EP consisting of a product of one-dimensional Gaussian Distributions and two-dimensional Gaussian distributions of the distribution $\Omega(\boldsymbol{x}', \boldsymbol{x}^*)$, is defined in the set of points $\hat{\mathcal{X}}$ and $\mathcal{X}^*$ by a product of one-dimensional Gaussian Distributions with mean $\hat{bc}_j$ and variance $\hat{ac}_j$ and a product of two-dimensional Gaussian Distributions with variance $\hat{\boldsymbol{A}}_{ij}$ and mean $\hat{\boldsymbol{b}}_{ij}$. The approximation for the set of points $\mathcal{X}^*$ is defined by a product of one-dimensional Gaussian Distributions with mean $\hat{dc}_j$ and variance $\hat{cc}_j$ and a product of two-dimensional Gaussian Distributions with variance $\hat{\boldsymbol{C}}_{ij}$ and mean $\hat{\boldsymbol{d}}_{ij}$.

In the next section, the computations of the Gaussian factor approximations $\tilde{\Phi}(\cdot)$ and $\tilde{\Omega}(\cdot, \cdot)$ defined by the EP factors $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$, $\hat{e}_j$, $\hat{f}_j$, $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$, required by EP, are explained in detail, following the algorithm described in Chapter 3.

B.3 The EP Approximation to the $\Phi(\cdot)$ and $\Omega(\cdot, \cdot)$ Factors

The EP algorithm updates each of the approximate factors presented in the previous section until convergence. The following sections will describe the necessary operations needed for the EP algorithm to update each of the factors. It the following subsection, it is assumed that we have already obtained the mean and variances of each of the $\mathcal{K}$ and $\mathcal{J}$ conditional predictive distributions, which will be explained in detail in section 4.3.

B.3.1 EP Update Operations for the $\Phi(\cdot)$ Factors

As it was explained in section 3, for the $\mathcal{M}$ Pareto Set points defined by the set $\mathcal{X}^*$, in every point $\boldsymbol{x}_i \in \mathcal{X}^*$, the EP algorithm will generate J approximations for the $\Phi(\boldsymbol{x}_j)$ factors for every constraint c_j that will be defined by its mean $\hat{f}_j^{\boldsymbol{x}}$ and its variance $\hat{e}_j^{\boldsymbol{x}}$. Computations are done for all the points $\boldsymbol{x}_i \in \mathcal{X}^*$. The operations for these factors are described as follows.

B.3.1.1 Computation of the Cavity Distribution

The first step performed by the EP algorithm is the computation of the Cavity Distribution $\tilde{q}^{\setminus n}(\boldsymbol{x})$. In order to make the computations easier, we first obtain the natural parameters of the Gaussian Distributions for all the $\mathcal{M}$ Pareto Set points by using the equations:

$$\begin{aligned} \hat{\boldsymbol{m}}_j &= \frac{\boldsymbol{\xi}_j}{diag(\boldsymbol{\Xi}_j)}\,, \\ \hat{\boldsymbol{v}}_j &= \frac{1}{diag(\boldsymbol{\Xi}_j)}\,. \end{aligned} \tag{B.11}$$

Where $\boldsymbol{\Xi}_j$ is a vector of the variances of the $\mathcal{M}$ points for the constraint c_j and $\boldsymbol{\Xi}$ is the matrix of all the variances of all $\mathcal{M}$ and $\mathcal{N}$ points which construction will be explained in detail in section 4.3. The term *diag* holds for the diagonal of $\boldsymbol{\Xi}$ as we are only interested in the variance of the M points and not the variance of these points with the N points for the factor $\Phi(\cdot)$. In the same way, $\boldsymbol{\xi}_j$, is the vector of means for the constraint c_j and $\boldsymbol{\xi}$ contains all the means of all the points for every constraint in $\boldsymbol{c}$. $\hat{\boldsymbol{m}}_j$ and $\hat{\boldsymbol{v}}_j$ hold the mean and variance natural parameters corresponding for all the points in the set $\mathcal{X}^*$.

Once we have obtain the natural parameters $\hat{\boldsymbol{m}}_j$ and $\hat{\boldsymbol{v}}_j$, we obtain the cavity distribution. As we are dealing with natural parameters, it is not necessary to use the formula for the ratio of Gaussian Distributions, the cavity distribution defined by mean $\hat{\boldsymbol{m}}^{\setminus j}$ and variance $\hat{\boldsymbol{v}}^{\setminus j}$ will simply be obtained by the substraction of the natural parameters between the approximated distribution defined by parameters $\hat{\boldsymbol{m}}_{\boldsymbol{j}}$ and $\hat{\boldsymbol{v}}_{\boldsymbol{j}}$ (which is equivalent to the product of all the factors for all the constraints) and the factor $\hat{\boldsymbol{e}}_j$ and $\hat{\boldsymbol{f}}_j$ corresponding to the constraint c_j that we want to update:

$$\begin{aligned} \hat{\boldsymbol{v}}_{nat}^{\setminus j} &= \hat{\boldsymbol{v}}_{\boldsymbol{j}} - \hat{\boldsymbol{e}}_j\,, \\ \hat{\boldsymbol{m}}_{nat}^{\setminus j} &= \hat{\boldsymbol{m}}_{\boldsymbol{j}} - \hat{\boldsymbol{f}}_j\,. \end{aligned} \tag{B.12}$$

Once the substraction is done, we transform the natural parameters of the cavity distribution into Gaussian parameters again by using the formula that converts natural to Gaussian parameters.

$$\begin{aligned} \hat{\boldsymbol{v}}^{\setminus j} &= \frac{1}{\hat{\boldsymbol{v}}_{nat}^{\setminus j}}\,, \\ \hat{\boldsymbol{m}}^{\setminus j} &= \hat{\boldsymbol{m}}_{nat}\hat{\boldsymbol{v}}^{\setminus j}\,. \end{aligned} \tag{B.13}$$

The variances $\hat{\boldsymbol{v}}^{\setminus j}$ need to be positive for the following operations.

B.3.1.2 Computation of the Partial Derivatives of the Normalization Constant

Once the cavities $\hat{\boldsymbol{v}}^{\setminus j}$ and $\hat{\boldsymbol{m}}^{\setminus j}$ have been computed, the EP need to compute the quantities required for the update of the factors $\hat{\boldsymbol{e}}_j$ and $\hat{\boldsymbol{f}}_j$ in order to minimize the KL divergence between $\Phi(\cdot)$ and the approximation distribution. These quantities are the firr st and second moments of the distribution that we want to approximate. These are given by the log of the partial derivatives of Z_j, the constant that normalizes the distribution that we want to approximate, in this case, $\hat{\Phi}(\cdot)$.

$$Z_j = \int \hat{\Phi}(\boldsymbol{x}^*)\, dc_j \,. \tag{B.14}$$

As $\Phi(\boldsymbol{x}^*)$ is approximated by a Gaussian Distribution $\hat{\Phi}(\boldsymbol{x}^*)$ with mean $\hat{\boldsymbol{m}}^{\setminus j}$ and variance $\hat{\boldsymbol{v}}^{\setminus j}$, the normalization constant Z_j can be computed in closed form and its given by the cumulative distribution function ,$\Phi(\cdot)$, of this Gaussian Distribution:

$$Z_j = \Phi\left(\frac{\hat{\boldsymbol{m}}^{\setminus j}}{\sqrt{\hat{\boldsymbol{v}}^{\setminus j}}}\right). \tag{B.15}$$

Let $\alpha = \frac{\hat{\boldsymbol{m}}^{\setminus j}}{\sqrt{\hat{\boldsymbol{v}}^{\setminus j}}}$, then $\log(Z_j) = \log(\Phi(\alpha))$. For numerical robustness, if $a, b \in \mathbb{R}$, we apply the rule $\frac{a}{b} = \exp\left(\log(a) - \log(b)\right)$. Using these expressions, the log-derivatives are computed as follows:

$$\begin{aligned}
\frac{\partial \log(Z_j)}{\partial \hat{\boldsymbol{m}}^{\setminus j}} &= \frac{\exp\{\log(N(\alpha)) - \log(Z_j)\}}{\sqrt{\hat{\boldsymbol{v}}^{\setminus j}}}\,, \\
\frac{\partial \log(Z_j)}{\partial \hat{\boldsymbol{v}}^{\setminus j}} &= -\frac{\exp\{\log(N(\alpha)) - \log(Z_j)\}\alpha}{2\hat{\boldsymbol{v}}^{\setminus j}}\,.
\end{aligned} \tag{B.16}$$

Where $N(\cdot)$ represent the Gaussian probability density function. These expressions are valid for computing the first and second moments, but they do not present numerical robustness in all experiments. Since the lack of robustness of $\frac{\partial \log(Z_j)}{\partial \hat{\boldsymbol{v}}^{\setminus j}}$, we use the formula given by the Appendix A of the work by Opper (Opper and Archambeau, 2009), and use the second partial derivative $\frac{\partial^2 \log(Z_j)}{\partial [\hat{\boldsymbol{m}}^{\setminus j}]^2}$ rather than $\frac{\partial \log(Z_j)}{\partial \hat{\boldsymbol{v}}^{\setminus j}}$. This derivative is given by the following expression:

$$\frac{\partial^2 \log(Z_j)}{\partial [\hat{\boldsymbol{m}}^{\setminus j}]^2} = -\exp\{\log(N(\alpha)) - \log(Z_j)\}\frac{\alpha \exp\{\log(N(\alpha)) - \log(Z_j)\}}{\hat{\boldsymbol{v}}^{\setminus j}}\,. \tag{B.17}$$

Given these derivatives, in the next section it will be explained how to obtain the individual approximate factors $\hat{\boldsymbol{e}}_j$ and $\hat{\boldsymbol{f}}_j$.

B.3.1.3 Computation of the First and Second Moments for the Updates

We now have to compute the first and second moments for the mentioned updates. As the distributions are going to be Gaussian, which belongs to the exponential family, we know that the first and second moment of the Gaussian Distribution are given by:

$$\mathbb{E}[\boldsymbol{x}] = \boldsymbol{\mu}\,, \tag{B.18}$$

$$\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T] = \boldsymbol{\Sigma} + \boldsymbol{\mu}\boldsymbol{\mu}^T\,, \tag{B.19}$$

In order to match the moments, we make the Kullback-Leibler divergence between both distributions zero. With the previous definition of Z_j and given the computed robust derivatives, the expressions that obtain the first and second moments that give the approximate factors $\hat{\boldsymbol{e}}_j$ that represents the variance and $\hat{\boldsymbol{f}}_j$ that represent the mean for the constraint $\boldsymbol{c}_j(\boldsymbol{X}^*)$ are given by the following expressions according to EP:

$$\begin{aligned}\hat{\boldsymbol{f}}_j &= \frac{\frac{\partial \log(Z_j)}{\partial \hat{\boldsymbol{m}}^{\setminus j}} - \hat{\boldsymbol{m}}^{\setminus j}\frac{\partial^2 \log(Z_j)}{\partial [\hat{\boldsymbol{m}}^{\setminus j}]^2}}{1 + \frac{\partial^2 \log(Z_j)}{\partial [\hat{\boldsymbol{m}}^{\setminus j}]^2}\hat{\boldsymbol{v}}^{\setminus j}}\,, \\ \hat{\boldsymbol{e}}_j &= -\frac{\frac{\partial^2 \log(Z_j)}{\partial [\hat{\boldsymbol{m}}^{\setminus j}]^2}}{1 + \frac{\partial^2 \log(Z_j)}{\partial [\hat{\boldsymbol{m}}^{\setminus j}]^2}\hat{\boldsymbol{v}}^{\setminus j}}\,.\end{aligned} \tag{B.20}$$

In practice, the updates are not absolute, they are dumped as the section 5.2 of this Appendix shows.

B.3.2 EP Update Operations for the $\Omega(\cdot,\cdot)$ Factors

Recalling section 3, for the $\mathcal{M}$ Pareto Set points defined by the set $\mathcal{X}^*$ and the $\mathcal{N}$ input space observation points defined by the set $\hat{\mathcal{X}}$, for every pair of points $\boldsymbol{x}_i \in \hat{\mathcal{X}}$ and $\boldsymbol{x}_j \in \mathcal{X}^*$, the EP will generate K two-dimensional gaussian approximations for every objective $\boldsymbol{f}_k$ that will be defined for the pair observation and pareto set point by factors defined by mean $\hat{\boldsymbol{b}}_{ij}$ and variance $\hat{\boldsymbol{A}}_{ij}$ and for the pair of pareto set points by factors defined by mean $\hat{\boldsymbol{d}}_{ij}$ and variance $\hat{\boldsymbol{C}}_{ij}$. It will also define J one-dimensional gaussian approximations for every constraint c_j that will be defined for the pair observation and pareto set point by factors defined by mean $\hat{bc}_j$ and variance $\hat{ac}_j$ and for the pair of pareto set points by factors defined by mean $\hat{dc}_j$ and variance $\hat{cc}_j$. Computations are done for all the pairs of points from the sets $\mathcal{X}^*$ and $\hat{\mathcal{X}}$. The necessary operations for computing these factors are described in the following sections.

B.3.2.1 Computation of the Cavity Distribution

For the factors $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$ that approximate the $\mathcal{J}$ one-dimensional gaussian approximations for every constraint c_j, the operations needed to extract the cavity distribution from the approximate distribution are the same ones as the ones described in Section 4.1.1. These operations are done for the observation points in $\hat{\mathcal{X}}$ for the factors $\hat{ac}_j$, $\hat{bc}_j$ and for the Pareto Set points in $\mathcal{X}^*$ for the factors $\hat{cc}_j$ and $\hat{dc}_j$. That is, obtaining the natural parameters of $\boldsymbol{\Xi}_j$ as in Eq. (16), substracting the natural parameters of the factor that is approximated, Eq. (17), and obtaining the gaussian parameters of the cavity that we define for a point $\boldsymbol{x}_i$, $m_{ij}^{\setminus b_j}$ and $v_{ij}^{\setminus a_j}$, as shown in Eq. (18).

Obtaining the cavity distribution for the factors $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$ and $\hat{\boldsymbol{d}}_{ij}$ that approximate the $\mathcal{K}$ two-dimensional gaussian approximations for every objective $\boldsymbol{f}_k$ follow different expressions as in this case the Gaussian Distributions are bivariate for every pair of points considered.

In the first case, for the case of approximating a distribution that consider a point $\boldsymbol{x}_i$ belonging to the observations set $\hat{\mathcal{X}}$ and a point $\boldsymbol{x}_j$ from the pareto set $\mathcal{X}^*$, that is, the factors $\hat{\boldsymbol{A}}_{ij}$ and $\hat{\boldsymbol{b}}_{ij}$, it is necessary to obtain, for every objective k and each of the pair of points mentioned, the natural parameters $\boldsymbol{m}_{ij(nat)}^k$ and $\boldsymbol{V}_{ij}^{k\,-1}$ of the Gaussian Process that models each of the $\mathcal{K}$ objectives $f(\cdot)_j$. These natural parameters are obtained by

the following expressions:

$$\begin{aligned} \boldsymbol{m}_{ij(nat)}^{k} &= \boldsymbol{V}_{ij}^{k^{-1}} \boldsymbol{m}_{ij}^{k}\,, \\ \boldsymbol{V}_{ij}^{k^{-1}} &= (\boldsymbol{V}_{ij}^{k})^{-1}\,, \end{aligned} \tag{B.21}$$

where $\boldsymbol{V}_{ij}^{k}$ is a 2x2 matrix that represent in the points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ the variance of the gaussian approximation of the objective k and $\boldsymbol{m}_{ij}^{k}$ is a vector that represent in the points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ the mean of the gaussian approximation of the objective k.

As in the constraints case, we now extract the cavity distribution that we define by the natural parameters $\boldsymbol{m}_{ijk(nat)}^{\backslash b}$ and $\boldsymbol{V}_{ijk(nat)}^{\backslash A}$, by substracting to the computed natural parameters $\boldsymbol{m}_{ij(nat)}^{k}$ and $\boldsymbol{V}_{ij}^{k^{-1}}$, computed in the previous step, the factors that we want to update $\boldsymbol{b}_{ij}^{k}$ and $\boldsymbol{A}_{ij}^{k}$. That is:

$$\begin{aligned} \boldsymbol{m}_{ijk(nat)}^{\backslash b} &= \boldsymbol{m}_{ij(nat)}^{k} - \boldsymbol{b}_{ij}^{k}\,, \\ \boldsymbol{V}_{ijk(nat)}^{\backslash A} &= \boldsymbol{V}_{ij}^{k^{-1}} - \boldsymbol{A}_{ij}^{k}\,. \end{aligned} \tag{B.22}$$

For the bivariate gaussian distribution, the step of obtaining the gaussian parameters from the natural parameters is defined by the following expressions:

$$\begin{aligned} \boldsymbol{m}_{ijk}^{\backslash b} &= \boldsymbol{V}_{ijk}^{\backslash A}\ \boldsymbol{m}_{ijk(nat)}\,, \\ \boldsymbol{V}_{ijk}^{\backslash A} &= (\boldsymbol{V}_{ijk(nat)}^{\backslash A})^{-1}\,, \end{aligned} \tag{B.23}$$

where $\boldsymbol{V}_{ijk}^{\backslash A}$ is a 2x2 matrix with the variances of each of the points and the correlation between each of them and $\boldsymbol{m}_{ijk}^{\backslash b}$ is a two position vector that represent the means. In the case of the factors $\hat{\boldsymbol{C}}_{ij}$ and $\hat{\boldsymbol{d}}_{ij}$ that consider two Pareto Set points, the operations for extracting the cavity distribution are the same ones as in the previous case.

B.3.2.2 Computation of the Partial Derivatives of the Normalization Constant

In this section, the operations needed to compute the partial derivatives for all the $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$, $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$ are described. These derivatives need previous computations in order to compute the normalization constant Z_Ω of the factor $\Omega(\cdot,\cdot)$ that we want to approximate. These computations are given by the following expressions, all of which depend upon terms computed in the previous section. The shown computations are the result of applying rules in order to be robust such as $a/b = \exp\{\log(a) - \log(b)\}$ and $ab = \exp\{\log(a) + \log(b)\}$.These operations are equivalent for the two points cases, but here, the necessary operations for computing the normalization constant Z_Ω are

described for the case of the factors $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{ac}_j$ and $\hat{bc}_j$:

$$\boldsymbol{s}_k = \boldsymbol{V}^{\setminus A}_{ijk[0,0]} + \boldsymbol{V}^{\setminus A}_{ijk[1,1]} - 2\boldsymbol{V}^{\setminus A}_{ijk[0,1]}\,, \tag{B.24}$$

$$\boldsymbol{\alpha}_k = \frac{\boldsymbol{m}^{\setminus b}_{ijk[0]} - \boldsymbol{m}^{\setminus b}_{ijk[1]}}{\sqrt{s_k}}\,, \tag{B.25}$$

$$\boldsymbol{\beta}_j = \frac{m^{\setminus b_j}_{ij}}{\sqrt{v^{\setminus a_j}_{ij}}}\,, \tag{B.26}$$

$$\boldsymbol{\phi} = \Phi(\boldsymbol{\alpha})\,, \tag{B.27}$$

$$\tag{B.28}$$

where $\Phi(\cdot)$ represents the c.d.f of a Gaussian distribution,

$$\boldsymbol{\gamma} = \Phi(\boldsymbol{\beta})\,, \tag{B.29}$$

$$\zeta = 1 - \exp\{\sum_{k=1}^{K} \log(\boldsymbol{\phi}_k)\}\,, \tag{B.30}$$

$$\log(\eta) = \sum_{j=1}^{J} \log(\boldsymbol{\gamma}_j) + \log(\zeta)\,, \tag{B.31}$$

$$\lambda = 1 - \exp\{\sum_{j=1}^{J} \log(\boldsymbol{\gamma}_j)\}\,, \tag{B.32}$$

$$\tau = \max(\log(\eta), \log(\lambda))\,, \tag{B.33}$$

$$\log(Z_\Omega) = \log(\exp\{\log(\eta) - \tau\} + \exp\{\log(\lambda) - \tau\}) + \tau\,. \tag{B.34}$$

Having computed these terms, the log partial derivatives for the update of the factors that collaborate to the approximation of the objective variances $\hat{\boldsymbol{A}}_{ij}$ and the objective means $\hat{\boldsymbol{b}}_{ij}$ are given by the expressions:

$$\boldsymbol{\rho}_k = -\exp\{\log(\mathcal{N}(\boldsymbol{\alpha}_k))\} - \log(Z_\Omega) + \sum_{k=1}^{K}\{\log(\Phi(\boldsymbol{\alpha}_k))\} - \log(\Phi(\boldsymbol{\alpha}_k)) + \sum_{j=1}^{J}\{\log(\Phi(\boldsymbol{\beta}_j))\}\,, \tag{B.35}$$

$$\frac{\partial \log(Z_\Omega)}{\partial \boldsymbol{m}^{\setminus b}_{ijk}} = \frac{\boldsymbol{\rho}_k}{\sqrt{\boldsymbol{s}_k}}\big[1, -1\big]\,,$$

$$\frac{\partial \log(Z_\Omega)}{\partial \boldsymbol{V}^{\setminus A}_{ijk}} = -\frac{\boldsymbol{\rho}_k \boldsymbol{\alpha}_k}{2\boldsymbol{s}_k}[[1, -1], [-1,\ 1]]\,. \tag{B.36}$$

Derivatives are computed for the two position vector mean and the 2x2 variance matrix, so they have the same structure, given by the [1, -1] and [[1, -1],[-1, 1]] expressions. The change in the sign appears due to the fact that the expression changes, whether it is the derivative of the mean of the observation point or the Pareto Set point or the derivative of the variance of one point or their correlation.

Alas, the derivative of the variance presents the same lack of robustness as in the constraint case shown in section 4.1.2. In order to ensure numerical robustness, we use the second partial derivative of the mean of the normalization constant instead of the

first partial derivative of the variance for the further computation of the second moment. That is,

$$\frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\setminus b}\right]^2} = -\frac{\boldsymbol{\rho}_k}{\boldsymbol{s}_k}(\boldsymbol{\alpha}_k + \boldsymbol{\rho}_k)[[1,-1],[-1,\ 1]]\,. \tag{B.37}$$

For the log partial derivatives for the update of the factors that collaborate to the approximation of the constraint variances $\hat{ac}_j$ and the constraint means $\hat{bc}_j$, let $\boldsymbol{\omega}_j$ be defined as:

$$\begin{aligned}\boldsymbol{\omega}_j =& \exp\{\log(\mathcal{N}(\boldsymbol{\beta}_j))\} - \log(Z_\Omega) + \log(\zeta) + \sum_{j=1}^{J}(\log(\Phi(\boldsymbol{\beta}_j))) - \log(\Phi(\boldsymbol{\beta}_j)) - \exp\{\log(\mathcal{N}(\boldsymbol{\beta}_j))\}\,,\\ &- \log(Z_\Omega) + \sum_{j=1}^{J}(\log(\Phi(\boldsymbol{\beta}_j))) - \log(\Phi(\boldsymbol{\beta}_j))\,.\end{aligned} \tag{B.38}$$

Then, the robust log partial derivatives for the first and the second moments are given by the expressions:

$$\begin{aligned}\frac{\partial \log(Z_\Omega)}{\partial m_{ij}^{\setminus \boldsymbol{b}_j}} &= \frac{\boldsymbol{\omega}_j}{\sqrt{\boldsymbol{s}_j}}\,,\\ \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ij}^{\setminus \boldsymbol{b}_j}]^2} &= -\frac{\boldsymbol{\omega}_j}{\boldsymbol{s}_j}\,(\boldsymbol{\beta}_j + \boldsymbol{\omega}_j)\,.\end{aligned} \tag{B.39}$$

The expressions for the log partial derivatives of $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$, $\hat{cc}_j$ and $\hat{dc}_j$ are similar to the presented expressions in this section, but taking into account pairs of points belonging to the set $\mathcal{X}^*$.

B.3.2.3 Computation of the First and Second Moments for the Updates

Giving the expressions computed in the previous section, the first and second moments of the different Gaussian Distributions that approximate the factor $\Omega(\cdot,\cdot)$ can now be computed.

The expressions for computing the factors $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$ for each of the K objectives and the factors $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$ for each of the J constraints are the following ones:

$$\hat{\boldsymbol{A}}_{ij}^k = \frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\setminus b}\right]^2}\ ((\boldsymbol{V}_{ijk}^{\setminus A}\frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\setminus b}\right]^2})^{-1}[[1,0],[0,1]])\,, \tag{B.40}$$

$$\hat{\boldsymbol{b}}_{ij}^k = ((\frac{\partial \log(Z_\Omega)}{\partial \boldsymbol{m}_{ijk}^{\setminus b}} - \boldsymbol{m}_{ijk}^{\setminus b})\frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\setminus b}\right]^2})\ ((\boldsymbol{V}_{ijk}^{\setminus A}\frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\setminus b}\right]^2})^{-1} + [[1,0],[0,1]])\,, \tag{B.41}$$

$$\hat{\boldsymbol{C}}_{ij}^k = \frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\setminus b}\right]^2}\ ((\boldsymbol{V}_{ijk}^{\setminus A}\frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\setminus b}\right]^2})^{-1}[[1,0],[0,1]])\,, \tag{B.42}$$

$$\hat{\boldsymbol{d}}_{ij}^k = ((\frac{\partial \log(Z_\Omega)}{\partial \boldsymbol{m}_{ijk}^{\setminus b}} - \boldsymbol{m}_{ijk}^{\setminus b})\frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\setminus b}\right]^2})\ ((\boldsymbol{V}_{ijk}^{\setminus A}\frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\setminus b}\right]^2})^{-1} + [[1,0],[0,1]])\,, \tag{B.43}$$

for the the rest of the factors, suppose that the index h refers to the points of the Pareto Set $\mathcal{X}^*$:

$$\hat{ac}_h^j = -\frac{\frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2}}{1 + \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2} v_{ic}^{\backslash a_j}}, \tag{B.44}$$

$$\hat{bc}_h^j = \frac{\frac{\partial \log(Z_\Omega)}{\partial m_{ic}^{\backslash \boldsymbol{b}_j}} - m_{ic}^{\backslash b_j} \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2}}{1 + \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2} v_{ic}^{\backslash a_j}}, \tag{B.45}$$

$$\hat{cc}_h^j = -\frac{\frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2}}{1 + \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2} v_{ic}^{\backslash a_j}}, \tag{B.46}$$

$$\hat{dc}_h^j = \frac{\frac{\partial \log(Z_\Omega)}{\partial m_{ic}^{\backslash \boldsymbol{b}_j}} - m_{ic}^{\backslash b_j} \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2}}{1 + \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2} v_{ic}^{\backslash a_j}}. \tag{B.47}$$

All these factors are then used to rebuild the means and the variances of the Gaussian Processes that model the K objectives and C constraints of a constrained multi-objective optimization problem, as will be shown in the following section. That is, C one-dimensional Gaussian Distributions for the constraint models and C one-dimensional Gaussian Distributions and K two-dimensional Gaussian Distributions for the objective models in each of the points in $\mathcal{X} = \{\mathcal{X}^* \cup \hat{\mathcal{X}} \cup \boldsymbol{x}\}$.

B.3.3 Reconstruction of the Conditional Predictive Distribution

In this section, we illustrate the way of obtaining a Conditional Predictive Distribution for every objective f_k and every constraint c_j, given a sampled Pareto Set $\mathcal{X}^* = \{\boldsymbol{x}_1^*, ..., \boldsymbol{x}_M^*\}$ of size M and a set of N input locations $\hat{\mathcal{X}} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ with corresponding observations of the k-th objective $\boldsymbol{y}_k$ and of the j-th constraint $\boldsymbol{y}_j$. For the following, it is assumed that we are given the EP approximate factors $\Phi(\cdot)$ and $\Omega(\cdot, \cdot)$, as an input for the next operations, which computation is explained in the previous section.

Recalling Eqs. 7, 8 and 9 of section 2, we want to obtain the J Conditional Predictive Distributions in the products of constraints and the K Conditional Predictive Distributions of the Gaussian Processes that model the objectives. The products presented in these factors are not a problem, due to the fact that the Gaussian Distributions are closed under the product operation, that is, the product of Gaussian Distributions is another Gaussian Distribution. These Conditional Predictive Distributions of the objectives and constraints are then used in Eq.(11) to build the final approximation.

Following the notation of section 4.1.1, let $\boldsymbol{\xi}_j$ and $\boldsymbol{\Xi}_j$ be the mean vector and variance matrix of the one-dimensional Gaussian Distributions of the $M + N$ points that generate the Gaussian Processes that model the constraints and let $\boldsymbol{m}_k$ and $\boldsymbol{V}_k$ be the mean vector and variance matrix of the two-dimensional Gaussian Distributions of the $M + N$ points that generate the Gaussian Processes that model the objectives. In order to update the constraint and objective distribution marginals, it is necessary to first follow the operations given by the equations 14 and 22, to obtain the natural parameters from the means and variances. Intuitively, as they are all natural parameters, these will be just sums taking into account that the matrices are formed first by the Pareto Set Points ,M,

and then by the observations N. Univariate factors are added to the diagonal of these matrices, as they are not correlated with other points. Once the natural parameters are computed, the new means $\boldsymbol{\xi}_j$, $\boldsymbol{m}_k$ and variances $\boldsymbol{\Xi}_j$, $\boldsymbol{V}_k$ marginals are updated from the EP factors $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$, $\hat{e}_j$, $\hat{f}_j$, $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$ by the following expressions:

$$
\begin{aligned}
\boldsymbol{\Xi}_{ii}^{j} &= \boldsymbol{\Xi}_{ii(old)}^{j} + \sum_{m=1}^{M} \hat{cc}_{mi}^{j} + \hat{e}_i^j && \text{for } \; i = 1, ..., M\,, \\
\boldsymbol{\Xi}_{ii}^{c} &= \boldsymbol{\Xi}_{ii(old)}^{j} + \sum_{m=1}^{M} \hat{ac}_{mi}^{j} && \text{for } \; i = M+1, ..., N+M\,, \\
\boldsymbol{\xi}_{i}^{c} &= \boldsymbol{\xi}_{i(old)}^{j} + \sum_{m=1}^{M} \hat{dc}_{mi}^{j} + \hat{f}_i^j && \text{for } \; i = 1, ..., M\,, \\
\boldsymbol{\xi}_{i}^{c} &= \boldsymbol{\xi}_{i(old)}^{j} + \sum_{m=1}^{M} \hat{bc}_{mi}^{j} && \text{for } \; i = M+1, ..., N+M\,, \\
\boldsymbol{V}_{ii}^{k} &= \boldsymbol{V}_{ii(old)}^{k} + \sum_{j=M+1}^{N} \hat{\boldsymbol{A}}_{ji[1,1]}^{k} + \sum_{j=1}^{M} \hat{\boldsymbol{C}}_{ij[0,0]}^{k} + \sum_{j=1}^{M} \hat{\boldsymbol{C}}_{ji[1,1]}^{k} && \text{for } \; i = 1, ..., M\,, \\
\boldsymbol{V}_{ii}^{k} &= \boldsymbol{V}_{ii(old)}^{k} + \sum_{j=1}^{M} \hat{\boldsymbol{A}}_{ij[0,0]}^{k} && \text{for } \; i = M+1, ..., N+M\,, \\
\boldsymbol{V}_{ij}^{k} &= \boldsymbol{V}_{ij(old)}^{k} + \boldsymbol{C}_{ij[0,1]}^{k} + \boldsymbol{C}_{ij[1,0]}^{k\,T} && \text{for } \; i = 1, ..., M, \text{ and for } \; j = 1, ..., M\,, \\
\boldsymbol{V}_{ij}^{k} &= \boldsymbol{V}_{ij(old)}^{k} + \boldsymbol{A}_{ij[0,1]}^{k} && \text{for } \; i = M+1, ..., N, \text{ and for } \; j = 1, ..., M\,, \\
\boldsymbol{V}_{ij}^{k} &= \boldsymbol{V}_{ij(old)}^{k} + \boldsymbol{A}_{ij[0,1]}^{k\,T} && \text{for } \; i = 1, ..., M, \text{ and for } \; j = M+1, ..., N\,, \\
\boldsymbol{m}_{i}^{k} &= \boldsymbol{m}_{i(old)}^{k} + \sum_{j=M+1}^{N+M} \hat{\boldsymbol{b}}_{ji[1]}^{k} + \sum_{j=1}^{M} \hat{\boldsymbol{d}}_{ij[0]}^{k} + \sum_{j=1}^{M} \hat{\boldsymbol{d}}_{ji[1]}^{k} && \text{for } \; i = 1, ..., M\,, \\
\boldsymbol{m}_{i}^{k} &= \boldsymbol{m}_{i(old)}^{k} + \sum_{j=1}^{M} \hat{\boldsymbol{b}}_{ij[0]}^{k} && \text{for } \; i = M+1, ..., N+M\,.
\end{aligned}
\tag{B.48}
$$

These natural parameters are then converted into Gaussian ones using the equations and 16 and 24. Once these operations are done the Gaussian Processes that model the objectives and constraints are updated from a full EP iteration.

B.3.4 The Conditional Predictive Distribution at a New Point

In this section, the computation of the conditional distributions for every model of the objectives f_k and the constraints c_j at a new candidate location $\boldsymbol{x}_{N+1}$ is explained. This requires that q($\boldsymbol{f}$,$\boldsymbol{c}$) is already computed approximated by the factors of EP. The interest lies in evaluating the conditional predictive variances for $f_k(\boldsymbol{x}_{N+1})$ and for $c_j(\boldsymbol{x}_{N+1})$ for every objective and constraint. With this variances, it is possible to compute the PESMOC acquisition function given by the Negative Differential Entropy w.r.t the

Conditional Predictive Distribution and the Predictive Distribution:

$$\alpha(\mathbf{x}) \approx \sum_{j=1}^{J} \log v_j^{\mathrm{PD}}(\mathbf{x}) + \sum_{k=1}^{K} \log v_k^{\mathrm{PD}}(\mathbf{x}) - \frac{1}{M} \sum_{m=1}^{M} \Big[\sum_{j=1}^{J} \log v_j^{\mathrm{CPD}}(\mathbf{x}|\mathcal{X}^{\star}_{(m)}) + \sum_{k=1}^{K} \log v_k^{\mathrm{CPD}}(\mathbf{x}|\mathcal{X}^{\star}_{(m)}) \Big] . \tag{B.49}$$

In order to obtain these variances, we need to execute the Expectation Propagation algorithm once again, but as it is only a new observation point, it is not necessary to compute the $\Phi(\cdot)$ factor. The Conditional Predictive Distribution variances expressions that we need to evaluate in order to obtain these variances are:

$$p(f_k(\boldsymbol{x}_{N+1}))|\hat{\mathcal{X}}, \mathcal{X}^*, \boldsymbol{x}_{N+1}) \approx \int Z_k^{-1} q(\boldsymbol{f}_k, \boldsymbol{f}_k(\boldsymbol{x}_{N+1})) \prod_{\mathbf{x}^{\star} \in \mathcal{X}^{\star}} \Big(\Omega(\mathbf{x}_{N+1}, \mathbf{x}^{\star}) \Big) d\boldsymbol{f}_k, d\boldsymbol{c}_j \,, \tag{B.50}$$

$$p(c_j(\boldsymbol{x}_{N+1}))|\hat{\mathcal{X}}, \mathcal{X}^*, \boldsymbol{x}_{N+1}) \approx \int Z_j^{-1} q(\boldsymbol{c}_j, \boldsymbol{c}_j(\boldsymbol{x}_{N+1})) \prod_{\mathbf{x}^{\star} \in \mathcal{X}^{\star}} \Big(\Omega(\mathbf{x}_{N+1}, \mathbf{x}^{\star}) \Big) d\boldsymbol{f}_k, d\boldsymbol{c}_j \,, \tag{B.51}$$

where Z is a normalization constant and $q(\boldsymbol{f}, \boldsymbol{f}(\boldsymbol{x}_{N+1}))$ and $q(\boldsymbol{c}, \boldsymbol{c}(\boldsymbol{x}_{N+1}))$ are multivariate gaussian distributions that results by extending $q(\boldsymbol{f})$ and $q(\boldsymbol{c})$ with the new point $\boldsymbol{x}_{N+1}$. We have only taken into account the approximate factors that depend on f_k and c_j. The covariances between $\boldsymbol{f}_k$ and $f_k(\boldsymbol{x}_{N+1})$ and the covariances between $\boldsymbol{c}_j$ and $c_j(\boldsymbol{x}_{N+1})$ are obtained from the GP posteriors for f_k and c_j given the observed data. In the same way, the mean and the variance of $f_k(\boldsymbol{x}_{N+1})$ and $c_j(\boldsymbol{x}_{N+1})$ can be obtained. As all the factors are Gaussian and the product operation is closed the results are also Gaussian Distributions.

Let $\tilde{\boldsymbol{f}}_k = (f_k(\boldsymbol{x}_1^*), ..., f_k(\boldsymbol{x}_M^*), f_k(\boldsymbol{x}_{N+1}^*))^T$ and $\tilde{\mathbf{c}}_j = (c_j(\boldsymbol{x}_1^*), ..., c_j(\boldsymbol{x}_M^*), c_j(\boldsymbol{x}_{N+1}^*))^T$. As $\prod_{\mathbf{x}^{\star} \in \mathcal{X}^{\star}}$
$(\Omega(\mathbf{x}_{N+1}, \mathbf{x}^{\star}))$ does not depend on $f_k(\boldsymbol{x}_1, ..., \boldsymbol{x}_N)$ nor in $c_j(\boldsymbol{x}_1, ..., \boldsymbol{x}_N)$, we can marginalize these variables in Eq. (55) to obtain Gaussian Distributions for every objective k and every constraint j in the point $\boldsymbol{x}_{N+1}$, proportional to:

$$\int q(\tilde{\boldsymbol{f}}_k) \prod_{\mathbf{x}^{\star} \in \mathcal{X}^{\star}} \Big(\Omega(\mathbf{x}_{N+1}, \mathbf{x}^{\star}) \Big) \prod_{i=1}^{M} df_k(\boldsymbol{x}_i^*) \,, \tag{B.52}$$

$$\int q(\tilde{\mathbf{c}}_j) \prod_{\mathbf{x}^{\star} \in \mathcal{X}^{\star}} \Big(\Omega(\mathbf{x}_{N+1}, \mathbf{x}^{\star}) \Big) \prod_{i=1}^{J} dc_j(\boldsymbol{x}_i^*) \,, \tag{B.53}$$

these expressions generate Gaussian Distributions, one for every objective and one for every constraint, at the point $\boldsymbol{x}_{N+1}$:

$$\int \mathcal{N}(\tilde{\boldsymbol{f}}_k|(\boldsymbol{V}^x)^{-1}\boldsymbol{m}^x, (\boldsymbol{V}^x)^{-1}) \prod_{i=1}^{M} df_k(\boldsymbol{x}_i^*) = \mathcal{N}(f_k(\boldsymbol{x}_{N+1})|m_k, v_k) \,, \tag{B.54}$$

$$\int \mathcal{N}(\tilde{\mathbf{c}}_j|(\boldsymbol{\Xi}^x)^{-1}\boldsymbol{\xi}^x, (\boldsymbol{\Xi}^x)^{-1}) \prod_{j=1}^{J} dc_j(\boldsymbol{x}_i^*) = \mathcal{N}(c_j(\boldsymbol{x}_{N+1})|\xi_j, v_j) \,, \tag{B.55}$$

where $\boldsymbol{m}^x$, $\boldsymbol{V}^x$, $\boldsymbol{\xi}^x$ and $\boldsymbol{\Xi}^x$ are the natural parameters of the Conditional Predictive Distributions for the objective k and the constraint j, which are Gaussian. The variances

v_k and v_j of the Gaussian Approximations of the objectives and the constraints are the ones needed to the entropy computation in Eq.(54). These are given by the last diagonal of the natural parameter variance matrices $\boldsymbol{V}^x$ and $\boldsymbol{\Xi}^x$. As the means are not needed, no further details are here given in order to compute them.

Each entry in $\boldsymbol{\Xi}^x$ and $\boldsymbol{V}^x$, as they are natural parameters, is given by the reconstruction of the predictive distribution, which is the most expensive part:

$$
\begin{aligned}
&\Xi^x_{i,j} = \Xi^c_{i,j} \quad \text{for} \quad 1 \leq i \leq M \quad \text{and} \quad 1 \leq j \leq M\,, \quad \text{and} \quad i \neq j\,,\\
&V^x_{i,j} = V^k_{i,j} \quad \text{for} \quad 1 \leq i \leq M \quad \text{and} \quad 1 \leq j \leq M\,, \quad \text{and} \quad i \neq j\,,\\
&V^x_{i,j} = \text{cov}(f_k(\mathbf{x}_{N+1}), f(\mathbf{x}^*_j)) + \tilde{c}_{N+1,j,k} \quad \text{for} \quad 1 \leq j \leq M\,, \quad \text{and} \quad i = M+1\,,\\
&V^x_{j,i} = V^x_{i,j} \quad \text{for} \quad j \neq i\,, \quad \text{and} \quad 1 \leq i\,, j \leq M\,,\\
&V^x_{i,i} = V^k_{i,i} + \tilde{v}^*_{N+1,j,k}\,, \quad \text{for} \quad 1 \leq i \leq M\,,\\
&\Xi^x_{M+1,M+1} = \text{var}(c_j(\mathbf{x})) + \sum_{j=1}^{M} \tilde{s}_{N+1,j,c}\,,\\
&V^x_{M+1,M+1} = \text{var}(f_k(\mathbf{x})) + \sum_{j=1}^{M} \tilde{v}_{N+1,j,k}\,.
\end{aligned}
\tag{B.56}
$$

where $\tilde{v}_{N+1,j,k}$, $\tilde{v}^*_{N+1,j,k}$, $\tilde{c}_{N+1,j,k}$ and $\tilde{s}_{N+1,j,c}$ are the parameters of each of the M factors $\Omega(\mathbf{x}_{N+1}, \mathbf{x}^\star)$, for $j = 1, \ldots, M$. The other terms, $\text{var}(f_k(\mathbf{x}))$, $\text{var}(c_j(\mathbf{x}))$ and $\text{cov}(f_k(\mathbf{x}_{N+1}), f(\mathbf{x}^*_j))$ are the posterior variances of $f_k(\mathbf{x}_{N+1})$, $c_j(\mathbf{x}_{N+1})$ and the posterior covariance between $f_k(\mathbf{x}_{N+1})$ and $f_k(\mathbf{x}^*_j)$.

The matrix $\mathbf{V}^x$, has a block structure in which only the last row and column depends on $\mathbf{x}_{N+1}$. This fact allows us to compute $v_k = (\mathbf{V}^x)^{-1}_{M+1,M+1}$ with cost $\mathcal{O}(M^3)$ using the expressions for block matrix inversion. All these computations are carried out using the open-BLAS library for linear algebra operations.

Given the variances v_k and v_j, the only task remaining is adding the variance of the additive Gaussian noise ϵ^k_{N+1} and ϵ^j_{N+1} to obtain the final variance of the Gaussian approximations to the conditional predictive distributions of $y^k_{N+1} = f_k(\boldsymbol{x}_{N+1})$ and $y^c_{N+1} = c_j(\boldsymbol{x}_{N+1})$.

B.4 Final Gaussian Approximation to the Conditional Predictive Distribution

B.4.1 Initialization and convergence of EP

When the EP algorithm computes the $\Phi(\cdot)$ and $\Omega(\cdot,\cdot)$ factors, it requires to set an initial value to all the factors that generates the Gaussians that approximate the $\Phi(\cdot)$ and $\Omega(\cdot,\cdot)$ factors. These factors, $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$, $\hat{e}_j$, $\hat{f}_j$, $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$ are all set to be zero. The convergence criterion for stopping the EP algorithm updating the parameters is that the absolute change in all the cited parameters should be below 10^{-4}. Other criteria may be used.

B.4.2 Parallel EP Updates and Damping

The updates of every approximate factor $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$, $\hat{e}_j$, $\hat{f}_j$, $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$ are executed in parallel as it is described in the work by Gerven (Gerven et al., 2009). The

cavity distribution for each of the factors is computed and then the factors are updated afterwards. Once these operations are done the EP approximation is recomputed as it is described in the section 4.3.

In order to improve the convergence behaviour of EP we use the damping technique described in Minka & Lafferty (Gelman et al., 2014). We use this technique for all the approximate factors. Damping simply reduces the quantity that the factor changes in every update as a linear combination between the old parameters and the new parameters. That is, if we define the old parameters of the factor to be updated as u_{old}, the new parameters as u_{new} and the updated factor as u, then the update expression is:

$$u = \theta u_{new} + (1 - \theta) u_{old} \,. \tag{B.57}$$

Where θ is the damping factor whose initial value is set to be 0.5, this factor controls the amount of damping, if this value is set to be one then no damping is employed. This factor is multiplied by 0.99 at each iteration, reducing the amount of change in the approximate factors in every iteration of the Bayesian Optimization. An issue that happens during the optimization process is that some covariance matrices become non positive definite due to a high large step size, that is, a high value of θ. If this happens in any iteration, an inner loop executes again the update operation with $\theta_{new} = \theta_{old}$ / 2 and the iteration is repeated. This inner loop is performed until the covariance matrices become non positive definite.

B.5 Sensitivity Analysis of the Sampled Pareto Set Size

In this section, we present the details of a sensitivity analysis of PESMOC with respect to the number of points included in each sample of the Pareto set $\mathcal{X}^{\star}_{(m)}$. We have considered the 6-dimensional synthetic problem described in the main manuscript. In this problem we have 6 black-boxes that are sampled from a GP prior. From these, 4 are objectives and 2 are constraints. We performed 100 experiments in which we evaluate 100 times the black-boxes and report average results. We consider different set sizes for $\mathcal{X}^{\star}_{(m)}$. Namely, 5, 10, 25, 50, 75 and 100. We include a figure that shows the average relative difference in log scale of the hyper-volume of the recommendation made w.r.t. the hypervolume of the actual soulution, at each iteration of the optimization process, for each Pareto set size. Furthermore, Table B.1 shows the average time required to compute the next evaluation using PESMOC, for each different size of the Pareto set $\mathcal{X}^{\star}$ considered.

TABLE B.1: Average time in seconds required to choose the next evaluation for each size of the Pareto Set sample $\mathcal{X}^{\star}_{(m)}$. These times do not include the GP fit, in contrast with the average times of the 4 dimensional problem scenario shown in Section 4.2 of the main manuscript.

	Time	
Size of $\mathcal{X}^{\star}_{(m)}$	**Mean**	**Standard Deviation**
5	95.98	10.67
10	99.05	11.53
25	109.48	10.92
50	133.02	11.68
75	159.90	15.34
100	190.25	25.71

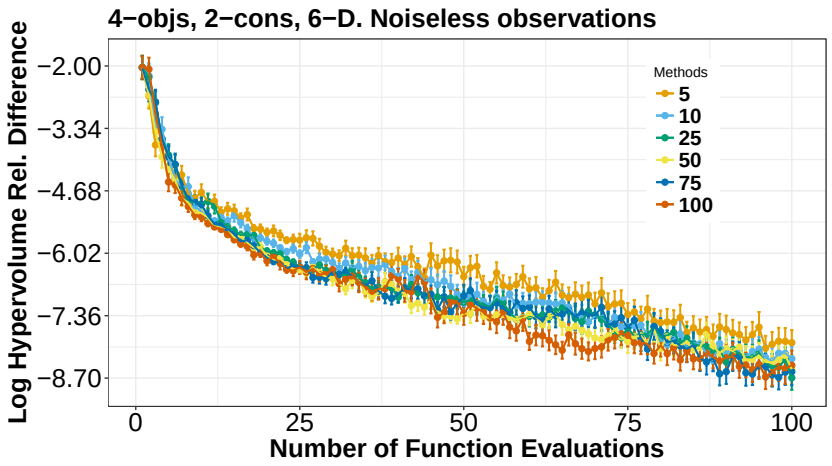

FIGURE B.1: Performance of PESMOC in the synthetic experiment for different Pareto set sizes.

We observe that by including more points in the Pareto Set, in general, the reuslts of PESMOC improve, as expected. However, after including around 25 points the results saturate and no further improvement in performance is observed. In general one should include as many points as possible in each $\mathcal{X}^{\star}_{(m)}$. However, including more points in each sample $\mathcal{X}^{\star}_{(m)}$ also increases the computational cost, as described in Table B.1. We observe that choosing 50 points shows a good trade-off between performance and computational time. In particular, this value gives good results and is not significantly more expensie than considering 25 points.

B.6 Sensitivity Analysis of the Number of Montecarlo Iterations

In this section, we present the details of a sensitivity analysis of PESMOC with respect to the number of Montecarlo iterations considered for sampling the hyperparameters of the Gaussian Processes. We have considered the 4-dimensional synthetic problem described in the main manuscript. In this problem we have 4 black-boxes that are sampled from a GP prior. From these, 2 are objectives and 2 are constraints. We performed 100 experiments in which we evaluate 100 times the black-boxes and report average results. We consider a different number of Montecarlo iterations. Namely, 2, 5, 10, 20, 30, 50, 80 and 100. We include a figure that shows the average relative difference in log scale of the hyper-volume of the recommendation made w.r.t. the hypervolume of the actual soulution, at each iteration of the optimization process, for each number of Montecarlo iterations.

We observe that, for a small number of Montecarlo iterations, in general adding more iterations make the reuslts of PESMOC improve, as expected. However, after including around 10 iterations the results saturate and no further improvement in performance is observed. In general one should include as many number of Montecarlo iterations as possible. However, including more iterations also increases the computational cost, so we observe that choosing 10 iterations shows a good trade-off between performance and computational time.

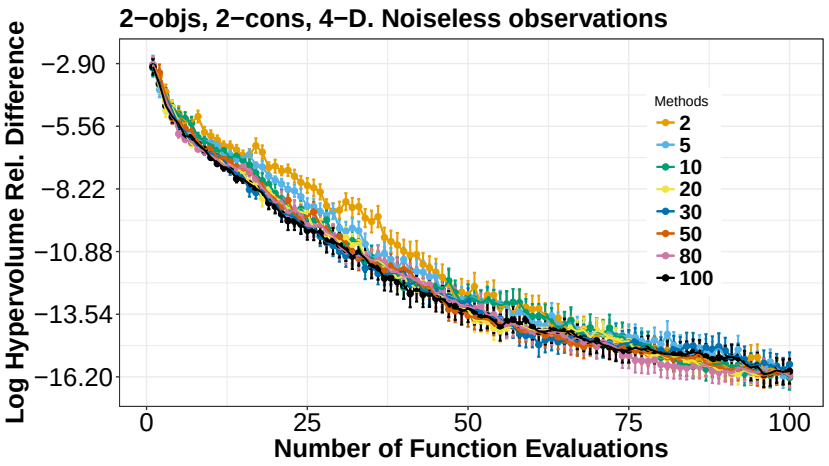

FIGURE B.2: Performance of PESMOC in the synthetic experiment for different number of Montecarlo iterations.

B.7 Percentage of Infeasible Solutions in Benchmark Experiments

We show in the next figure the percentage of infeasible solutions for the different benchmark experiments shown in the main manuscript. The plots show the percentage of infeasible solutions for every iteration of Bayesian Optimization on these problems for every tested acquisition function. A higher percentage of infeasible solutions implies that the suggestions provided by the Bayesian Optimization criterion were not located in the feasible region and hence, are poorer that feasible ones. We remind that to be an infeasible point means that the probability of satisfying any of the constraints in that point was lower in at least one of them than 1 minus δ, typically set to 0.05 in these experiments. This fact may occur, for example for the PESMOC criteria, because the criterion has selected a point that may be close to the border of the feasible region and may be a good point in order to gain information about the Pareto Set, but unfortunately, is infeasible as it lies in the infeasible side of the border, same analogy can be applied for BMOO, which can suggest an infeasible point if it considers it good for optimizing the multiobjective problem. In the random case this does not apply, as the approach performs pure exploration. At the beggining, the infeasibility percentage of solutions is higher due to he fact that the shape of the constraints is unknown and all the criteria are basically exploring the space, hence, they suggest infeasible solutions with a high probability until the shape of the constraints is reasonably known so the approaches can take the constraints into account for suggesting new points. Then, the infeasibility percentage drops down as iterations are computed.

It is shown that, in average, both PESMOC approaches outperform the other alternatives of multi-objective constrained optimization in infeasibility percentage. The TNK problem has the particular feature that it contains an optimum near one of its constraints, that is the reason why the infeasibility grows around iteration 20 in all the approaches. This is because as the optimum is more located and the constraint is defined, the methods tend to search in that area, leaving unfeasible results. Once the constraint is defined, they do not search outside of the Feasible Space anymore, and hence, the unfeasible results dissapear. TwoBarTruss problem has the same nature as TNK, with

the optimum lying in the frontier of the feasible and infeasible space. We, once again, see how PESMOC decoupled explores massively this area, giving lots of infeasible results in iterations 20 to 30. PESMOC decoupled, as it evaluates every black box separately, learns the shape of these constraints and, as the main manuscript states, it delivers a better solution that the other approaches in less evaluations once it knows the shape of these constraints and it is able to suggest points inside of the feasible space and close to the Pareto set.

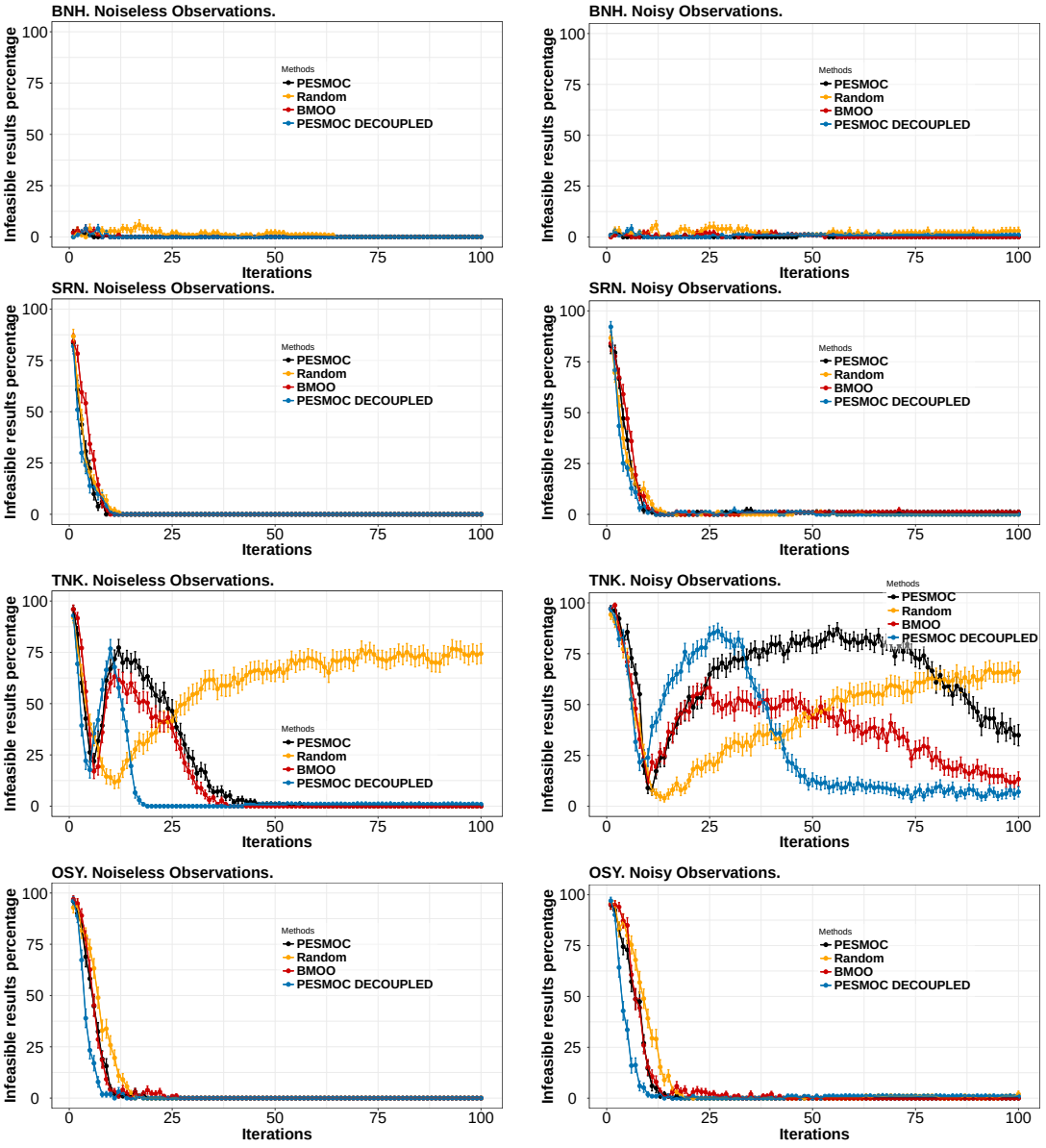

FIGURE B.3: Percentage of infeasible results in every iteration of experiments BNH, SRN, TNK and OSY.

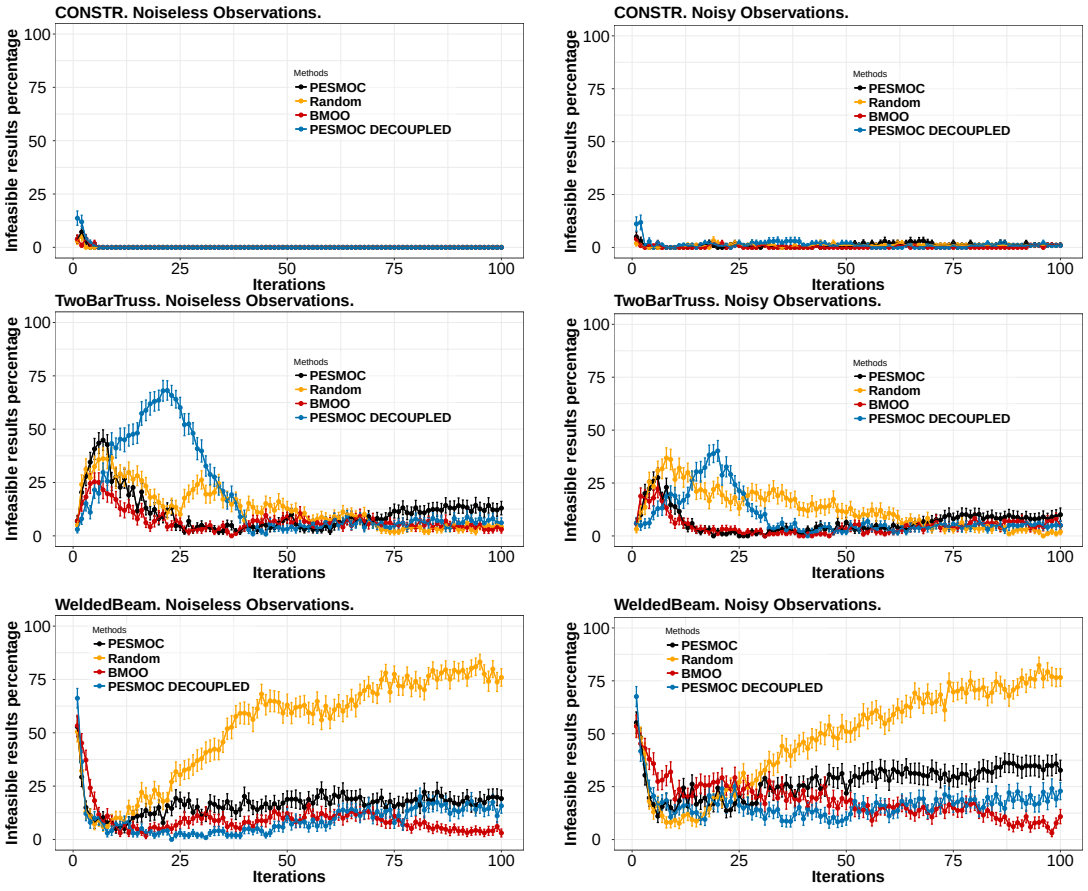

FIGURE B.4: Percentage of infeasible results in every iteration of experiments CONSTR, TwoBarTruss and WeldedBeam.

Appendix C

Appendix for Chapter 5

In this Appendix, we give additional information about PPESMOC. In particular, we provide information about the EP algorithm used to compute the approximation to the PPESMOC acquisition function, the computation of the PPESMOC gradients to introduce them into the L-BFGS optimization algorithm in order to getter better results from the optimization of the acquisition function and results of additional experiments.

C.1 Optimization of the PPESMOC Acquisition Function Approximation

The PPESMOC acquisition function $\alpha(\cdot)$ has B times more dimensions than the sequential PESMOC acquisition function, where B is the number of points considered in the batch. Due to the curse of dimensionality, obtaining an estimate of the maximum of the acquisition function is not longer feasible by the procedure done in the PESMOC acquisition function, based only on using a grid and a local search procedure, like the L-BFGS algorithm, approximating the gradients by differences. Hence, it is necessary to compute the exact gradients of the PPESMOC acquisition function w.r.t the inputs, $\mathbf{X}$.

Importantly, as explained in the main document, the parameters of the approximate factors can be considered to be fixed after EP has converged. This simplifies significantly the gradient computation. We compute the gradients of the PPESMOC acquisition function by using the Autograd tool. To optimize PPESMOC, we choose an initial point of PPESMOC at random. Then, we run the L-BFGS algorithm with the initial point and the gradients of PPESMOC computed by Autograd. We only compute the factors of the observations and Pareto set point in the first iteration of the optimization as they are the same for all the optimization process. For every new point in the optimization done by L-BFGS we only refine once with EP the test factors. We set a maximum of 100 iterations of the L-BFGS optimization procedure.

C.2 Expectation Propagation Factors Computation

Recall from the main manuscript that, in this work, we wish to approximate the Conditional Predictive Distribution of the set defined by the points $\mathcal{X} = \{\{\boldsymbol{x}_n\}_{n=1}^N \cup \mathcal{X}^* \cup \boldsymbol{X}\}$. This set is, the union between the N observation points in the input space $\{\boldsymbol{x}_n\}_{n=1}^N$, the Pareto set points $\mathcal{X}^\star$ of M points and the B candidate points $\mathbf{X}$ to be evaluated. The Gaussian Approximation will then be a multivariate Gaussian Distribution over

$N + M + B$ variables. The Conditional Predictive Distribution is given by the following expression:

$$p(\mathbf{Y}|\mathcal{D}, \mathbf{X}, \mathcal{X}^\star) = \int p(\mathbf{Y}|\mathbf{X}, \mathbf{F}, \mathbf{C}) p(\mathcal{X}^\star|\mathbf{F}, \mathbf{C}) p(\mathbf{F}|\mathcal{D}) p(\mathbf{C}|\mathcal{D}) d\mathbf{F} d\mathbf{C}\,, \tag{C.1}$$

where

$$p(\mathcal{X}^\star|\mathbf{F}, \mathbf{C}) \propto \prod_{\mathbf{x}^\star \in \mathcal{X}^\star} \left(\left[\prod_{j=1}^{J} \Phi_j(\mathbf{x}^\star) \right] \left[\prod_{\mathbf{x}' \in \mathcal{X}} \Omega(\mathbf{x}', \mathbf{x}^\star) \right] \right), \tag{C.2}$$

and:

$$\Omega(\mathbf{x}', \mathbf{x}^\star) = \left[\prod_{j=1}^{J} \Theta(c_j(\mathbf{x}')) \right] \Psi(\mathbf{x}', \mathbf{x}^\star) + \left[1 - \prod_{j=1}^{J} \Theta(c_j(\mathbf{x}')) \right] \cdot 1\,, \tag{C.3}$$

$$\psi(\mathbf{x}', \mathbf{x}^\star) = 1 - \prod_{k=1}^{K} \Theta(f_k(\mathbf{x}^\star) - f_k(\mathbf{x}'))\,, \tag{C.4}$$

$$\Phi_j(\mathbf{x}^\star) = \Theta(c_j(\mathbf{x}^\star))\,, \tag{C.5}$$

All the non-Gaussian factors $(\Theta(c_j(\mathbf{x}^\star)), \Theta(f_k(\mathbf{x}^\star) - f_k(\mathbf{x}')), \Theta(c_j(\mathbf{x}')), \Omega(\mathbf{x}_q, \mathbf{x}^\star))$ are approximated using the expectation propagation algorithm. In this section, we provide the necessary computations to approximate these factors. In the noiseless case the Dirac delta functions are substituted by Gaussians.

In PESMOC the factors that depend and not depend on the candidate point $\mathbf{x}$ and those that depend are treated differently. For the former ones, the EP algorithm is only executed for one iteration. We employ the same procedure for the PPESMOC acquisition function.

We define the sampled Pareto set as $\mathcal{X}^\star = \{\boldsymbol{x}_1^\star, ..., \boldsymbol{x}_M^\star\}$ of size M and the set of N observations in the input space as $\hat{\mathcal{X}} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ with the corresponding batch of observations of the k-th objective $\boldsymbol{Y}_k$ and of the c-th constraint $\boldsymbol{Y}_j$. Then, the process values for each objectives and constraints at that points observed are defined by $\boldsymbol{F}_k = (f_k(\boldsymbol{x}_1^\star), ..., f_k(\boldsymbol{x}_M^\star), f_k(\boldsymbol{x}_1), \ldots, f_k(\boldsymbol{x}_N))^T$ and $\boldsymbol{C}_j = (c_j(\boldsymbol{x}_1^\star), ..., c_j(\boldsymbol{x}_M^\star), c_j(\boldsymbol{x}_1), \ldots, c_j(\boldsymbol{x}_N))^T$. If we define $\boldsymbol{F} = \{\boldsymbol{F}_1, ..., \boldsymbol{F}_K\}$ and $\boldsymbol{C} = \{\boldsymbol{C}_1, ..., \boldsymbol{C}_J\}$, let $q(\boldsymbol{F}, \boldsymbol{C})$ be the distribution that we want tt o approximate, $p(\mathcal{X}^\star|\mathbf{F}, \mathbf{C}) p(\mathbf{F}|\mathcal{D}) p(\mathbf{C}|\mathcal{D})$ is:

$$p(\boldsymbol{F}, \boldsymbol{C}|\mathcal{X}^\star) \propto \prod_{\mathbf{x}^\star \in \mathcal{X}^\star} \left(\left[\prod_{j=1}^{J} \Phi_j(\mathbf{x}^\star) \right] \left[\prod_{\mathbf{x}' \in \mathcal{X}} \Omega(\mathbf{x}', \mathbf{x}^\star) \right] \right) p(\mathbf{F}|\mathcal{D}) p(\mathbf{C}|\mathcal{D}) d\mathbf{F} d\mathbf{C}\,. \tag{C.6}$$

This last expression becomes equivalent to the next one using the fact that the posterior distributions of the GPs factorize as a product of Gaussians:

$$p(\boldsymbol{F}, \boldsymbol{c}) = \frac{1}{Z_q}\left[\prod_{k=1}^{K}\mathcal{N}\left(\boldsymbol{f}_k \mid \boldsymbol{m}_{pred}^{\boldsymbol{f}_k}, \boldsymbol{V}_{pred}^{\boldsymbol{f}_k}\right)\right]\left[\prod_{j=1}^{J}\mathcal{N}\left(\boldsymbol{c}_j \mid \boldsymbol{m}_{pred}^{\boldsymbol{c}_j}, \boldsymbol{V}_{pred}^{\boldsymbol{c}_j}\right)\right]\times$$
$$\prod_{\mathbf{x}^\star\in\mathcal{X}^\star}\left(\left[\prod_{j=1}^{J}\Phi_j(\mathbf{x}^\star)\right]\left[\prod_{\mathbf{x}'\in\mathcal{X}}\Omega(\mathbf{x}', \mathbf{x}^\star)\right]\right), \tag{C.7}$$

where $\boldsymbol{m}_{pred}^{\boldsymbol{f}_k}$ and $\boldsymbol{V}_{pred}^{\boldsymbol{f}_k}$ are the mean and covariance matrix of the posterior distributions of $\boldsymbol{f}_k$ given the data in $\mathcal{D}$ and $\boldsymbol{m}_{pred}^{\boldsymbol{c}_j}$ and $\boldsymbol{V}_{pred}^{\boldsymbol{c}_j}$ are the mean and covariance matrix of the posterior distribution of $\boldsymbol{c}_j$ given the data in $\mathcal{D}$. These means and variances are computed according to the equations 2.22-2.24 in (Rasmussen, 2003):

$$\begin{aligned}\boldsymbol{m}_{pred}^{\boldsymbol{f}_k} &= \boldsymbol{K}_*^k\left(\boldsymbol{K}^f + v_f^2\mathbb{I}\right)^{-1}\boldsymbol{y}^k,\\ \boldsymbol{V}_{pred}^{\boldsymbol{f}_k} &= \boldsymbol{K}_{*,*}^k - \boldsymbol{K}_*^k\left(\boldsymbol{K}^k + v_f^2\mathbb{I}\right)^{-1}\left[\boldsymbol{K}_*^k\right],\end{aligned} \tag{C.8}$$

where $\boldsymbol{K}_*^k$ is an $(N+1)\times N$ matrix with the prior cross-covariances between elements of $\boldsymbol{f}_k$ and $f_{k,1}, ..., f_{k,n}$ and $\boldsymbol{K}_{*,*}^k$ is an $(N+1)\times(N+1)$ matrix with the prior covariances between the elements of $\boldsymbol{f}_k$ and v_k is the standard deviation of the additive Gaussian noise in the evaluations of $\boldsymbol{f}_k$. Following the same reasoning, we have that:

$$\begin{aligned}\boldsymbol{m}_{pred}^{\boldsymbol{c}_j} &= \boldsymbol{K}_*^j\left(\boldsymbol{K}^j + v_j^2\mathbb{I}\right)^{-1}\boldsymbol{y}^j,\\ \boldsymbol{V}_{pred}^{\boldsymbol{c}_j} &= \boldsymbol{K}_{*,*}^j - \boldsymbol{K}_*^j\left(\boldsymbol{K}^j + v_j^2\mathbb{I}\right)^{-1}\left[\boldsymbol{K}_*^j\right],\end{aligned} \tag{C.9}$$

where $\boldsymbol{K}_*^j$ is an $(N+M)\times N$ matrix with the prior cross-covariances between elements of $\boldsymbol{c}_j$ and $c_{j,1}, ..., c_{j,n}$ and $\boldsymbol{K}_{*,*}^j$ is an $(N+M)\times(N+M)$ matrix with the prior covariances between the elements of $\boldsymbol{c}_j$ and v_j is the standard deviation of the additive Gaussian noise in the evaluations of $\boldsymbol{c}_j$.

The other non-Gaussian factors presented in Eq.(11), $\Phi_j(\mathbf{x}^\star)$ and $\Omega(\mathbf{x}', \mathbf{x}^\star)$, are the problematic ones, as they are not Gaussian Distributions. Hence they will be approximated by Gaussians with EP, as will be described in the next sections.

C.3 Using Expectation Propagation to Approximate the Conditional Predictive Distribution

This section explains how the EP algorithm approximate the previous product of factors, giving a product of Gaussian Distributions which we call the Gaussian Approximation to the Conditional Predictive Distribution, shown in the previous section. As it is a product where different factors are involved, we have to divide the problem in the approximation of the different factors for Gaussian Distributions. These are the $\Phi_j(\boldsymbol{x}^*$ factors and the $\Omega(\mathbf{x}', \mathbf{x}^\star)$ factors, which will be approximated by one-dimensional and two-dimensional Gaussian Distributions respectively.

The factors $\Phi(\boldsymbol{x}^*)$ that represent if a Pareto Set point $\boldsymbol{x}^*$ is feasible evaluated in a certain constraint $c_j(\boldsymbol{x}^*)$, are approximated by a one-dimensional un-normalized Gaussian distribution $\tilde{\Phi}(\boldsymbol{x}^*)$. This distribution is expressed in exponential family form in the next

equation:

$$\Phi(\boldsymbol{x}^*) \approx \tilde{\Phi}(\boldsymbol{x}^*) \propto \exp\left\{ -\frac{c_j(\boldsymbol{x}^*)^2 \hat{v}_j^{\boldsymbol{x}^*}}{2} + c_j(\boldsymbol{x}^*)\hat{m}_j^{\boldsymbol{x}^*} \right\}, \tag{C.10}$$

where $\hat{v}_j^{\boldsymbol{x}^*}$ and $\hat{m}_j^{\boldsymbol{x}^*}$ are natural parameters that are going to be adjusted by EP. The variance of the Gaussian Distribution, $\hat{v}_j^{\boldsymbol{x}^*}$, EP factor in every point, $\boldsymbol{x}^*$, for every constraint, c_j will be denoted by $\hat{e}_j$ and the mean EP factor by $\hat{f}_j$. That is, the one-dimensional Gaussian Distribution approximation of $\Phi(\boldsymbol{x}^*)$, in every constraint c_j computed by EP, $\tilde{\Phi}(\boldsymbol{x}^*)$ is defined in every point $\boldsymbol{x}^*$ belonging to $\mathcal{X}^*$, by its mean $\hat{f}_j$ and its variance $\hat{e}_j$. There will be as many Gaussian Distributions as points multiplied by constants.

The factors $\Omega(\cdot,\cdot)$, that represent if a point $\boldsymbol{x}_j$ is not dominated by the other point $\boldsymbol{x}_i$ and it is feasible over all the constraints $\boldsymbol{c}(\boldsymbol{x}_j)$, are approximated by a product of $\mathcal{J}$ one-dimensional un-normalized Gaussian Distributions where $\mathcal{J}$ are the number of constraints and $\mathcal{K}$ two-dimensional un-normalized Gaussian Distributions where $\mathcal{K}$ are the number of objectives. This product of distributions is expressed by the following equation:

$$\Omega(\boldsymbol{x}', \boldsymbol{x}^*) \approx \tilde{\Omega}(\boldsymbol{x}', \boldsymbol{x}^*) \propto \prod_{k=1}^{K} \exp\left\{ -\frac{1}{2}\boldsymbol{v}_k^T \tilde{\boldsymbol{V}}_k^{\Omega} \boldsymbol{v}_k + (\tilde{\boldsymbol{m}}_k^{\Omega})^T \boldsymbol{v}_k \right\} \times$$
$$\prod_{j=1}^{J} \exp\left\{ -\frac{c_j(\boldsymbol{x}^*)^2 \tilde{v}_j^{\Omega}}{2} + c_j(\boldsymbol{x}^*)\tilde{m}_j^{\Omega} \right\}, \tag{C.11}$$

where $\boldsymbol{v}_k$ is defined as the vector $(f_k(\boldsymbol{x}'), f_k(\boldsymbol{x}^*))^T$, and $\tilde{\boldsymbol{V}}_k$, $\tilde{\boldsymbol{m}}_k$, $\tilde{v}_j^{\Omega}$ and $\tilde{m}_j^{\Omega}$ are natural parameters adjusted by EP. As the product represents a product of two-dimensional un-normalized Gaussian Distributions, $\tilde{\boldsymbol{V}}_k$ is a 2×2 matrix and $\tilde{\boldsymbol{m}}_k$ is a two-dimensional vector.

For the set of $\mathcal{N}$ observation points in the input space $\hat{\mathcal{X}}$ and the set of $\mathcal{M}$ Pareto Set points $\mathcal{X}^*$, we define the variance of the two-dimensional Gaussian Distribution, $\tilde{\boldsymbol{V}}_k$, EP factor of an observation point $\boldsymbol{x}_i$ with respect to a Pareto Point $\boldsymbol{x}_j$ as $\hat{\boldsymbol{A}}_{ij}$ and the mean EP factor as $\hat{\boldsymbol{b}_{ij}}$. We denote the variance of the one-dimensional Gaussian Distribution, $\tilde{v}_j^{\Omega}$, EP factor in every point $\boldsymbol{x}_j$ as $\hat{ac}_j$ and the mean EP factor by $\hat{bc}_j$.

For the set of $\mathcal{M}$ Pareto Set points $\mathcal{X}^*$, we define the variance of the two-dimensional Gaussian Distribution, $\tilde{\boldsymbol{V}}_k$, EP factor of a point $\boldsymbol{x}_i$ with respect to another Pareto Point $\boldsymbol{x}_j$ as $\hat{\boldsymbol{C}}_{ij}$ and the mean EP factor as $\hat{\boldsymbol{d}}_{ij}$. We denote the variance of the one-dimensional Gaussian Distribution $\tilde{v}_j^{\Omega}$ EP factor in every point $\boldsymbol{x}_j$ as $\hat{cc}_j$ and the mean EP factor by $\hat{dc}_j$.

That is, the approximation $\tilde{\Omega}(\boldsymbol{x}', \boldsymbol{x}^*)$ computed by EP consisting of a product of one-dimensional Gaussian Distributions and two-dimensional Gaussian distributions of the distribution $\Omega(\boldsymbol{x}', \boldsymbol{x}^*)$, is defined in the set of points $\hat{\mathcal{X}}$ and $\mathcal{X}^*$ by a product of one-dimensional Gaussian Distributions with mean $\hat{bc}_j$ and variance $\hat{ac}_j$ and a product of two-dimensional Gaussian Distributions with variance $\hat{\boldsymbol{A}}_{ij}$ and mean $\hat{\boldsymbol{b}}_{ij}$. The approximation for the set of points $\mathcal{X}^*$ is defined by a product of one-dimensional Gaussian Distributions with mean $\hat{dc}_j$ and variance $\hat{cc}_j$ and a product of two-dimensional Gaussian Distributions with variance $\hat{\boldsymbol{C}}_{ij}$ and mean $\hat{\boldsymbol{d}}_{ij}$.

In the next section, the computations of the Gaussian factor approximations $\tilde{\Phi}(\cdot)$ and $\tilde{\Omega}(\cdot,\cdot)$ defined by the EP factors $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$, $\hat{e}_j$, $\hat{f}_j$, $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$, required by EP, are explained in detail, following the algorithm described in Chapter 3.

C.4 The EP Approximation to the $\Phi(\cdot)$ and $\Omega(\cdot,\cdot)$ Factors

The EP algorithm updates each of the approximate factors presented in the previous section until convergence. The following sections will describe the necessary operations needed for the EP algorithm to update each of the factors. It the following subsection, it is assumed that we have already obtained the mean and variances of each of the $\mathcal{K}$ and $\mathcal{J}$ conditional predictive distributions, which will be explained in detail in section 4.3.

C.4.1 EP Update Operations for the $\Phi(\cdot)$ Factors

As it was explained in section 3, for the $\mathcal{M}$ Pareto Set points defined by the set $\mathcal{X}^*$, in every point $\boldsymbol{x}_i \in \mathcal{X}^*$, the EP algorithm will generate J approximations for the $\Phi(\boldsymbol{x}_j)$ factors for every constraint c_j that will be defined by its mean $\hat{f}_j^{\boldsymbol{x}}$ and its variance $\hat{e}_j^{\boldsymbol{x}}$. Computations are done for all the points $\boldsymbol{x}_i \in \mathcal{X}^*$. The operations for these factors are described as follows.

C.4.1.1 Computation of the Cavity Distribution

The first step performed by the EP algorithm is the computation of the Cavity Distribution $\tilde{q}^{\backslash n}(\boldsymbol{x})$. In order to make the computations easier, we first obtain the natural parameters of the Gaussian Distributions for all the $\mathcal{M}$ Pareto Set points by using the equations:

$$\begin{aligned} \hat{\boldsymbol{m}}_j &= \frac{\boldsymbol{\xi}_j}{diag(\boldsymbol{\Xi}_j)}\,, \\ \hat{\boldsymbol{v}}_j &= \frac{1}{diag(\boldsymbol{\Xi}_j)}\,. \end{aligned} \tag{C.12}$$

Where $\boldsymbol{\Xi}_j$ is a vector of the variances of the $\mathcal{M}$ points for the constraint c_j and $\boldsymbol{\Xi}$ is the matrix of all the variances of all $\mathcal{M}$ and $\mathcal{N}$ points which construction will be explained in detail in section 4.3. The term *diag* holds for the diagonal of $\boldsymbol{\Xi}$ as we are only interested in the variance of the M points and not the variance of these points with the N points for the factor $\Phi(\cdot)$. In the same way, $\boldsymbol{\xi}_j$, is the vector of means for the constraint c_j and $\boldsymbol{\xi}$ contains all the means of all the points for every constraint in $\boldsymbol{c}$. $\hat{\boldsymbol{m}}_j$ and $\hat{\boldsymbol{v}}_j$ hold the mean and variance natural parameters corresponding for all the points in the set $\mathcal{X}^*$.

Once we have obtain the natural parameters $\hat{\boldsymbol{m}}_j$ and $\hat{\boldsymbol{v}}_j$, we obtain the cavity distribution. As we are dealing with natural parameters, it is not necessary to use the formula for the ratio of Gaussian Distributions, the cavity distribution defined by mean $\hat{\boldsymbol{m}}^{\backslash j}$ and variance $\hat{\boldsymbol{v}}^{\backslash j}$ will simply be obtained by the subtraction of the natural parameters between the approximated distribution defined by parameters $\hat{\boldsymbol{m}}_{\boldsymbol{j}}$ and $\hat{\boldsymbol{v}}_{\boldsymbol{j}}$ (which is equivalent to the product of all the factors for all the constraints) and the factor $\hat{\boldsymbol{e}}_j$ and $\hat{\boldsymbol{f}}_j$ corresponding to the constraint c_j that we want to update:

$$\begin{aligned} \hat{\boldsymbol{v}}_{nat}^{\backslash j} &= \hat{\boldsymbol{v}}_{\boldsymbol{j}} - \hat{\boldsymbol{e}}_j\,, \\ \hat{\boldsymbol{m}}_{nat}^{\backslash j} &= \hat{\boldsymbol{m}}_{\boldsymbol{j}} - \hat{\boldsymbol{f}}_j\,. \end{aligned} \tag{C.13}$$

Once the subtraction is done, we transform the natural parameters of the cavity distribution into Gaussian parameters again by using the formula that converts natural to Gaussian parameters.

$$\begin{aligned}\hat{\boldsymbol{v}}^{\setminus j} &= \frac{1}{\hat{\boldsymbol{v}}_{nat}^{\setminus j}}\,,\\ \hat{\boldsymbol{m}}^{\setminus j} &= \hat{\boldsymbol{m}}_{nat}\hat{\boldsymbol{v}}^{\setminus j}\,.\end{aligned} \tag{C.14}$$

The variances $\hat{\boldsymbol{v}}^{\setminus j}$ need to be positive for the following operations.

C.4.1.2 Computation of the Partial Derivatives of the Normalization Constant

Once the cavities $\hat{\boldsymbol{v}}^{\setminus j}$ and $\hat{\boldsymbol{m}}^{\setminus j}$ have been computed, the EP need to compute the quantities required for the update of the factors $\hat{\boldsymbol{e}}_j$ and $\hat{\boldsymbol{f}}_j$ in order to minimize the KL divergence between $\Phi(\cdot)$ and the approximation distribution. These quantities are the first and second moments of the distribution that we want to approximate. These are given by the log of the partial derivatives of Z_j, the constant that normalizes the distribution that we want to approximate, in this case, $\hat{\Phi}(\cdot)$.

$$Z_j = \int \hat{\Phi}(\boldsymbol{x}^*)\; dc_j\,. \tag{C.15}$$

As $\Phi(\boldsymbol{x}^*)$ is approximated by a Gaussian Distribution $\hat{\Phi}(\boldsymbol{x}^*)$ with mean $\hat{\boldsymbol{m}}^{\setminus j}$ and variance $\hat{\boldsymbol{v}}^{\setminus j}$, the normalization constant Z_j can be computed in closed form and its given by the cumulative distribution function ,$\Phi(\cdot)$, of this Gaussian Distribution:

$$Z_j = \Phi\left(\frac{\hat{\boldsymbol{m}}^{\setminus j}}{\sqrt{\hat{\boldsymbol{v}}^{\setminus j}}}\right). \tag{C.16}$$

Let $\alpha = \frac{\hat{\boldsymbol{m}}^{\setminus j}}{\sqrt{\hat{\boldsymbol{v}}^{\setminus j}}}$, then $\log(Z_j) = \log(\Phi(\alpha))$. For numerical robustness, if $a, b \in \mathbb{R}$, we apply the rule $\frac{a}{b} = \exp\left(\log(a) - \log(b)\right)$. Using these expressions, the log-derivatives are computed as follows:

$$\begin{aligned}\frac{\partial \log(Z_j)}{\partial \hat{\boldsymbol{m}}^{\setminus j}} &= \frac{\exp\{\log(N(\alpha)) - \log(Z_j)\}}{\sqrt{\hat{\boldsymbol{v}}^{\setminus j}}}\,,\\ \frac{\partial \log(Z_j)}{\partial \hat{\boldsymbol{v}}^{\setminus j}} &= -\frac{\exp\{\log(N(\alpha)) - \log(Z_j)\}\alpha}{2\hat{\boldsymbol{v}}^{\setminus j}}\,.\end{aligned} \tag{C.17}$$

Where $N(\cdot)$ represent the Gaussian probability density function. These expressions are valid for computing the first and second moments, but they do not present numerical robustness in all experiments. Since the lack of robustness of $\frac{\partial \log(Z_j)}{\partial \hat{\boldsymbol{v}}^{\setminus j}}$, we use the formula given by the Appendix A of the work by Opper Opper and Archambeau (2009), and use the second partial derivative $\frac{\partial^2 \log(Z_j)}{\partial [\hat{\boldsymbol{m}}^{\setminus j}]^2}$ rather than $\frac{\partial \log(Z_j)}{\partial \hat{\boldsymbol{v}}^{\setminus j}}$. This derivative is given by the following expression:

$$\frac{\partial^2 \log(Z_j)}{\partial [\hat{\boldsymbol{m}}^{\setminus j}]^2} = -\exp\{\log(N(\alpha)) - \log(Z_j)\}\frac{\alpha \exp\{\log(N(\alpha)) - \log(Z_j)\}}{\hat{\boldsymbol{v}}^{\setminus j}}\,. \tag{C.18}$$

Given these derivatives, in the next section it will be explained how to obtain the individual approximate factors $\hat{\boldsymbol{e}}_j$ and $\hat{\boldsymbol{f}}_j$.

C.4.2 EP Update Operations for the $\Omega(\cdot,\cdot)$ Factors

Recalling section 3, for the $\mathcal{M}$ Pareto Set points defined by the set $\mathcal{X}^*$ and the $\mathcal{N}$ input space observation points defined by the set $\hat{\mathcal{X}}$, for every pair of points $\boldsymbol{x}_i \in \hat{\mathcal{X}}$ and $\boldsymbol{x}_j \in \mathcal{X}^*$, the EP will generate K two-dimensional Gaussian approximations for every objective $\boldsymbol{f}_k$ that will be defined for the pair observation and Pareto set point by factors defined by mean $\hat{\boldsymbol{b}}_{ij}$ and variance $\hat{\boldsymbol{A}}_{ij}$ and for the pair of Pareto set points by factors defined by mean $\hat{\boldsymbol{d}}_{ij}$ and variance $\hat{\boldsymbol{C}}_{ij}$. It will also define J one-dimensional Gaussian approximations for every constraint c_j that will be defined for the pair observation and Pareto set point by factors defined by mean $\hat{bc}_j$ and variance $\hat{ac}_j$ and for the pair of Pareto set points by factors defined by mean $\hat{dc}_j$ and variance $\hat{cc}_j$. Computations are done for all the pairs of points from the sets $\mathcal{X}^*$ and $\hat{\mathcal{X}}$. The necessary operations for computing these factors are described in the following sections.

C.4.2.1 Computation of the Cavity Distribution

For the factors $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$ that approximate the $\mathcal{J}$ one-dimensional Gaussian approximations for every constraint c_j, the operations needed to extract the cavity distribution from the approximate distribution are the same ones as the ones described in Section 4.1.1. These operations are done for the observation points in $\hat{\mathcal{X}}$ for the factors $\hat{ac}_j$, $\hat{bc}_j$ and for the Pareto Set points in $\mathcal{X}^*$ for the factors $\hat{cc}_j$ and $\hat{dc}_j$. That is, obtaining the natural parameters of $\boldsymbol{\Xi}_j$ as in Eq. (16), subtracting the natural parameters of the factor that is approximated, Eq. (17), and obtaining the Gaussian parameters of the cavity that we define for a point $\boldsymbol{x}_i$, $m_{ij}^{\setminus b_j}$ and $v_{ij}^{\setminus a_j}$, as shown in Eq. (18).

Obtaining the cavity distribution for the factors $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$ and $\hat{\boldsymbol{d}}_{ij}$ that approximate the $\mathcal{K}$ two-dimensional Gaussian approximations for every objective $\boldsymbol{f}_k$ follow different expressions as in this case the Gaussian Distributions are bivariate for every pair of points considered.

In the first case, for the case of approximating a distribution that consider a point $\boldsymbol{x}_i$ belonging to the observations set $\hat{\mathcal{X}}$ and a point $\boldsymbol{x}_j$ from the Pareto set $\mathcal{X}^*$, that is, the factors $\hat{\boldsymbol{A}}_{ij}$ and $\hat{\boldsymbol{b}}_{ij}$, it is necessary to obtain, for every objective k and each of the pair of points mentioned, the natural parameters $\boldsymbol{m}_{ij(nat)}^k$ and $\boldsymbol{V}_{ij}^{k^{-1}}$ of the Gaussian Process that models each of the $\mathcal{K}$ objectives $f(\cdot)_j$. These natural parameters are obtained by the following expressions:

$$\begin{aligned} \boldsymbol{m}_{ij(nat)}^k &= \boldsymbol{V}_{ij}^{k^{-1}} \boldsymbol{m}_{ij}^k \,, \\ \boldsymbol{V}_{ij}^{k^{-1}} &= (\boldsymbol{V}_{ij}^k)^{-1} \,, \end{aligned} \tag{C.19}$$

where $\boldsymbol{V}_{ij}^k$ is a 2x2 matrix that represent in the points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ the variance of the Gaussian approximation of the objective k and $\boldsymbol{m}_{ij}^k$ is a vector that represent in the points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ the mean of the Gaussian approximation of the objective k.

As in the constraints case, we now extract the cavity distribution that we define by the natural parameters $\boldsymbol{m}_{ijk(nat)}^{\setminus b}$ and $\boldsymbol{V}_{ijk(nat)}^{\setminus A}$, by subtracting to the computed natural parameters $\boldsymbol{m}_{ij(nat)}^k$ and $\boldsymbol{V}_{ij}^{k^{-1}}$, computed in the previous step, the factors that we want

to update $\boldsymbol{b}_{ij}^{k}$ and $\boldsymbol{A}_{ij}^{k}$. That is:

$$\begin{aligned}\boldsymbol{m}_{ijk(nat)}^{\backslash b} &= \boldsymbol{m}_{ij(nat)}^{k} - \boldsymbol{b}_{ij}^{k}\,,\\ \boldsymbol{V}_{ijk(nat)}^{\backslash A} &= \boldsymbol{V}_{ij}^{k\,-1} - \boldsymbol{A}_{ij}^{k}\,.\end{aligned} \tag{C.20}$$

For the bivariate Gaussian distribution, the step of obtaining the Gaussian parameters from the natural parameters is defined by the following expressions:

$$\begin{aligned}\boldsymbol{m}_{ijk}^{\backslash b} &= \boldsymbol{V}_{ijk}^{\backslash A}\ \boldsymbol{m}_{ijk(nat)}\,,\\ \boldsymbol{V}_{ijk}^{\backslash A} &= (\boldsymbol{V}_{ijk(nat)}^{\backslash A})^{-1}\,,\end{aligned} \tag{C.21}$$

where $\boldsymbol{V}_{ijk}^{\backslash A}$ is a 2x2 matrix with the variances of each of the points and the correlation between each of them and $\boldsymbol{m}_{ijk}^{\backslash b}$ is a two position vector that represent the means. In the case of the factors $\hat{\boldsymbol{C}}_{ij}$ and $\hat{\boldsymbol{d}}_{ij}$ that consider two Pareto Set points, the operations for extracting the cavity distribution are the same ones as in the previous case.

C.4.2.2 Computation of the Partial Derivatives of the Normalization Constant

In this section, the operations needed to compute the partial derivatives for all the $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$, $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$ are described. These derivatives need previous computations in order to compute the normalization constant Z_Ω of the factor $\Omega(\cdot,\cdot)$ that we want to approximate. These computations are given by the following expressions, all of which depend upon terms computed in the previous section. The shown computations are the result of applying rules in order to be robust such as $a/b = \exp\{\log(a) - \log(b)\}$ and $ab = \exp\{\log(a) + \log(b)\}$.These operations are equivalent for the two points cases, but here, the necessary operations for computing the normalization constant Z_Ω are described for the case of the factors $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{ac}_j$ and $\hat{bc}_j$:

$$\boldsymbol{s}_k = \boldsymbol{V}_{ijk[0,0]}^{\backslash A} + \boldsymbol{V}_{ijk[1,1]}^{\backslash A} - 2\boldsymbol{V}_{ijk[0,1]}^{\backslash A}\,, \tag{C.22}$$

$$\boldsymbol{\alpha}_k = \frac{\boldsymbol{m}_{ijk[0]}^{\backslash b} - \boldsymbol{m}_{ijk[1]}^{\backslash b}}{\sqrt{s_k}}\,, \tag{C.23}$$

$$\boldsymbol{\beta}_j = \frac{m_{ij}^{\backslash b_j}}{\sqrt{v_{ij}^{\backslash a_j}}}\,, \tag{C.24}$$

$$\boldsymbol{\phi} = \Phi(\boldsymbol{\alpha})\,, \tag{C.25}$$

$$\tag{C.26}$$

where $\Phi(\cdot)$ represents the c.d.f of a Gaussian distribution,

$$\boldsymbol{\gamma} = \Phi(\boldsymbol{\beta})\,, \tag{C.27}$$

$$\zeta = 1 - \exp\{\sum_{k=1}^{K} \log(\boldsymbol{\phi}_k)\}\,, \tag{C.28}$$

$$\log(\eta) = \sum_{j=1}^{J} \log(\boldsymbol{\gamma}_j) + \log(\zeta)\,, \tag{C.29}$$

$$\lambda = 1 - \exp\{\sum_{j=1}^{J} \log(\boldsymbol{\gamma}_j)\}\,, \tag{C.30}$$

$$\tau = \max(\log(\eta), \log(\lambda))\,, \tag{C.31}$$

$$\log(Z_\Omega) = \log(\exp\{\log(\eta) - \tau\} + \exp\{\log(\lambda) - \tau\}) + \tau\,. \tag{C.32}$$

Having computed these terms, the log partial derivatives for the update of the factors that collaborate to the approximation of the objective variances $\hat{\boldsymbol{A}}_{ij}$ and the objective means $\hat{\boldsymbol{b}}_{ij}$ are given by the expressions:

$$\boldsymbol{\rho}_k = -\exp\{\log(\mathcal{N}(\boldsymbol{\alpha}_k))\} - \log(Z_\Omega) + \sum_{k=1}^{K}\{\log(\Phi(\boldsymbol{\alpha}_k))\} - \log(\Phi(\boldsymbol{\alpha}_k)) + \sum_{j=1}^{J}\{\log(\Phi(\boldsymbol{\beta}_j))\}\,, \tag{C.33}$$

$$\begin{aligned}
\frac{\partial \log(Z_\Omega)}{\partial \boldsymbol{m}_{ijk}^{\backslash b}} &= \frac{\boldsymbol{\rho}_k}{\sqrt{\boldsymbol{s}_k}}\big[1, -1\big]\,, \\
\frac{\partial \log(Z_\Omega)}{\partial \boldsymbol{V}_{ijk}^{\backslash A}} &= -\frac{\boldsymbol{\rho}_k \boldsymbol{\alpha}_k}{2\boldsymbol{s}_k}[[1, -1], [-1,\ 1]]\,.
\end{aligned} \tag{C.34}$$

Derivatives are computed for the two position vector mean and the 2x2 variance matrix, so they have the same structure, given by the [1, -1] and [[1, -1],[-1, 1]] expressions. The change in the sign appears due to the fact that the expression changes, whether it is the derivative of the mean of the observation point or the Pareto Set point or the derivative of the variance of one point or their correlation.

Alas, the derivative of the variance presents the same lack of robustness as in the constraint case shown in section 4.1.2. In order to ensure numerical robustness, we use the second partial derivative of the mean of the normalization constant instead of the first partial derivative of the variance for the further computation of the second moment. That is,

$$\frac{\partial^2 \log(Z_\Omega)}{\partial \big[\boldsymbol{m}_{ijk}^{\backslash b}\big]^2} = -\frac{\boldsymbol{\rho}_k}{\boldsymbol{s}_k}(\boldsymbol{\alpha}_k + \boldsymbol{\rho}_k)[[1, -1], [-1,\ 1]]\,. \tag{C.35}$$

For the log partial derivatives for the update of the factors that collaborate to the approximation of the constraint variances $\hat{ac}_j$ and the constraint means $\hat{bc}_j$, let $\boldsymbol{\omega}_j$ be

defined as:

$$\boldsymbol{\omega}_j = \exp\{\log(\mathcal{N}(\boldsymbol{\beta}_j))\} - \log(Z_\Omega) + \log(\zeta) + \sum_{j=1}^{J}(\log(\Phi(\boldsymbol{\beta}_j))) - \log(\Phi(\boldsymbol{\beta}_j)) - \exp\{\log(\mathcal{N}(\boldsymbol{\beta}_j))\}\,,$$

$$- \log(Z_\Omega) + \sum_{j=1}^{J}(\log(\Phi(\boldsymbol{\beta}_j))) - \log(\Phi(\boldsymbol{\beta}_j))\,. \tag{C.36}$$

Then, the robust log partial derivatives for the first and the second moments are given by the expressions:

$$\begin{aligned}\frac{\partial \log(Z_\Omega)}{\partial m_{ij}^{\backslash \boldsymbol{b}_j}} &= \frac{\boldsymbol{\omega}_j}{\sqrt{\boldsymbol{s}_j}}\,,\\ \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ij}^{\backslash \boldsymbol{b}_j}]^2} &= -\frac{\boldsymbol{\omega}_j}{\boldsymbol{s}_j}\,(\boldsymbol{\beta}_j + \boldsymbol{\omega}_j)\,.\end{aligned} \tag{C.37}$$

The expressions for the log partial derivatives of $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$, $\hat{cc}_j$ and $\hat{dc}_j$ are similar to the presented expressions in this section, but taking into account pairs of points belonging to the set $\mathcal{X}^*$.

C.4.2.3 Computation of the First and Second Moments for the Updates

Giving the expressions computed in the previous section, the first and second moments of the different Gaussian Distributions that approximate the factor $\Omega(\cdot,\cdot)$ can now be computed.

The expressions for computing the factors $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$ for each of the K objectives and the factors $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$ for each of the J constraints are the following ones:

$$\hat{\boldsymbol{A}}_{ij}^k = \frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\backslash b}\right]^2}\,((\boldsymbol{V}_{ijk}^{\backslash A}\frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\backslash b}\right]^2})^{-1}[[1,0],[0,1]])\,, \tag{C.38}$$

$$\hat{\boldsymbol{b}}_{ij}^k = ((\frac{\partial \log(Z_\Omega)}{\partial \boldsymbol{m}_{ijk}^{\backslash b}} - \boldsymbol{m}_{ijk}^{\backslash b})\frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\backslash b}\right]^2})\,((\boldsymbol{V}_{ijk}^{\backslash A}\frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\backslash b}\right]^2})^{-1} + [[1,0],[0,1]])\,, \tag{C.39}$$

$$\hat{\boldsymbol{C}}_{ij}^k = \frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\backslash b}\right]^2}\,((\boldsymbol{V}_{ijk}^{\backslash A}\frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\backslash b}\right]^2})^{-1}[[1,0],[0,1]])\,, \tag{C.40}$$

$$\hat{\boldsymbol{d}}_{ij}^k = ((\frac{\partial \log(Z_\Omega)}{\partial \boldsymbol{m}_{ijk}^{\backslash b}} - \boldsymbol{m}_{ijk}^{\backslash b})\frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\backslash b}\right]^2})\,((\boldsymbol{V}_{ijk}^{\backslash A}\frac{\partial^2 \log(Z_\Omega)}{\partial \left[\boldsymbol{m}_{ijk}^{\backslash b}\right]^2})^{-1} + [[1,0],[0,1]])\,, \tag{C.41}$$

for the the rest of the factors, suppose that the index h refers to the points of the Pareto Set $\mathcal{X}^*$:

$$\hat{ac}_h^j = -\frac{\frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2}}{1 + \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2} v_{ic}^{\backslash a_j}}, \tag{C.42}$$

$$\hat{bc}_h^j = \frac{\frac{\partial \log(Z_\Omega)}{\partial m_{ic}^{\backslash \boldsymbol{b}_j}} - m_{ic}^{\backslash b_j} \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2}}{1 + \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2} v_{ic}^{\backslash a_j}}, \tag{C.43}$$

$$\hat{cc}_h^j = -\frac{\frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2}}{1 + \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2} v_{ic}^{\backslash a_j}}, \tag{C.44}$$

$$\hat{dc}_h^j = \frac{\frac{\partial \log(Z_\Omega)}{\partial m_{ic}^{\backslash \boldsymbol{b}_j}} - m_{ic}^{\backslash b_j} \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2}}{1 + \frac{\partial^2 \log(Z_\Omega)}{\partial [m_{ic}^{\backslash \boldsymbol{b}_j}]^2} v_{ic}^{\backslash a_j}}. \tag{C.45}$$

All these factors are then used to rebuild the means and the variances of the Gaussian Processes that model the K objectives and C constraints of a constrained multi-objective optimization problem, as will be shown in the following section. That is, C one-dimensional Gaussian Distributions for the constraint models and C one-dimensional Gaussian Distributions and K two-dimensional Gaussian Distributions for the objective models in each of the points in $\mathcal{X} = \{\mathcal{X}^* \cup \hat{\mathcal{X}} \cup \boldsymbol{x}\}$.

C.4.3 Reconstruction of the Conditional Predictive Distribution

In this section, we illustrate the way of obtaining a Conditional Predictive Distribution for every objective f_k and every constraint c_j, given a sampled Pareto Set $\mathcal{X}^* = \{\boldsymbol{x}_1^*, ..., \boldsymbol{x}_M^*\}$ of size M and a set of N input locations $\hat{\mathcal{X}} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ with corresponding observations of the k-th objective $\boldsymbol{y}_k$ and of the j-th constraint $\boldsymbol{y}_j$. For the following, it is assumed that we are given the EP approximate factors $\Phi(\cdot)$ and $\Omega(\cdot, \cdot)$, as an input for the next operations, which computation is explained in the previous section.

Recalling Eqs. 7, 8 and 9 of section 2, we want to obtain the J Conditional Predictive Distributions in the products of constraints and the K Conditional Predictive Distributions of the Gaussian Processes that model the objectives. The products presented in these factors are not a problem, due to the fact that the Gaussian Distributions are closed under the product operation, that is, the product of Gaussian Distributions is another Gaussian Distribution. These Conditional Predictive Distributions of the objectives and constraints are then used in Eq.(11) to build the final approximation.

Following the notation of section 4.1.1, let $\boldsymbol{\xi}_j$ and $\boldsymbol{\Xi}_j$ be the mean vector and variance matrix of the one-dimensional Gaussian Distributions of the $M + N$ points that generate the Gaussian Processes that model the constraints and let $\boldsymbol{m}_k$ and $\boldsymbol{V}_k$ be the mean vector and variance matrix of the two-dimensional Gaussian Distributions of the $M + N$ points that generate the Gaussian Processes that model the objectives. In order to update the constraint and objective distribution marginals, it is necessary to first follow the operations given by the equations 14 and 22, to obtain the natural parameters from the means and variances. Intuitively, as they are all natural parameters, these will be just sums taking into account that the matrices are formed first by the Pareto Set Points ,M,

and then by the observations N. Univariate factors are added to the diagonal of these matrices, as they are not correlated with other points. Once the natural parameters are computed, the new means $\boldsymbol{\xi}_j$, $\boldsymbol{m}_k$ and variances $\boldsymbol{\Xi}_j$, $\boldsymbol{V}_k$ marginals are updated from the EP factors $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$, $\hat{e}_j$, $\hat{f}_j$, $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$ by the following expressions:

$$
\begin{aligned}
\boldsymbol{\Xi}_{ii}^{j} &= \boldsymbol{\Xi}_{ii(old)}^{j} + \sum_{m=1}^{M} \hat{cc}_{mi}^{j} + \hat{e}_i^j && \text{for } \; i = 1, ..., M\,, \\
\boldsymbol{\Xi}_{ii}^{c} &= \boldsymbol{\Xi}_{ii(old)}^{j} + \sum_{m=1}^{M} \hat{ac}_{mi}^{j} && \text{for } \; i = M+1, ..., N+M\,, \\
\boldsymbol{\xi}_{i}^{c} &= \boldsymbol{\xi}_{i(old)}^{j} + \sum_{m=1}^{M} \hat{dc}_{mi}^{j} + \hat{f}_i^j && \text{for } \; i = 1, ..., M\,, \\
\boldsymbol{\xi}_{i}^{c} &= \boldsymbol{\xi}_{i(old)}^{j} + \sum_{m=1}^{M} \hat{bc}_{mi}^{j} && \text{for } \; i = M+1, ..., N+M\,, \\
\boldsymbol{V}_{ii}^{k} &= \boldsymbol{V}_{ii(old)}^{k} + \sum_{j=M+1}^{N} \hat{\boldsymbol{A}}_{ji[1,1]}^{k} + \sum_{j=1}^{M} \hat{\boldsymbol{C}}_{ij[0,0]}^{k} + \sum_{j=1}^{M} \hat{\boldsymbol{C}}_{ji[1,1]}^{k} && \text{for } \; i = 1, ..., M\,, \\
\boldsymbol{V}_{ii}^{k} &= \boldsymbol{V}_{ii(old)}^{k} + \sum_{j=1}^{M} \hat{\boldsymbol{A}}_{ij[0,0]}^{k} && \text{for } \; i = M+1, ..., N+M\,, \\
\boldsymbol{V}_{ij}^{k} &= \boldsymbol{V}_{ij(old)}^{k} + \boldsymbol{C}_{ij[0,1]}^{k} + \boldsymbol{C}_{ij[1,0]}^{k\;T} && \text{for } \; i = 1, ..., M, \text{ and for } \; j = 1, ..., M\,, \\
\boldsymbol{V}_{ij}^{k} &= \boldsymbol{V}_{ij(old)}^{k} + \boldsymbol{A}_{ij[0,1]}^{k} && \text{for } \; i = M+1, ..., N, \text{ and for } \; j = 1, ..., M\,, \\
\boldsymbol{V}_{ij}^{k} &= \boldsymbol{V}_{ij(old)}^{k} + \boldsymbol{A}_{ij[0,1]}^{k\;T} && \text{for } \; i = 1, ..., M, \text{ and for } \; j = M+1, ..., N\,, \\
\boldsymbol{m}_{i}^{k} &= \boldsymbol{m}_{i(old)}^{k} + \sum_{j=M+1}^{N+M} \hat{\boldsymbol{b}}_{ji[1]}^{k} + \sum_{j=1}^{M} \hat{\boldsymbol{d}}_{ij[0]}^{k} + \sum_{j=1}^{M} \hat{\boldsymbol{d}}_{ji[1]}^{k} && \text{for } \; i = 1, ..., M\,, \\
\boldsymbol{m}_{i}^{k} &= \boldsymbol{m}_{i(old)}^{k} + \sum_{j=1}^{M} \hat{\boldsymbol{b}}_{ij[0]}^{k} && \text{for } \; i = M+1, ..., N+M\,.
\end{aligned}
\tag{C.46}
$$

These natural parameters are then converted into Gaussian ones using the equations and 16 and 24. Once these operations are done the Gaussian Processes that model the objectives and constraints are updated from a full EP iteration.

C.4.4 The Conditional Predictive Distribution at a New Batch

After running EP until convergence one simply has to compute the covariance matrix of the posterior distribution for the process values of each objective and constraint at the candidate points $\mathbf{X}$. This implies computing the covariance matrix that results from the EP approximation to (C.7). For this, one only has to replace the non-Gaussian factors with the corresponding approximation. The covariance matrices that are needed can be obtained using the fact that the Gaussian family is closed under the product operation. See Eq. (**??**).

C.4.5 Initialization and Convergence of EP

When the EP algorithm computes the $\Phi(\cdot)$ and $\Omega(\cdot,\cdot)$ factors, it requires to set an initial value to all the factors that generates the Gaussians that approximate the $\Phi(\cdot)$ and $\Omega(\cdot,\cdot)$ factors. These factors, $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$, $\hat{e}_j$, $\hat{f}_j$, $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$ are all set to be zero. The convergence criterion for stopping the EP algorithm updating the parameters is that the absolute change in all the cited parameters should be below 10^{-4}. Other criteria may be used.

C.4.6 Parallel EP Updates and Damping

The updates of every approximate factor $\hat{\boldsymbol{A}}_{ij}$, $\hat{\boldsymbol{b}}_{ij}$, $\hat{\boldsymbol{C}}_{ij}$, $\hat{\boldsymbol{d}}_{ij}$, $\hat{e}_j$, $\hat{f}_j$, $\hat{ac}_j$, $\hat{bc}_j$, $\hat{cc}_j$ and $\hat{dc}_j$ are executed in parallel as it is described in the work by Gerven Gerven et al. (2009). The cavity distribution for each of the factors is computed and then the factors are updated afterwards. Once these operations are done the EP approximation is recomputed as it is described in the section 4.3.

In order to improve the convergence behavior of EP we use the damping technique described in Minka & Lafferty Minka and Lafferty (2012). We use this technique for all the approximate factors. Damping simply reduces the quantity that the factor changes in every update as a linear combination between the old parameters and the new parameters. That is, if we define the old parameters of the factor to be updated as u_{old}, the new parameters as u_{new} and the updated factor as u, then the update expression is:

$$u = \theta u_{new} + (1-\theta)u_{old}\,. \tag{C.47}$$

Where θ is the damping factor whose initial value is set to be 0.5, this factor controls the amount of damping, if this value is set to be one then no damping is employed. This factor is multiplied by 0.99 at each iteration, reducing the amount of change in the approximate factors in every iteration of the Bayesian Optimization. An issue that happens during the optimization process is that some covariance matrices become non positive definite due to a high large step size, that is, a high value of θ. If this happens in any iteration, an inner loop executes again the update operation with $\theta_{new} = \theta_{old}$ / 2 and the iteration is repeated. This inner loop is performed until the covariance matrices become non positive definite.

C.5 Additional Experiments Information

In this section, we include additional information about the experiments described on the main manuscript and their results.

C.5.1 Benchmark Experiments

We include tables with the analytical expressions of the benchmark of functions used for the benchmark experiments.

TABLE C.1: Summary of BNH, SRN, TNK and OSY problems used in the benchmark experiments.

Benchmark Experiments		
Problem Name	Input Space	Objectives $f_k(\mathbf{x})$ and Constraints $c_j(\mathbf{x})$
BNH	$x_1 \in [0, 5]$ $x_2 \in [0, 3]$	$f_1(\mathbf{x}) = 4x_1^2 + 4x_2^2$ $f_2(\mathbf{x}) = (x_1 - 5)^2 + (x_2 - 5)^2$ $c_1(\mathbf{x}) \equiv (x_1 - 5)^2 + x_2^2 \leq 25$ $c_2(\mathbf{x}) \equiv (x_1 - 8)^2 + (x_2 + 3)^2 \geq 7.7$
SRN	$x_1 \in [-20, 20]$ $x_2 \in [-20, 20]$	$f_1(\mathbf{x}) = 2 + (x_1 - 2)^2 + (x_2 - 2)^2$ $f_2(\mathbf{x}) = 9x_1 - (x_2 - 1)^2$ $c_1(\mathbf{x}) \equiv x_1^2 + x_2^2 \leq 225$ $c_2(\mathbf{x}) \equiv x_1 - 3x_2 + 10 \leq 0$
TNK	$x_1 \in [0, \pi]$ $x_2 \in [0, \pi]$	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}) = x_2$ $c_1(\mathbf{x}) \equiv x_1^2 + x_2^2 - 1 - 0.1cos(16arctan\frac{x_1}{x_2}) \geq 0$ $c_2(\mathbf{x}) \equiv (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \leq 0.5$
OSY	$x_1 \in [0, 10]$ $x_2 \in [0, 10]$ $x_3 \in [1, 5]$ $x_4 \in [0, 6]$ $x_5 \in [1, 5]$ $x_6 \in [0, 10]$	$f_1(\mathbf{x}) = -[25(x_1 - 2)^2 + (x_2 - 2)^2 + (x_3 - 1)^2$ $+(x_4 - 4)^2 + (x_5 - 1)^2$ $f_2(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2$ $c_1(\mathbf{x}) \equiv x_1 + x_2 - 2 \geq 0$ $c_2(\mathbf{x}) \equiv 6 - x_1 - x_2 \geq 0$ $c_3(\mathbf{x}) \equiv 2 - x_2 + x_1 \geq 0$ $c_4(\mathbf{x}) \equiv 2 - x_1 + 3x_2 \geq 0$ $c_5(\mathbf{x}) \equiv 4 - (x_3 - 3)^2 - x_4 \geq 0$ $c_6(\mathbf{x}) \equiv (x_5 - 3)^2 + x_6 - 4 \geq 0$

TABLE C.2: Summary of CONSTR and Two-bar Truss problems used in the benchmark experiments.

Benchmark Experiments		
Problem Name	Input Space	Objectives $f_k(\mathbf{x})$ and Constraints $c_j(\mathbf{x})$
CONSTR	$x_1 \in [0.1, 10]$ $x_2 \in [0, 5]$	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}) = \frac{(1+x_2)}{x_1}$ $c_1(\mathbf{x}) \equiv x_2 + 9x_1 \geq 6$ $c_2(\mathbf{x}) \equiv -x_2 + 9x_1 \geq 1$
Two-bar Truss Design	$x_1 \in [0, 0.01]$ $x_2 \in [0, 0.01]$ $x_3 \in [1, 3]$	$f_1(\mathbf{x}) = x_1\sqrt{16 + x_3^2} + x_2\sqrt{1 + x_3^2}$ $f_2(\mathbf{x}) = \max(\frac{20\sqrt{16+x_3}}{x_1 x_3}, \frac{80\sqrt{1+x_3^2}}{x_2 x_3})$ $c_1(\mathbf{x}) \equiv \max(\frac{20\sqrt{16+x_3}}{x_1 x_3}, \frac{80\sqrt{1+x_3^2}}{x_2 x_3}) \leq 10^5$

C.5.2 Real Experiments

A summary of the parameters considered in the experiment of the hyper-parameter tuning of the deep neural network, their potential values, and their impact in each black-box function (prediction error, time and chip area) is displayed in Table C.3.

TABLE C.3: Parameter space of the deep neural network experiments. PE = Prediction error. T = Time. CA = Chip area.

Parameter	**Min**	**Max**	**Step**	**Black-box**
Hidden Layers	1	3	1	PE/T/CA
Neurons per Layer	5	300	1	PE/T/CA
Learning rate	e^{-20}	1	ϵ	PE
Dropout rate	0	0.9	ϵ	PE
ℓ_1 penalty	e^{-20}	1	ϵ	PE
ℓ_2 penalty	e^{-20}	1	ϵ	PE
Memory partition	1	32	2^x	CA
Loop unrolling	1	32	2^x	CA

Bibliography

Abramowitz, M. and Stegun, I. A. (1965). Handbook of mathematical functions with formulas, graphs, and mathematical table. In *US Department of Commerce*. National Bureau of Standards Applied Mathematics series 55.

Abramowitz, M., Stegun, I. A., and Romer, R. H. (1988). *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. American Association of Physics Teachers.

Agustín-Blas, L., Salcedo-Sanz, S., Vidales, P., Urueta, G., and Portilla-Figueras, J. (2011). Near optimal citywide wifi network deployment using a hybrid grouping genetic algorithm. *Expert Systems with Applications*, 38(8):9543–9556.

Agustín-Blas, L. E., Salcedo-Sanz, S., Ortiz-García, E., Portilla-Figueras, A., and Pérez-Bellido, A. (2009). A hybrid grouping genetic algorithm for assigning students to preferred laboratory groups. *Expert Systems with Applications*, 36(3):7234–7241.

Alaya, I., Solnon, C., and Ghedira, K. (2007). Ant colony optimization for multi-objective optimization problems. In *19th IEEE International Conference on Tools with Artificial Intelligence*, pages 450–457.

Albert, A. (1972). Regression and the moore-penrose pseudoinverse.

Ariizumi, R., Tesch, M., Choset, H., and Matsuno, F. (2014). Expensive multiobjective optimization for robotics with consideration of heteroscedastic noise. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2230–2235.

Arinaga, R. and Cheung, K. (2012). Atlas of global wave energy from 10 years of reanalysis and hindcast data. *Renewable Energy*, 39(1):49–64.

Artin, E. (2015). *The Gamma function*. Courier Dover Publications.

Azimi, J., Jalali, A., and Fern, X. (2012). Hybrid batch Bayesian optimization. *arXiv preprint arXiv:1202.5597*.

Bahaj, A. (2011). Generating electricity from the oceans. *Renewable and Sustainable Energy Reviews*, 15(7):3399–3416.

Baheri, A., Bin-Karim, S., Bafandeh, A., and Vermillion, C. (2017). Real-time control using Bayesian optimization: A case study in airborne wind energy systems. *Control Engineering Practice*, 69:131–140.

Balandat, M., Karrer, B., Jiang, D., Daulton, S., Letham, B., Wilson, A., and Bakshy, E. (2019). BoTorch: Programmable Bayesian Optimization in PyTorch. *arxiv e-prints*.

Bengio, Y. (2009). *Learning deep architectures for AI*. Now Publishers Inc.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554.

Bergstra, J., Yamins, D., Cox, D. D., et al. (2013). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, volume 13, page 20. Citeseer.

Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., and Jones, Z. M. (2016). mlr: Machine learning in r. *The Journal of Machine Learning Research*, 17(1):5938–5942.

Bischl, B., Richter, J., Bossek, J., Horn, D., Thomas, J., and Lang, M. (2017a). mlrmbo: A modular framework for model-based optimization of expensive black-box functions. *arXiv preprint arXiv:1703.03373*.

Bischl, B., Richter, J., Bossek, J., Horn, D., Thomas, J., and Lang, M. (2017b). mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions. *arXiv:1703.03373*.

Bishop, C. (2006). *Pattern recognition and machine learning*. springer.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR.

Borge, J., Reichert, K., and Hessner, K. (2013). Detection of spatio-temporal wave grouping properties by using temporal sequences of x-band radar images of the sea surface. *Ocean Modelling*, 61:21–37.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Brochu, E., Cora, V., and De Freitas, N. (2010). A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Brown, E. and Sumichrast, R. (2005). Evaluating performance advantages of grouping genetic algorithms. *Engineering Applications of Artificial Intelligence*, 18(1):1–12.

Bühlmann, P. and Yu, B. (2002). Analyzing bagging. *The Annals of Statistics*, 30:927–961.

Bui, T., Hernández-Lobato, D., Hernandez-Lobato, J., Li, Y., and Turner, R. (2016). Deep Gaussian processes for regression using approximate expectation propagation. In *International Conference on Machine Learning*, pages 1472–1481.

Cahill, B. and Lewis, T. (2013). Wave energy resource characterisation of the atlantic marine energy test site. *International Journal of Marine Energy*, 1:3–15.

Cai, Z. and Wang, Y. (2006). A multiobjective optimization-based evolutionary algorithm for constrained optimization. *IEEE Transactions on evolutionary computation*, (6):658–675.

Chafekar, D., Xuan, J., and Rasheed, K. (2003). Constrained multi-objective optimization using steady state genetic algorithms. In *Genetic and Evolutionary Computation Conference*, pages 813–824.

Chaslot, G. M. J.-B. C. (2010). *Monte-carlo tree search*. Maastricht University.

Chen, T., Fox, E., and Guestrin, C. (2014). Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691.

Chen, Y., Huang, A., Wang, Z., Antonoglou, I., Schrittwieser, J., Silver, D., and de Freitas, N. (2018). Bayesian optimization in alphago. *arXiv preprint arXiv:1812.06855*.

Chib, S. and Greenberg, E. (1995). Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335.

Chollet, F. (2015). Keras.

Coello, C. A., Pulido, G. T., and Lechuga, M. S. (2004). Handling multiple objectives with particle swarm optimization. *IEEE Transactions on evolutionary computation*, 8:256–279.

Comola, F., Andersen, T., Martinelli, L., Burcharth, H., and Ruol, P. (2014). Damage pattern and damage progression on breakwater roundheads under multidirectional waves. *Coastal Engineering*, 83:24–35.

Córdoba, I., Garrido-Merchán, E., Hernández-Lobato, D., Bielza, C., and Larranaga, P. (2018). Bayesian optimization of the pc algorithm for learning Gaussian Bayesian networks. In *Conference of the Spanish Association for Artificial Intelligence*, pages 44–54. Springer.

Cornejo-Bueno, L., Garrido-Merchán, E., Hernández-Lobato, D., and Salcedo-Sanz, S. (2018). Bayesian optimization of a hybrid system for robust ocean wave features prediction. *Neurocomputing*, 275:818–828.

Cornejo-Bueno, L., Nieto-Borge, J., García-Díaz, P., Rodríguez, G., and Salcedo-Sanz, S. (2016). Significant wave height and energy flux prediction for marine energy applications: A grouping genetic algorithm–extreme learning machine approach. *Renewable Energy*, 97:380–389.

Cover, T. and Thomas, J. (2012). *Elements of information theory*. John Wiley & Sons.

Cuadra, L., Salcedo-Sanz, S., Nieto-Borge, J., Alexandre, E., and Rodríguez, G. (2016). Computational intelligence in wave energy: Comprehensive review and case study. *Renewable and Sustainable Energy Reviews*, 58:1223–1246.

D., H.-L., J.M., H.-L., A., S., and R.P., A. (2016). Predictive entropy search for multi-objective Bayesian optimization. In *International Conference on Machine Learning*, pages 1492–1501.

Damianou, A. and Lawrence, N. (2013). Deep Gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215.

Davis, L. (1991). Handbook of genetic algorithms.

Daxberger, E. A. and Low, B. K. H. (2017). Distributed batch Gaussian process optimization. In *International Conference on Machine Learning-Volume 70*, pages 951–960.

De Lit, P., Falkenauer, E., and Delchambre, A. (2000). Grouping genetic algorithms: an efficient method to solve the cell formation problem. *Mathematics and Computers in simulation*, 51(3-4):257–271.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6:182–197.

Desautels, T., Krause, A., and Burdick, J. W. (2014). Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. *Journal of Machine Learning Research*, 15:3873–3923.

Dheeru, D. and Karra Taniskidou, E. (2017). UCI machine learning repository.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Duda, R. O., Hart, P. E., et al. (1973). *Pattern classification and scene analysis*, volume 3. Wiley New York.

Emmerich, A. (2008). The computation of the expected improvement in dominated hypervolume of Pareto front approximations. Technical Report LIACS TR-4-2008, Leiden University, The Netherlands.

Emmerich, M. and Klinkenberg, J. W. (2008). The computation of the expected improvement in dominated hypervolume of Pareto front approximations. *Rapport technique, Leiden University*.

Esteban, M. and Leary, D. (2012). Current developments and future prospects of offshore wind and ocean energy. *Applied Energy*, 90(1):128–136.

Fadaeenejad, M., Shamsipour, R., Rokni, S., and Gomes, C. (2014). New approaches in harnessing wave energy: With special attention to small islands. *Renewable and Sustainable Energy Reviews*, 29:345–354.

Falcao, A. (2010). Wave energy utilization: A review of the technologies. *Renewable and sustainable energy reviews*, 14(3):899–918.

Falkenauer, E. (1993). The grouping genetic algorithms: widening the scope of the ga's. *JORBEL-Belgian Journal of Operations Research, Statistics, and Computer Science*, 33(1-2):79–102.

Falkenauer, E. (1998). *Genetic algorithms and grouping problems*. John Wiley & Sons, Inc.

Féliot, P., Bect, J., and Vazquez, E. (2017). A Bayesian approach to constrained single-and multi-objective optimization. *Journal of Global Optimization*, 67:97–133.

Fernández-Sánchez, D., Garrido-Merchán, E. C., and Hernández-Lobato, D. (2020). Max-value entropy search for multi-objective Bayesian optimization with constraints. *arXiv preprint arXiv:2011.01150*.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., and Hutter, F. (2019). Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning*, pages 113–134. Springer, Cham.

Fonseca, C. M. and Fleming, P. J. (1998). Multiobjective optimization and multiple constraint handling with evolutionary algorithms. i. a unified formulation. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 28:26–37.

Frazier, P. I. (2018). A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *Annals of statistics*, pages 1189–1232.

Gardner, J. R., Kusner, M. J., Xu, Z. E., Weinberger, K. Q., and Cunningham, J. P. (2014). Bayesian optimization with inequality constraints. In *International Conference on Machine Learning*, pages 937–945.

Garnett, R., Osborne, M. A., and Roberts, S. J. (2010). Bayesian optimization for sensor set selection. In *Proceedings of the 9th ACM/IEEE international conference on information processing in sensor networks*, pages 209–219.

Garrido-Merchán, E. and Albarca-Molina, A. (2018). Suggesting cooking recipes through simulation and Bayesian optimization. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 277–284. Springer.

Garrido-Merchán, E. and Hernández-Lobato, D. (2019a). Dealing with categorical and integer-valued variables in Bayesian optimization with Gaussian processes. *Neurocomputing*.

Garrido-Merchán, E. and Hernández-Lobato, D. (2019b). Predictive entropy search for multi-objective Bayesian optimization with constraints. *Neurocomputing*.

Garrido-Merchán, E. C. and Hernández-Lobato, D. (2020). Parallel predictive entropy search for multi-objective Bayesian optimization with constraints. *arXiv preprint arXiv:2004.00601*.

Gelbart, M. A., Snoek, J., and Adams, R. P. (2014). Bayesian optimization with unknown constraints. In *Uncertainty in Artificial Intelligence*, pages 250–259.

Gelman, A., Vehtari, A., Jylänki, P., Robert, C., Chopin, N., and Cunningham, J. P. (2014). Expectation propagation as a way of life. *arXiv preprint arXiv:1412.4869*, 157.

Gerven, M., Cseke, B., Oostenveld, R., and Heskes, T. (2009). Bayesian source localization with the multivariate Laplace prior. In *Advances in Neural Information Processing Systems*, pages 1901–1909.

Ghahramani, Z. (2006). *Information theory*. Wiley Online Library.

Glover, F. and Kochenberger, G. (2006). *Handbook of metaheuristics*, volume 57. Springer Science & Business Media.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. (2018). Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276.

González, J. (2016). Gpyopt: a Bayesian optimization framework in python.

González, J., Dai, Z., Damianou, A., and Lawrence, N. D. (2017). Preferential Bayesian optimization. *arXiv preprint arXiv:1704.03651*.

González, J., Dai, Z., Hennig, P., and Lawrence, N. (2016). Batch Bayesian optimization via local penalization. In *Artificial intelligence and statistics*, pages 648–657.

Gonzalvez, J., Lezmi, E., Roncalli, T., and Xu, J. (2019). Financial applications of Gaussian processes and Bayesian optimization. *arXiv preprint arXiv:1903.04841*.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Griffiths, R. and Hernández-Lobato, J. (2020). Constrained Bayesian optimization for automatic chemical design using variational autoencoders. *Chemical Science*.

Gumbel, E. J. (1958). *Statistics of extremes*. Columbia university press.

Gupta, S., Shilton, A., Rana, S., and Venkatesh, S. (2018). Exploiting strategy-space diversity for batch Bayesian optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 538–547.

Hammersley, J. (2013). *Monte Carlo methods*. Springer Science & Business Media.

Havasi, M., Hernández-Lobato, J., and Murillo-Fuentes, J. (2018). Inference in deep Gaussian processes using stochastic gradient hamiltonian monte carlo. In *Advances in Neural Information Processing Systems*, pages 7506–7516.

Hebbal, A., Brevault, L., Balesdent, M., Talbi, E.-G., and Melab, N. (2019). Bayesian optimization using deep gaussian processes. *arXiv preprint arXiv:1905.03350*.

Hebbal, A., Brevault, L., Balesdent, M., Talbi, E.-G., and Melab, N. (2020). Bayesian optimization using deep gaussian processes with applications to aerospace system design. *Optimization and Engineering*, pages 1–41.

Hennig, P. and Schuler, C. (2012). Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(Jun):1809–1837.

Hernández-Lobato, D., Hernandez-Lobato, J. M., Shah, A., and Adams, R. P. (2016). Predictive entropy search for multi-objective Bayesian optimization. In *International Conference on Machine Learning*, pages 1492–1501.

Hernández-Lobato, D., Martínez-Muñoz, G., and Suárez, A. (2009). Statistical instance-based pruning in ensembles of independent classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:364–369.

Hernández-Lobato, J., Gelbart, M., R.P., A., M.W., H., and Ghahramani, Z. (2016). A general framework for constrained Bayesian optimization using information-based search. *Journal of Machine Learning Research*, 17:1–53.

Hern'andez-Lobato, J. M. and Adams, R. (2015). Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869. PMLR.

Hernández-Lobato, J. M., Gelbart, M. A., Hoffman, M. W., Adams, R. P., and Ghahramani, Z. (2015). Predictive entropy search for Bayesian optimization with unknown constraints. In *International Conference on Machine Learning*, pages 1699–1707.

Hernández-Lobato, J. M., Gelbart, M. A., Reagen, B., Adolf, R., Hernández-Lobato, D., Whatmough, P., Brooks, D., Wei, G.-Y., and Adams, R. P. (2016). Designing neural network hardware accelerators with decoupled objective evaluations.

Hernández-Lobato, J. M., Hoffman, M., and Ghahramani, Z. (2014). Predictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems*, pages 918–926.

Ho, Y. and Pepyne, D. (2002). Simple explanation of the no-free-lunch theorem and its implications. *Journal of optimization theory and applications*, 115(3):549–570.

Hoffman, M., Brochu, E., and de Freitas, N. (2011). Portfolio allocation for Bayesian optimization. In *UAI*, pages 327–336. Citeseer.

Houlsby, N., Hernández-Lobato, J. M., Huszar, F., and Ghahramani, Z. (2012). Collaborative Gaussian processes for preference learning. In *Advances in Neural Information Processing Systems*, pages 2096–2104.

Huang, G., , Zhu, Q., and Siew, C. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501.

Hutter, F. (2009). *Automated configuration of algorithms for solving hard computational problems*. PhD thesis, University of British Columbia.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer.

Hutter, F., Kotthoff, L., and Vanschoren, J. (2019). *Automated machine learning: methods, systems, challenges*. Springer Nature.

Jaffar, J. and Maher, M. (1994). Constraint logic programming: A survey. *The journal of logic programming*, 19:503–581.

James, T., Brown, E., and Keeling, K. (2007a). A hybrid grouping genetic algorithm for the cell formation problem. *Computers & Operations Research*, 34(7):2059–2079.

James, T., Vroblefski, M., and Nottingham, Q. (2007b). A hybrid grouping genetic algorithm for the registration area planning problem. *Computer Communications*, 30(10):2180–2190.

Jamil, M. and Yang, X. (2013). A literature survey of benchmark functions for global optimization problems. *arXiv preprint arXiv:1308.4008*.

Jones, D., Schonlau, M., and Welch, W. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233.

Jylänki, P., Nummenmaa, A., and Vehtari, A. (2014). Expectation propagation for neural networks with sparsity-promoting priors. *The Journal of Machine Learning Research*, 15(1):1849–1901.

Kathuria, T., Deshpande, A., and Kohli, P. (2016). Batched Gaussian process bandit optimization via determinantal point processes. In *Advances in Neural Information Processing Systems*, pages 4206–4214.

Ketkar, N. (2017). Introduction to pytorch. In *Deep learning with python*, pages 195–208. Springer.

Kim, S. and Suh, K. (2014). Determining the stability of vertical breakwaters against sliding based on individual sliding distances during a storm. *Coastal engineering*, 94:90–101.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Knowles, J. (2006). Parego: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *Evolutionary Computation, IEEE Transactions on*, 10:50–66.

Kochanski, G., Golovin, D., Karro, J., Solnik, B., Moitra, S., , and Sculley, D. (2017). Bayesian optimization for a better dessert.

Kotthoff, L., Thornton, C., Hoos, H., Hutter, F., and Leyton-Brown, K. (2017). Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *The Journal of Machine Learning Research*, 18(1):826–830.

Kushner, H. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106.

Kuss, M. and Rasmussen, C. E. (2005). Assessing approximate inference for binary Gaussian process classification. *Journal of machine learning research*, 6(Oct):1679–1704.

LeCun, Y. (1998). The MNIST Database of Handwritten Digits. *http://yann. lecun. com/exdb/mnist/*.

LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. at&t labs.

Lévesque, J., Durand, A., Gagné, C., and Sabourin, R. (2017). Bayesian optimization for conditional hyperparameter spaces. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 286–293. IEEE.

Li, C., Gupta, S., Rana, S., Nguyen, V., Venkatesh, S., and Shilton, A. (2018). High dimensional Bayesian optimization using dropout. *arXiv preprint arXiv:1802.05400*.

Liu, L., Wang, D., and Peng, Z. (2016). Path following of marine surface vehicles with dynamical uncertainty and time-varying ocean disturbances. *Neurocomputing*, 173:799–808.

Lizotte, D. (2008). *Practical Bayesian optimization*. University of Alberta.

Lizotte, D. J., Wang, T., Bowling, M. H., and Schuurmans, D. (2007). Automatic gait optimization with Gaussian process regression. In *IJCAI*, volume 7, pages 944–949.

López, I., Andreu, J., Ceballos, S., De Alegría, I., and Kortabarria, I. (2013). Review of wave energy technologies and the necessary power-equipment. *Renewable and sustainable energy reviews*, 27:413–434.

Lubinsky, D. S. and Rabinowitz, P. (1984). Rates of convergence of Gaussian quadrature for singular integrands. *mathematics of computation*, 43(167):219–242.

Lyu, W., Yang, F., Yan, C., Zhou, D., and Zeng, X. (2018). Batch bayesian optimization via multi-objective acquisition ensemble for automated analog circuit design. In *International conference on machine learning*, pages 3306–3314. PMLR.

MacKay, D. (1992). A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472.

MacKay, D. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

MacKay, D. J. (1995). Probable networks and plausible predictions—a review of practical Bayesian methods for supervised neural networks. *Network: computation in neural systems*, 6(3):469–505.

Maclaurin, D., Duvenaud, D., and Adams, R. P. (2015). Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML Workshop*, volume 238, page 5.

Markov, S. (2017). Skopt documentation.

Martínez-Muñoz, G. and Suárez, A. (2005). Switching class labels to generate classification ensembles. *Pattern Recognition*, 38:1483–1494.

Minka, T. (2001a). Expectation propagation for approximate Bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc.

Minka, T. and Lafferty, J. (2012). Expectation-propogation for the generative aspect model. *arXiv preprint arXiv:1301.0588*.

Minka, T. P. (2001b). *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology.

Mockus, J., Tiesis, V., and Zilinskas, A. (1978). The application of Bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2.

Moreno-Muñoz, P., Artés, A., and Alvarez, M. (2018). Heterogeneous multi-output Gaussian process prediction. In *Advances in neural information processing systems*, pages 6711–6720.

Murphy, K. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Murray, I. and Adams, R. P. (2010). Slice Sampling Covariance Hyperparameters of Latent Gaussian models.

Neal, R. (2003). Slice sampling. *Annals of statistics*, pages 705–741.

Neal, R. (2012). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

Nickisch, H. and Rasmussen, C. E. (2008). Approximations for binary Gaussian process classification. *Journal of Machine Learning Research*, 9(Oct):2035–2078.

Opper, M. and Archambeau, C. (2009). The variational Gaussian approximation revisited. *Neural computation*, 21(3):786–792.

Osman, I. and Laporte, G. (1996). Metaheuristics: A bibliography.

Pardalos, P. and Romeijn, H. (2013). *Handbook of global optimization*, volume 2. Springer Science & Business Media.

Parr, J. (2013). *Improvement criteria for constraint handling and multiobjective optimization*. PhD thesis. University of Southampton.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, 12:2825–2830.

Pedregosa, F. e. a. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830.

Perrone, V., Donini, M., Kenthapadi, K., and Archambeau, C. (2020). Fair Bayesian optimization. *arXiv preprint arXiv:2006.05109*.

Picheny, V. (2015). Multiobjective optimization using Gaussian process emulators via stepwise uncertainty reduction. *Statistics and Computing*, 25:1265–1280.

Ponweiser, W., Wagner, T., Biermann, D., and Vincze, M. (2008). Multiobjective optimization on a limited budget of evaluations using model-assisted\ mathcal {S}-metric selection. In *Parallel Problem Solving from Nature–PPSN X*, pages 784–794.

Rahimi, A., Recht, B., et al. (2007). Random features for large-scale kernel machines. In *NIPS*, volume 3, page 5. Citeseer.

Rainforth, T., Le, T., van de Meent, J., Osborne, M., and Wood, F. (2016). Bayesian optimization for probabilistic programs. In *Advances in Neural Information Processing Systems*, pages 280–288.

Rana, S., Li, C., Gupta, S., Nguyen, V., and Venkatesh, S. (2017). High dimensional Bayesian optimization with elastic Gaussian process. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2883–2891. JMLR. org.

Rao, S. and Mandal, S. (2005). Hindcasting of storm waves using neural networks. *Ocean Engineering*, 32(5-6):667–684.

Rasmussen, C. (2003). Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer.

Ross, S., Kelly, J., Sullivan, R., Perry, W., Mercer, D., Davis, R., Washburn, T., Sager, E., Boyce, J., and Bristow, V. (1996). *Stochastic processes*, volume 2. Wiley New York.

Rusu, L. and Soares, C. (2012). Wave energy assessments in the azores islands. *Renewable Energy*, 45:183–196.

Salimbeni, H. and Deisenroth, M. (2017). Doubly stochastic variational inference for deep Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 4588–4599.

Schonlau, M., Welch, W. J., and Jones, D. R. (1998). Global versus local search in constrained optimization of computer models. *Lecture Notes-Monograph Series*, 34:11–25.

Seeger, M. (2005). Expectation propagation for exponential families.

Shah, A. and Ghahramani, Z. (2015). Parallel predictive entropy search for batch global optimization of expensive objective functions. In *Advances in Neural Information Processing Systems*, pages 3330–3338.

Shah, A. and Ghahramani, Z. (2016). Pareto frontier learning with expensive correlated objectives. In *International Conference on Machine Learning*, pages 1919–1927. PMLR.

Shah, A., Wilson, A., and Ghahramani, Z. (2014). Student-t processes as alternatives to Gaussian processes. In *Artificial intelligence and statistics*, pages 877–885.

Shah, A., Wilson, A. G., and Ghahramani, Z. (2013). Bayesian optimization using Student-t processes. In *NIPS Workshop on Bayesian Optimisation*.

Shahriari, B., Swersky, K., Wang, Z., Adams, R., and De Freitas, N. (2015). Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Shan, S. and Wang, G. G. (2005). An efficient pareto set identification approach for multiobjective optimization on black-box functions.

Shao, Y. S., Reagen, B., Wei, G., and Brooks, D. (2014). Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *International Symposium on Computer Architecture*, pages 97–108.

Shawe-Taylor, J., Cristianini, N., et al. (2004). *Kernel methods for pattern analysis*. Cambridge university press.

Siarry, P. and Collette, Y. (2003). Multiobjective optimization: principles and case studies.

Singh, H., Misra, N., Hnizdo, V., Fedorowicz, A., and Demchuk, E. (2003). Nearest neighbor estimates of entropy. *American journal of mathematical and management sciences*, 23:301–321.

Snelson, E. and Ghahramani, Z. (2005). Sparse Gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18:1257–1264.

Snelson, E. and Ghahramani, Z. (2006). Sparse Gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264.

Snoek, J. (2013). *Bayesian optimization and semiparametric models with applications to assistive technology*. PhD thesis. University of Toronto.

Snoek, J., Larochelle, H., and Adams, R. (2012). Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. P. (2015). Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pages 2171–2180.

Snoek, J., Swersky, K., Zemel, R., and Adams, R. (2014). Input warping for Bayesian optimization of non-stationary functions. In *International Conference on Machine Learning*, pages 1674–1682.

Solé, X., Ramisa, A., and Torras, C. (2014). Evaluation of random forests on large-scale classification problems using a bag-of-visual-words representation. In *CCIA*, pages 273–276.

Souza, A., Nardi, L., Oliveira, L. B., Olukotun, K., Lindauer, M., and Hutter, F. (2020). Prior-guided Bayesian optimization. *arXiv preprint arXiv:2006.14608*.

Springenberg, J., Klein, A., Falkner, S., and Hutter, F. (2016). Bayesian optimization with robust Bayesian neural networks. In *Advances in Neural Information Processing Systems*, pages 4134–4142.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. (2009). Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*.

Stein, M. L. (2012). *Interpolation of spatial data: some theory for Kriging*. Springer Science & Business Media.

Thornton, C., Hutter, F., Hoos, H., and Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855.

Titsias, M. (2009). Variational learning of inducing variables in sparse Gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574.

Törn, A. and Žilinskas, A. (1989). *Global optimization*, volume 350. Springer.

Tracey, B. D. and Wolpert, D. (2018). Upgrading from Gaussian processes to Student's t-processes. In *2018 AIAA Non-Deterministic Approaches Conference*, page 1659.

Villemonteix, J., Vazquez, E., and Walter, E. (2009). An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4):509.

Wang, F., Zhang, J., Zheng, X., Wang, X., Yuan, Y., Dai, X., Zhang, J., and Yang, L. (2016). Where does alphago go: From church-turing thesis to alphago thesis and beyond. *IEEE/CAA Journal of Automatica Sinica*, 3(2):113–120.

Wang, Z. and Jegelka, S. (2017). Max-value entropy search for efficient Bayesian optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3627–3635. JMLR. org.

Wang, Z., Zoghi, M., Hutter, F., Matheson, D., De Freitas, N., et al. (2013). Bayesian optimization in high dimensions via random embeddings. In *IJCAI*, pages 1778–1784.

While, L., Hingston, P., Barone, L., and Huband, S. (2006). A faster algorithm for calculating hypervolume. *IEEE transactions on evolutionary computation*, 10(1):29–38.

Williams, C. K. and Barber, D. (1998). Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351.

Yao, X., Liu, Y., and Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary computation*, 3(2):82–102.

Yoshimi, G. (2010). *Random Seas and Design of Maritime Structures*, volume 33. World Scientific Publishing Company.

Zheng, Z. and Sun, L. (2016). Path following control for marine surface vessel with uncertainties and input saturation. *Neurocomputing*, 177:158–167.

Zhu, C., Byrd, R., Lu, P., and Nocedal, J. (1997). Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560.

Zitzler, E. and Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE transactions on evolutionary computation*, 3:257–271.