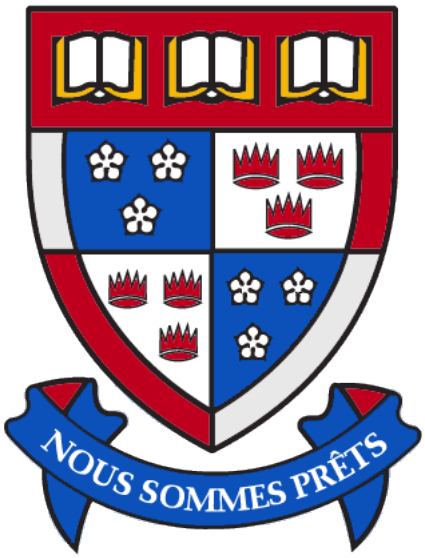

Simon Fraser University

Faculty of Sciences, Math Department

Here comes the title

Juan Gabriel García Osorio

Advisor: John Stockie

Commitee: Paul Tupper

Burnaby B.C - May 2017

Contents

Acknowlegments

Chapter 1

Introduction

Chapter 2

Theoretical and Computational Framework

The foundations of our approach to estimate parameters and solve inverse problems is the framework of Bayesian statistics. Unlike frequentist statistics, in the Bayesian approach, randomness is a measure of uncertainty, is not a matter of frequency . For instance a question like what is the probability of having life in Mars? If the answer is say 0.01, in the frequentist stastistics framework, this number is interpreted as, for every hundred times we go to mars under identical circumstances, one of those times we will find life there. In the Bayesian approach the number 0.01 is interpreted as: given our current level of knowledge about mars we think that is very unlikely that mars shelters life. Clearly there is a big phylosofical difference between these two approaches that has a direct impact in how far reaching is each point of view [5].

In real life the uncertainty associated to a measurement or quantity of interest is usually connected with the uncertainty of other variables involved in the problem under study. In general the connection between the different variables is far from trivial and hard to assess. Bayesian methodology provides a rigurous framework for the study of this connection, using whatever information is available for the problem. The cornerstone of this idea in the mathematical language is the Bayes formula

$$\mathbb{P}_{post}(A|B) = \frac{\mathbb{P}_{like}(B|A)\mathbb{P}_{prior}(A)}{\mathbb{P}(B)}. \tag{2.1}$$

To explain in detail the above equation, let us first do the following defin-

tion [3]

Definition 1. *A probability space is a triple* $(\Omega, \mathscr{F}, \mathbb{P})$*, where* Ω *is a set called sample space. The element* $\mathscr{F}$ *is a subset of the power set of* Ω *that satisfies*

1. $\emptyset, \Omega \in \mathscr{F}$.
2. *If* $A \in \mathscr{F}$ *then* $A^c \in \mathscr{F}$.
3. *If* $A_1, A_2, \ldots \in \mathscr{F}$ *then* $\bigcup_{i\in\mathbb{N}} A_i \in \mathscr{F}$.

A set that satisfies properties 1 to 3 is called a $\sigma-$ *algebra and its elements are called events. The map* $\mathbb{P} : \mathscr{F} \to [0, 1]$ *satisfies*

1. $\mathbb{P}(\Omega) = 1$.
2. *If* $A_1, A_2, \ldots \in \mathscr{F}$ *are pairwise disjoint, then*

$$\mathbb{P}(\bigcup_{i\in\mathbb{N}} A_i) = \sum_{i\in\mathbb{N}} \mathbb{P}(A_i).$$

$\mathbb{P}$ *is called a probability measure.*

In equation (2.1), $\mathbb{P}, \mathbb{P}_{prior}, \mathbb{P}_{like}, \mathbb{P}_{post}$ are different probability measures defined in the same sample space Ω. The sets A and B are subsets of the sample space Ω and are elements of the associated $\sigma-$algebra $\mathscr{F}$. The bar in the probability measures e.g. $\mathbb{P}_{like}(\cdot|\cdot)$, means conditional probability. Let us introduce some terminology: the term $\mathbb{P}_{like}(B|A)$ is called the *likelihood* of B given A, $\mathbb{P}_{prior}(A)$ is called the *prior* for A. The prior expreses how much we believe the event A to happen without assuming anything about how the event B might influence the event A. The term $\mathbb{P}(B)$ is a *normalization constant* defined as

$$Z(B) = \int_\Omega \mathbb{P}_{like}(B|A)\mathbb{P}_{prior}(A)d\mathbb{P}.$$

The term $\mathbb{P}_{post}(A|B)$ is the *posterior*.

Simply put formula (2.1) implies: the probability that the event A given that the event B has already occurred is proportional to the probability of the event B given that the event A has already happened weighted by the prior probability of A.

This interpretation of Bayes rule serves as a motivation for using it in the field of inverse problems. Inverse problems are of the type 'find the cause of this effect'. This name is relative to what is known as the direct problem. These are problems of the type 'find the effect of this cause'. Often inverse problems ill-posed, this means that these problems might not satisfy one or more of the following properties [8]:

- Existence: There exists a solution for the problem.
- Uniqueness: The problem has a unique solution.
- Stability: Small changes in inputs result in small changes in outputs.

This is a serious issue. If the problem under study has at least one solution but is not stable, for example, how can we assess the accuracy of the result obtained? An statistical approach is called for under this setting. But the most important reason of why the Bayesian framework is useful to solve inverse problems is that Bayes rule unifies the problem of 'find the cause of this effect' (e.g. finding the posterior) with the direct problem of 'find the effect of this cause' (e.g. finding the likelihood).

Let us be precise of how the Bayesian approach can be used to solve inverse problems with an example. Consider the problem of finding the location where a rock that smashed a window was thrown. We can start by considering the following events

A=x,y,z coordinates where the rock was thrown.
B=x,y,z coordinates where the rock hit the window.

In this case Bayes formula tells us that if we want to estimate the likely locations where the rock was thrown from (find the posterior probability), given that we know the coordinates where it hit the window we need to include our prior believe on where do we think the rock was actually thrown (Setting a prior probability). Also it is necessary to find the connection with the direct problem. In this case the direct problem is to find where the rock hit the window assuming that we know where it was thrown (find the likelihood probability)

Let us evaluate how we could estimate the different probabilities mentioned in the previous paragraph. First, to find the likelihood we need to know how the rock's impact position in the window is related to the launch

location . If we treat air resistance as a source of uncertainty we can use the kinematics equations for parabolic trajectories to get [1]

$$\mathbf{r} = \mathbf{r}_0 + \mathbf{v}_0 t + \frac{1}{2}\mathbf{g}t^2, \tag{2.2}$$

where $\mathbf{r}$ and $\mathbf{r}_0$ are the final and initial position of the rock respectively, $\mathbf{v}_0$ is the initial velocity and $\mathbf{g}$ is the gravity vector. The scalar t represents time. In a more physical language, to estimate the likelihood it is necessary to estimate $\mathbf{r}$ (where the rock hit the window) assuming we know $\mathbf{r}_0$ (where it was thrown). To do so, we need to estimate the initial velocity of the rock $\mathbf{v}_0$. Once all the other variables are identified the value of t can be computed in a straightforward manner.

Equations in physics are just models of reality and as such are just an approximation to it. To take this into account we add an extra layer to the model by adding a random parameter that accounts for the discrepancy of our model. We propose

$$\mathbf{r} = \mathbf{r}_0 + \mathbf{v}_0 t + \frac{1}{2}\mathbf{g}t^2 + \epsilon,$$

where ϵ is a random vector distributed as $\vec{\epsilon} \sim \mathscr{N}(0, \sigma I)$. Here I represents the 3×3 identity matrix and $\sigma > 0$ parametrizes one belief in quantifing the accuracy of equation (2.2). By introducing a random variable into the model we can now think of all of the variables involved in equation (2.2) as random, that is, we now look at the associated stochastic equation. With this notation we can cast equation (2.1) into (assuming independence between $\mathbf{r}_0$ and $\mathbf{v}_0$)

$$\mathbb{P}_{post}(\mathbf{r}_0|\mathbf{r}, \mathbf{v}_0) = \frac{\mathbb{P}_{like}(\mathbf{r}|\mathbf{r}_0, \mathbf{v}_0)\mathbb{P}_{prior}(\mathbf{r}_0)}{\mathbb{P}(\mathbf{r}|\mathbf{v}_0)}.$$

Since ϵ is Gaussian we can readily obtain [15]

$$\mathbf{r}|\mathbf{r}_0, \mathbf{v}_0 \sim \mathscr{N}(\mathbf{r}_0 + \mathbf{v}_0 t + \frac{1}{2}\mathbf{g}t^2, \sigma I).$$

This equation gives an explicit density for the likelihood. We now turn our attention to model the prior. Suppose that we suspect that the rock was thrown from the bedroom of one of our neighboors. One way to model this suspicion is to assume a prior distribution on $\mathbf{r}_0$ as

$$\mathbf{r}_0 \sim \mathscr{N}(\mathbf{z}, \lambda I),$$

where $\mathbf{z}$ is the coordinate vector of the center of mass of neighbohr's room and λ represents the variance of the launch location around the point $\mathbf{z}$. We note that this is one way to model prior knowledge and other priors are also possible for our problem.

The last quantity to find is the normalization constant $\mathbb{P}(\mathbf{r}|\mathbf{v}_0)$. Since all probabilities are normalized we get

$$\begin{aligned} 1 &= \int_{\mathbb{R}^3} \mathbb{P}_{post}(\mathbf{r}_0|\mathbf{r}, \mathbf{v}_0) d\mathbf{r}_0 \\ &= \int_{\mathbb{R}^3} \frac{\mathbb{P}_{like}(\mathbf{r}|\mathbf{r}_0, \mathbf{v}_0)\mathbb{P}_{prior}(\mathbf{r}_0)}{\mathbb{P}(\mathbf{r}|\mathbf{v}_0)} d\mathbf{r}_0. \end{aligned}$$

Since the denominator in the last integrand is a constant with respect to the variable of integration we conclude

$$\mathbb{P}(\mathbf{r}|\mathbf{v}_0) = \int_{\mathbb{R}^3} \mathbb{P}_{like}(\mathbf{r}|\mathbf{r}_0, \mathbf{v}_0)\mathbb{P}_{prior}(\mathbf{r}_0) d\mathbf{r}_0.$$

To evaluate our model, we decide to write a computer code that simulates (2.2) for a wide range of different values of $\vec{r}, \vec{v_0}$ fixing $\vec{r_0}$. Then simulate (2.2) fixing a different value of r_0 and simulating for the same range of $\vec{r}, \vec{v_0}$ and so on. With the data obtained from the simulation we can finally evaluate the posterior $P(\vec{r_0}|\vec{r}, I)$. Recalling that in general the posterior is a probability density, we can use it to estimate in several ways, what we consider to be the best estimate of where the rock was thrown and the uncertainty attached to that estimate . For instance we can pick any of these estimates [15]

$$\begin{aligned} \vec{r}_{MAP} &= argmax_{\vec{r_0}\in\mathbb{R}^3} P(\vec{r_0}|\vec{r}), \\ \vec{r}_{CM} &= \int_{\mathbb{R}^3} \vec{r_0} P(\vec{r_0}|\vec{r}) d\vec{r_0}, \\ \vec{r}_{ML} &= argmax_{\vec{r_0}\in\mathbb{R}^3} P(\vec{r}|\vec{r_0}). \end{aligned}$$

In these equations $\vec{r_{MAP}}$ is known as the maximum a posteriori estimate, $\vec{r}_{CM}$ is the conditional mean estimate and $\vec{r}_{ML}$ is the maximum likelihood estimate.

Voilá! We solved our crime, we have a way to estimate where the rock was thrown and a way to measure how confident we are about it. The posterior is what is called a solution in Bayesian inverse problems.

Unfortunately in real life things are not as easy as were described here in the above example. In reality we have to deal with several issues such as

1. Uncertainties in experimental measurements difficult to assess.
2. Not enough information to create a reliable estimate.
3. Computational complexity of the physical model makes prohibitive too many simulations.
4. The dimensionality of the probabilities involved is high so sampling from them is far from trivial.
5. Evaluating any of the possible estimates for the quantity of interest might be very hard.

Just to name a few of the issues that arises when dealing with Bayesian inverse problems. To solve each one of this issues we have available tons of different approaches. In the problem described in the introduction of this work (Chapter 1), all of the issues mentioned above come into play. In this chapter we are going to discuss from a theoretical and computational point of view how to deal with issues 3, 4 and 5 of the list.

2.1 Dealing with the computational complexity of the physical model

Physical models of the reality are usually complex and when numerically simulated, are computationally intense. Following O'Hagan in [11], we can think of a physical or mathematical model as a function $f(\cdot)$ that takes a vector x of inputs and gives back a vector y of outputs. But as mentioned before models are just that, models, so the question is: how to evaluate the uncertainty associated to the model? Of course when we talk about uncertainty we are talking about every possible source of randomness in the model or lack of information of it. That is, the use of the word uncertainty in this work is related to either epistemic or aleatory [7].

Besides uncertainty there is another very important concept: sensitivity. When looking at a model's performance it is critical to assess how changes or uncertainties in the input x affect the output y. Therefore if we assess

sensitivity of the model we can assess uncertainty of the outputs. One more reason one might be interest in performing a sensitivity analysis is to detect what variables are relevant and what variables are not, allowing to reduce the dimensionality of the problem .

As mentioned in the introduction chapter the physical model we are dealing with is complex and computationally intense. This means that assessing the uncertainty and sensitivity associated via classical methods as in [14] is not feasible.Here is where the concept of emulator as defined in [11] comes into play. The idea is to approximate the function $f(\cdot)$ with an approximation function $\hat{f}(\cdot)$. But the approximation is not any approximation, is what we call and statistical approximation. The idea of this statistical approximation is to associate a probability distribution to each value $f(x)$ then take the mean of this distribution as $\hat{f}(x)$, for example.
Now that we have an idea of what we want as an approximation for the emulator, how can we construct it? Following [11] the idea to create the emulator is a process that satisfies

- At points x were we know the output of the physical model i.e. $f(x)$ we want to approximate the emulator at those points should satisfy $\hat{f}(x) = f(x)$ with uncertainty zero.
- For the points x^* where we don't know the output $f(x^*)$, the emulator should give back an estimate $\hat{f}(x^*)$, based on the distribution for $f(x^*)$. That estimate should reflect the uncertainty associated with the interpolation/extrapolation done at that point.

In Bayesian analysis of computer code output one of the methods to produce an emulator with the desired extrapolation/interpolation properties is what is known as a Gaussian process (GP).

2.1.1 Gaussian Processes

Definition 2. *A GP is a collection of random variables $\{f(x)\}_{x\in X}$, for some index family X, such that any finite set of them has a jointly Gaussian distribution [13].*

A GP is specified by two quantities, the mean function and the covariance function. Following the same notation as in Rasmussen [13] we set

$$m(x) = \mathbb{E}(f(x)),$$
$$k(x, x') = \mathbb{E}((f(x) - m(x))(f(x') - m(x'))).$$

So if $\{f(x)\}$ is a GP with mean $m(x)$ and covariance $k(x, x')$ we will write

$$f(x) \sim \mathbf{GP}(m(x), k(x, x')).$$

To understand the notion of a GP observe that what we are trying to do is to approximate a function $f(\cdot)$, then the symbol $f(x)$ represents a random variable whose realizations are the possible values of the function $f(\cdot)$ at x and $m(x)$ is the best approximation we can do to predict the actual value of $f(\cdot)$ at that point x. The uncertainty associated to that prediction is going to be related to the quantity $k(x, x)$.

Since we are going to use GP to fit data in a high dimensional euclidean space, it will be convenient to think of the set of index X in definition 2 as $\mathbb{R}^n$ for some $n \geq 1$.

The reason why the definition of a Gaussian process is useful and powerful is because a GP is completely characterized by $m(x)$ and $k(x, x')$[1] [9]. Therefore to fit and predict using GPs we only need to define just two quantities. For example the way to choose the covariance kernel is with the aid of some real valued function. For example a very common covariance function is given by what is called the squared exponential (SE)

$$k(x, x') = e^{-\frac{1}{2}\|x-x'\|_2^2}. \tag{2.3}$$

In this case this definition of the covariance is telling us that points that are close to each other are highly correlated whereas far away points has a correlation that decays exponentially fast with distance. There are some 'standard' ways to choose the covariance function depending on the kind of regularity we want for the GP fitting. To see what is the relation between regularity of the fitting and the kernel we need to realize that once the covariance function has been specified we end up with a distributions over some function space. Talking about distribution over spaces of functions is a delicate topic that has to be done in a rigorous manner. That is why at this point it is convenient to digress for a moment an talk about distributions in function spaces. The details presented here follow from [9].

Distributions Over Function Spaces

Since in general the relevant function spaces (e.g. L^p spaces, Sobolev spaces, etc...) are normed vector spaces, they have a topology inherited from the

[1] $k(x, x')$ is usually known as the kernel of the GP

metric induced by the norm, therefore in general, function spaces are topological vector spaces (TVS).

If $\mathscr{X}$ is a TVS and $\mathscr{X}^*$ is its dual, we will denote the action of an element $f \in \mathscr{X}^*$ over an element $x \in \mathscr{X}$ as $\langle f, x\rangle$. On the other hand we define a random vector taking values in $\mathscr{X}$ as a measurable map

$$X : (\Omega, \mathscr{F}, P) \longrightarrow \mathscr{X},$$

where $(\Omega, \mathscr{F}, P)$ is a probability space. To say that a random variable X takes vales in $\mathscr{X}$ we will write $X \in \mathscr{X}$. Therefore by letting $\mathscr{X} = L^2(\mathbb{R})$, for example, we now understand what does it mean $X \in L^2(\mathbb{R})$.

On the other hand we say that a random vector $X \in \mathscr{X}$ is called Gaussian if $\langle f, x\rangle$ is a normally distributed random variable for all $f \in \mathscr{X}^*$. Observe that x represents a realization of X, so the definition of a Gaussian vector in $\mathscr{X}$ makes sense. For normally distributed random vectors in $\mathbb{R}^n$ we have quantities like expectation and covariance matrix. In a more general (possible infinite dimensional) framework we say that a vector $a \in \mathscr{X}$ is the expectation of $X \in \mathscr{X}$ if

$$\mathbb{E}(f, X) = \langle f, a\rangle, \qquad \text{for all } f \in \mathscr{X}^*.$$

Also a linear operator $K : \mathscr{X}^* \longrightarrow \mathscr{X}$ is called the covariance operator (e.g covariance matrix in the finite dimensional case) if

$$cov(\langle f_1, X\rangle, \langle f_2, X\rangle) = \langle f_1, K f_2\rangle,$$

for all $f_1, f_2 \in \mathscr{X}^*$. It can be shown that the covariance operator has the same nice properties as in the finite dimensional case, such as symmetry and non-negative definiteness.

With the construction of a rigorous framework that allows us to talk about Gaussian distributions in function spaces we can see why once the covariance function and the mean function are defined then we have created a Gaussian distribution over some function of spaces and why the nature of the function space depends heavily on the regularity properties of the covariance kernel. In this work we are going to be mainly concerned to work with kernels that are at least continuous (For example the SE kernel in equation (2.3) is C^∞). To see why by just giving a recipe for the covariance and the mean defines a distribution over some function spaces, consider the following case, which is

our case of interest in this work. Since we are interest in at least continuous covariance functions consider $\mathscr{X} = \mathbb{C}(T)$ where $T \subset \mathbb{R}^n$ and T is compact. The space of real valued continuous functions in T is a Banach space with the norm [2]

$$\|g\| = \max_{x \in T} |g(x)|.$$

The dual space of $\mathscr{X}$ is given by $\mathscr{X}^* = \mathbb{M}(T)$ the set of sign measures on T. In this case the duality is given by

$$\langle \mu, g \rangle = \int_T g d\mu.$$

Therefore by having a GP, $\{f(x)\}_{x \in T}$ (see definition 2) with mean function $m(x)$ and covariance function $k(x, x')$, we can see the GP as a Gaussian random variable $f \in \mathscr{X}$ with mean and covariance operator defined as [9]

$$\begin{aligned} \mathbb{E}(f) &= m \in \mathbb{C}(T), \\ (K\nu)(s) &= \int_{\mathbb{R}^n} k(x, x')\nu(dx'), \qquad \text{for } \nu \in \mathbb{M}(T). \end{aligned}$$

With this mathematical framework to work with GP, it is possible to explain how they work. Imagine that we have some experimental measurements or emulator results (train inputs) $\{(x_i, f(x_i))\}_{i=1}^m \in \mathbb{R}^{n+1}$, for a real valued function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. Having these values we would like to make inference on the possible values of f on some set of points $\{x_j^*\}_{j=1}^k$ (test inputs). Since we assumed that $\{f(x)\}$ is a GP, that means that the vectors

$$\begin{aligned} \mathbf{f} &= \begin{bmatrix} f(x_1) & \dots & f(x_m) \end{bmatrix}^T, \\ \mathbf{f}_* &= \begin{bmatrix} f(x_1^*) & \dots & f(x_l^*) \end{bmatrix}^T. \end{aligned}$$

Have a jointly Gaussian distribution given by (assuming no trend in the data, i.e. mean zero).

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathscr{N}\left(0, \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right), \tag{2.4}$$

where $(K(X, X))_{ij} = cov(f(x_i), f(x_j)), K(X, X_*)_{ij} = cov(f(x_i), f(x_j^*))$ and so forth.

Since we know the vector $\mathbf{f}$ and we want to make inferences about the vector $\mathbf{f}_*$, then we are looking for the distribution of $\mathbf{f}_*|\mathbf{f}$. By well known properties of the multivariate Gaussian distribution we can check that [10]

$$\mathbf{f}_*|\mathbf{f} \sim \mathscr{N}\left(K(X_*,X)K(X,X)^{-1}\mathbf{f}, K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*)\right). \tag{2.5}$$

So to interpolate the function along the test inputs we can choose the mean of the distribution of $\mathbf{f}_*|\mathbf{f}$ as the interpolant. Also we can have a good idea of how reliable if the fitting by looking at the covariance matrix. As we can see choosing the covariance kernel to create the covariance matrix is a crucial step in the fitting process. How do we choose the kernel? There are some standard ways to do that and some standard kernels to choose from. Some of the most common kernels are (setting $r = \|x - x'\|_2$)

- Gauss: $k(r;\theta) = e^{-\frac{1}{2}(\frac{r}{\theta})^2}$
- Exponential: $k(r;\theta) = e^{-\frac{r}{\theta}}$
- Matern $\frac{3}{2}$: $k(r;\theta) = (1 + \frac{\sqrt{3}r}{\theta})e^{-\frac{\sqrt{3}r}{\theta}}$.
- Matern $\frac{5}{2}$: $k(r;\theta) = (1 + \frac{\sqrt{5}r}{\theta} + \frac{5}{3}(\frac{r}{\theta})^2)e^{-\frac{\sqrt{5}r}{\theta}}$.
- Power-Exponential: $k(r;\theta,p) = e^{-(\frac{r}{\theta})^p}$

As we can see covariance kernels are usually defined in terms of parameters, so depending on the model under study we can find the parameters that best suits the model. The question now is: how to choose the right parameters for the model? Consider the following situation: we have a set of training points $\{(x_i, f(x_i))\}_{i=1}^n$ and we want to predict the outputs $f(x)$ for some points in x that is not in the training set. To do the prediction using GPs we choose some kernel $k(r;\theta)$ that depends on the parameter θ (θ could be an scalar, vector, etc.). In this case to predict the values of $y = \{f(x_1^*), \dots f(x_m^*)\}$, we can use Bayes rule and optimize the likelihood, mathematically we want to optimize

$$p(y|\{(x_i, f(x_i))\}_{i=1}^n, \theta).$$

By definition 2 we know that the conditional probability has to be distributed as a multivariate normal distribution[2], hence by taking the log of the probability we get

$$\log(p(y|\{(x_i, f(x_i))\}_{i=1}^n, \theta)) = -\frac{1}{2}y^T K_y(\theta)^{-1}y - \frac{1}{2}\log|K_y(\theta)| - \frac{n}{2}\log(2\pi). \tag{2.6}$$

Here $K_y(\theta)$ is the covariance matrix associated to the GP conditional to $\{(x_i, f(x_i))\}$ (see equation (2.4)). In this case we can see explicitly the dependence on the parameter θ for the prediction of the Gaussian Process. Since we want to do the best possible prediction, it makes sense to try to optimize equation (2.6) with respect to the parameter θ. This gives a reliable criteria on how to tune up the parameters of the chosen kernel.

If we are dealing with a complex high-dimensional physical model that is emulated by $\hat{f}$, we need to be very careful of what points to choose as our training points for our GP. Usually training points come from the output of the simulation of the physical model through the emulator $\hat{f}$, this is a problem. Ideally we would like to pick as many training points as possible to make the fitting better, but picking too many points to do the interpolation might result in a very high computational demand, on the other hand if we pick up just a few points to do the emulation and then do the fitting with those points, this might result in a really bad fitting with the consequence of obtaining unreliable inferences when using GPs to fit the test points. This shows that we need a systematic way to choose how to get the training points. We need to choose those points in a way that we can fill as much space as possible with as few training points as possible without sacrificing quality in the prediction for the test points. This can be accomplish through something known as space filling designs.

2.1.2 Experimental Design and Sensitivity Analysis

At this point it is clear that if we want to use GPs to fit the data obtained through an emulator $\hat{f}(x)$ we need to decide how many runs of the emulator we are going to do. If the domain of the emulator is a subset of $\mathbb{R}^n$ then

[2]Recall that the $n-dimensional$ Gaussian measure $N(\mu, \Sigma)$ has Lebesgue density given by $\frac{1}{\sqrt{(2\pi)^n|\Sigma|}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

we need to decide what points in the domain we are going to choose as well. This is a very delicate issue since we need to find the right balance between the number of possible emulations given time and computational power constraints and a good distribution in space of these training points to get a good fitting in the output on every test point we choose.

Given an set T, there are several ways to create space filling designs for it [12]. In this work we are going to focus on maximin designs [6]. To define this kind of design, consider a metric space (T, d) (e.g. $T \subset \mathbb{R}^n$, compact and d the Euclidean distance) and any subset of T, S with finite (fixed) cardinality, say $|S| = n$. A maximim distance design S^o is such that

$$\max_{S} \min_{s,s'} d(s, s') = \min_{s,s' \in S^o} d(s, s') = d^o.$$

That is, we are looking for the set S^o of cardinality n that maximizes the minimum distance among its elements. As an example consider $T = [0, 1]^3$, the unit cube in $\mathbb{R}^3$ and $n = 8$, in that case the design that maximizes the minimum distance among its elements is given by choosing the points that live in each of the 8 vertex of the cube. In this case $d^o = 1$. The problem of finding the optimum in a maximin design is hard, therefore in general is necessary to use computational tools for optimization. In chapter 4 we will see how to use these computational tools to create a proper experimental maximin design.

The connexion of maximin experimental designs with GP is the following. Consider the GP $\{f(x)\}_{x \in T}$. If we fix S and let f_s to be the conditional distribution of $\{f(x)\}_{x \in T}$ given the realizations $f(s), s \in S$, if we let K_s to be the correlation of the distribution $f(s), s \in S$. Then it can be shown that the minimax design minimizes the quantity

$$M(S) = -det(K_s).$$

Since the covariance matrix is positive defined (hence the correlation matrix is also positive defined) , then $det(K_s) > 0$. By minimizing the negative of the determinant we are getting looking for the maximum value of the determinant. This is achieved when the column vectors of a matrix are close to orthogonal. In statistical terms what we are looking for is the least possible correlation between the difference points in the maximin design. Here is where its importance lies.

If we have a good space filling design, then we have a good fit when using GPs to interpolate the points we want to get information about. If we have a reliable fitting we have the chance to reduce the dimensionality of the model and hence its complexity. Indeed, assume that you want to fit a function $f : T \subset \mathbb{R}^n \longrightarrow \mathbb{R}$ and you want to know if all the n variables in the domain are relevant to understand the behaviour of f. For example if f behaves like the function $f(x_1, \ldots, x_n) = x_1 + x_2 + 10^{-8}x_3$ on $T = [0, 1]^n$, then clearly instead of working in an $n-$dimensional space and dealing with all its complexities we can reduce the model to a simple $2-$dimensional one. The question is then. How to quantify the dependence (or sensitivity) of f on $x_1, \ldots, x_n$? There are multiple ways to perform such sensitivity analysis [14]. In this work we are going to focus on the method introduced by Ilya Meyerovich Sobol. This method is what is called a variance-based Monte Carlo method (VBMCM). The idea of a VBMCM is to use the variance produced by the inputs of a function as an indicator of its importance. The description of Sobol's method follows from [14].

Witout lost of generallity we may assume that we have a function $f : \Omega^n \subset \mathbb{R}^n \longrightarrow \mathbb{R}$ where Ω^k is the $n-$dimensional unit cube[3]. Sobol's idea was to decompose f as

$$f(x_1, \ldots, x_n) = f_0 + \sum_{k=1}^{n} f_k(x_k) + \sum_{1 \leq k < l \leq n} f_{kl}(x_k, x_l) + \ldots + f_{1,2,\ldots,n}(x_1, \ldots, x_n).$$

The functions in this decomposition cannot be arbitrary functions. First f_0 has to be a constant. On the other hand the other functions must satisfy

$$\int_{[0,1]} f_{i_1,\ldots,i_s} dx_{i_k} = 0 \qquad \text{if } 1 \leq k \leq s.$$

Therefore by Fubinni's theorem we may conclude that functions with different subindices are pairwise orthogonal in Ω^n. The second consequence of this result is that

$$\int_{\Omega^n} f dx = f_0 + \int_{\Omega^n} \sum_{k=1}^{n} f_k(x_k) + \int_{\Omega^n} \sum_{1 \leq k < l \leq n} f_{kl}(x_k, x_l) dx + \ldots + \int_{\Omega^n} f_{1,2,\ldots,n}(x_1, \ldots, x_n) dx. = f_0.$$

[3]Since our goal is to study the behaviour of a function that is obtained by the output of some physical experiment or simulation of it we may assume that the functions of interested in this work have a compact domain, hence by some reescaling we can always assume that the domain of the function under study has as its domain the unit hypercube

So by obtaining f_0 we can use recursion to obtain the other functions in the decomposition. For example if $dx_{\sim 1}$ represent integration with respect to $x_2, x_3, \ldots, x_n$, it is not hard to see that

$$f_1(x_1) = -f_0 + \int_{[0,1]^{n-1}} f(x) dx_{\sim 1}.$$

Similar formulas hold for the other functions in the decomposition of f. By knowing explicitly all the functions in the decomposition allows us to assess the relevance of each variable in the behaviour of f by defining on them its variances. The total variance D of f is defined as

$$D = \int_{\Omega^n} f^2(x) dx - f_0^2.$$

As we can see this is the same as the usual definition of the variance of a random variable X in statistics:

$$Var(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2.$$

Where in this case the expectation operator is obtained by taking the integral over the whole space Ω^n. Similarly we compute the partial variances as

$$D_{i_1,\ldots,i_s} = \int_{[0,1]^{n-1}} f^2_{i_1,\ldots,i_s} dx_{i_1} \ldots dx_{i_s}.$$

With these variances we define the $s-th$ order sensitivity as

$$S_{i_1,\ldots,i_s} = \frac{D_{i_1,\ldots,i_s}}{D}.$$

This quantity is a measure of the effect in the output from the interaction of the variables $x_{i_1}, x_{i_2}, \ldots, x_{i_s}$. If we want to know the effect in the output given by each of the variables $x_1, \ldots, x_n$ we would have to study the first order sensitivity indices $S_1, \ldots, S_n$. Finally if we want to assess the full effect that one variable has on the output of a function f we calculate what is defined as the total effect index. If for example we want to calculate the total effect index for the variable x_1 we would do so by calculating

$$S_1 + S_{12} + S_{13} + \ldots + S_{123} + S_{124} + \ldots + S_{12\ldots n}.$$

As we can see working with GPs and doing a sensitivity analysis of a function via Sobol indices is not hard conceptually but computationally challenging. Writing a routine that interpolate test points using training points as the base using GPs or calculate sensitivity using Sobol indices accurately might be time consuming. That is why as we will see in chapter three and four we used R packages to assist us with this kind of calculations. Talking about the packages used in this work is our next topic.

2.1.3 R packages

In this work we used two different R packages. One to do the fitting with GPs (DiceKriging) and the other to do a sensitivity analysis using Sobol Indices (Sensitivity). We are going to briefly describe both of this packages.

Package: DiceKriging

According the description of the package (http://dice.emse.fr/) it is used for Estimation, validation and prediction of GP models.

The way this package works is as follows. First it creates an element of the class 'km' by receiving as an input a trending formula, the set of training points (x_i, f_i) and a choice of a covariance kernel. The kernels available are: Gauss, Exponential, Matern $\frac{3}{2}$,Matern $\frac{5}{2}$ and power exponential. It is also possible to work with tailored covariance kernels but we won't explore that possibility. Once the 'km' object is created we can do predictions on test points x^* by using the function predict. Predict takes as an input a km object, the set of test points and some other optional parameters. Gives as an output an R list that contains the estimation of $f(x^*)$ using the mean of the GP and the lower and upper 95% confidence interval using the covariance matrix (see equation (2.5)). One of the nice feature that the DiceKriging package has is that once you choose a kernel, you don't have to set the parameters of the kernel chosen. Using an optimization routine the function predict chooses the best combination of parameters through a Maximum Likelihood Optimization [4] (see 2.6).

Package: Sensitivity

The main function we used was the function SobolGP. This function takes as its main input an object from the class 'km' A matrix representing a sample

of random points in the domain of the function f we want to calculate its sensitivity (this function f was previously 'fitted' by the function km in package DiceKriging) and the main output are two lists. One lists that contains all the results for the GP-based sensitivity analysis for the main effect and one list with the results for the GP-based sensitivity analysis for the total effects.

For the next chapter we are going to explain how to use in a toy problem all of this tools explained in this chapter, so for chapter four we can focus on the results instead on how we applied what was explained here.

Chapter 3

Test Problem: How Theory Works in Practice

In the previous chapter we described an overview of the theoretical and computational tools needed obtain the solution to Bayesian inverse problems and uncertainty quantification of the solution. In this chapter we are going to work on a test problem. The idea is to illustrate how the theory can be applied in practice, so, in chapter four we can focus mainly on the results and the solution of the problem described in chapter one.

To talk about the solution of an inverse problem we first need to specify what is the forward problem. To keep things not to complicated we are going to define the forward problem as the following partial differential equation (PDE)

$$\begin{cases} \Delta u = e^{-b\|x\|_2} & \text{for } x \in \Omega = [0,1] \times [0,1] \subset \mathbb{R}^2 \\ u = 0 & \text{for } x \in \partial\Omega \end{cases} \tag{3.1}$$

where b is some real parameter. The way to think about this problem is as follows: the function u represents somei physical quantity of interest that is modeled by equation (3.1). In the language of chapter two this system is represented by the function $f(\cdot)$. But there is a caveat with this model, we don't know the value of the parameter b that best describes the reality. All we have is some (small) number of experimental measurements m of the function u within Ω. An easy way to solve the problem of finding the bet b that suits the experimental data would go as follows. If you truly believe that model (3.1) describes the behaviour of u, then build a program that simulates that system for a big range of parameters b and then pick the results that

are closer to the experimental data. Problem solved? Not quite, even for this simple system running too many simulations for different parameters b is computationally expensive and time consuming. Let alone more realistic physical models with more complicated set of equations describing them.
A more feasible solution is given by the theory developed in chapter two. Run a few times a program that simulates model (3.1), then construct a computationally cheap[1] emulator $\hat{f}(\cdot)$ to fill the missing data on possible values for b. Then with the aid of Bayes formula, find the posterior for b given the experimental data m, i.e.

$$P(b|m) \propto P(m|b)P(b). \tag{3.2}$$

With this posterior find a possible value of b (e.g mean of the posterior) and quantify the uncertainty with respect to that prediction (e.g. 95% Bayesian confidence interval). Now that we have the recipe we can put hands on in the calculations.

To generate syntetic data of model (3.1) we proceed as follows. We assume that the true value of b is 0.925 then run a Matlab script that simulates the PDE using the five point extensil for the laplacian in a finite difference scheme [16]. The solution with $b = 0.925$ is shown in Figure 3.1. To simulate the experimental measures m, we chose 10 random points in the domain and evalute the numerical solution there. To each of the measurements we added the noise of a random variable normally distributed with mean 0 and $\sigma = 0.01$. With this we get a signal to noise ration (SNR) of about 1:3. In real life measurements are more precise than this, however in this test problem we are not taking into account the inaccuracies from the model itself to describe the phisical reality. To compensate for that we chose a bigger SNR.
With the synthetic data at hand we can tackle the question of what value of b fits the data the most. In principle $b \in \mathbb{R}$, so virtually b could be any real value. In real life most of the time the physics of the problem dictates the allowed values of b and experience allows to bound what the real value of b could be. In our case we pretend that the physics and experience tell that b is a positive number that could be anywhere between zero and two, that is we strongly believe that $b \in (0, 2]$.

[1]Cheap relative to the computational cost of actually simulate the physical system through equation (3.1)

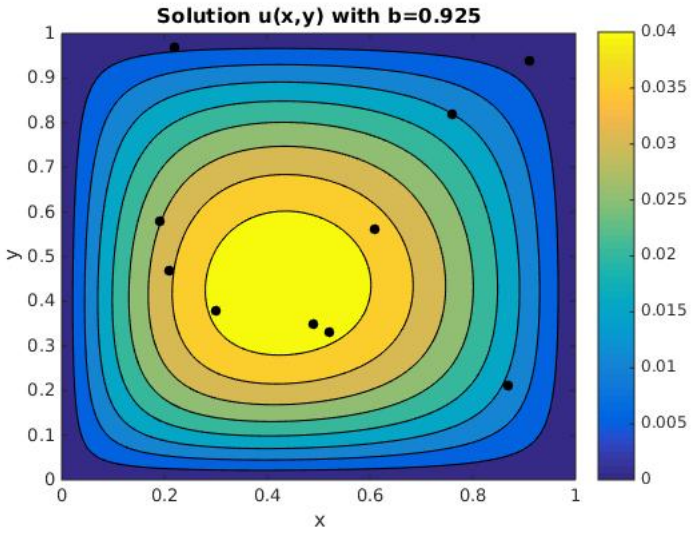

Figure 3.1: Numerical solution of the system (3.1) using a finite difference scheme. The mesh size used in x and y was 0.01. The value of the parameter b was set at 0.925. The black dots in the plot represent the points used to generate the synthetic data about the experimental measures m.

At this point we use the concept of emulator as in the previous chapter. If we had unlimited computational power or time then we can run the finite difference scheme used to obtain Figure 3.1 with a huge set of possible values for b in the interval $(0, 2]$. And pick the value that replicates the better the syntetic data m. Since we do not have unlimited computational power or time what we are going to do is to pick some n values $b_k \in (0, 2], k = 1, 2, \ldots, n$, run the finite difference scheme on those and then use an emulator $\hat{f}$ with GPs to fill the missing data. The criteria to choose the number n comes from considerations of how much time and computational resources we have. For this case we chose $n = 10$. How to distribute those 10 points in the interval $(0, 2]$. Here we use the concept space filling design. In the one dimensional setting it is not hard to see that a maximin design gives an equal spacing of the 10 points in the interval $(0, 2]$ hence we choose

$$b_k = 0.2k \qquad \text{for } k = 1, 2, \ldots, 10.$$

We are ready to run the emulator $\hat{f}$ for these values of b, get a numerical value at the 10 points in the domain where the synthetic data were created

(black dots in Figure 3.1) and use GPs to fill the missing information. Take into account that this process of 'filling the blanks' has to be done in all of the 10 measurement points in the domain Ω of definition of the phyiscal model.

In Figure 3.2 we can see the results from running the emulator (black dots) compared with the experimental measurement for each of the 10 sites (black line).

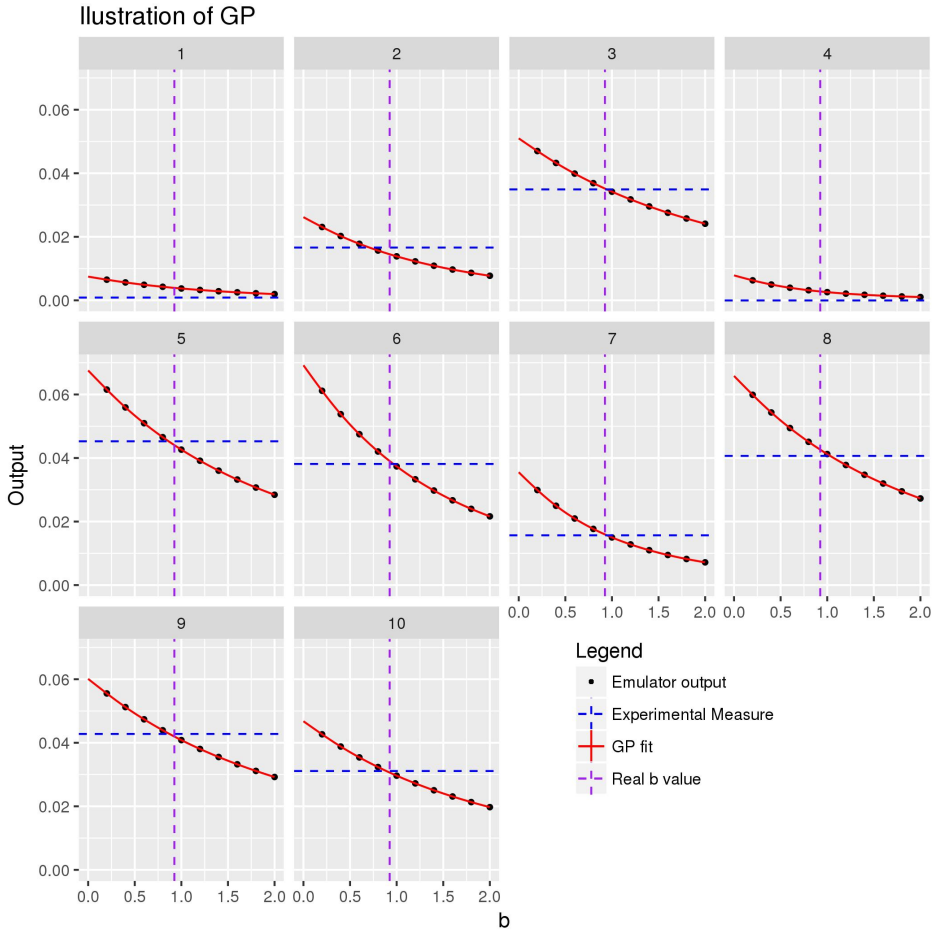

Figure 3.2: whatever

As we can see in Figure 3.2, the emulator sometimes overestimates the value of b and sometimes underestimates it, this behaviour is expected. The hope is that this inaccuracies oscilate around the true value of b.

Since we only have a limited number of prediction of the emulator for different values of b (black points in Figure 3.2), we would like to extrapolate/interpolate those results to get more data and with that extra data, assess the value of b. As explained in Chapter 2, one very useful way to obtain the extra data is through Gaussian processes. Using the language of the previous chapter what we want to do is, having the training points $\{(x_i, f(x_i)\}_{i=1}^{10}$ we want to predict the value of different test points x_i^*. This prediction is shown as a red line in Figure 3.2. The GP fit allows us to approxiamte as much value as we want, along with the uncertainty associated with the prediction. Having this data we can now proceed to use the Bayesian methodology. The idea goes as follows. if we denote the GP fit at at point b as $G(b)$, then we may assume an additive noise model for the output y as

$$y = G(b) + \vec{\epsilon}. \tag{3.3}$$

$\vec{\epsilon}$ is a random vector distributed as $\vec{\epsilon} \sim \mathcal{N}(0, \sigma I)$. Where $sigma = bla$ and I is the 10×10 identity matrix. In the Bayesian framework we are interest in finding the value of b given the experimental measurements m. To that end we go back to the begining of this chapter and use equation (3.2). To use this equation we need to find the likelihood $\mathbb{P}_{like}(m|b)$. Under the assumption of the additive noise model in equation (3.3) we get as in the smashed window example in chapter 2 that

$$m|b \sim \mathcal{N}(G(b), \sigma I),$$

more precisely

$$\mathbb{P}_{like}(m|b) = \frac{1}{(2\pi\sigma)^{n/2}} \exp\left(-\frac{\|G(b) - m\|_2^2}{2\sigma^2}\right).$$

Now we need to choose a prior for b. This is a delicate issue and a polemic one in the Statistics community. The prior should reflect all of our current knowledge about b. However your knowledge about b might be different than my knowledge about b, hence your prior for b might look different than mine. For the moment we won't worry about that. Let's assume that our prior for b is given by

$$b \sim U(0, 2).$$

where $U(a, b)$ is the uniform distribution in a, b. Putting all of this into formula (3.2) we get

$$\mathbb{P}_{post}(b|m) \propto \chi_{[0,2]} \exp\left(-\frac{\|G(b) - m\|_2^2}{2\sigma^2}\right),$$

where $\chi_{[a,b]}$ is the indicator function of the set $[a, b]$. This equation can be interpreted as the update in knowledge from to prior to the posterior in the light of the experimental data obtained represented by the likelihood. This change from prior to posterior is shown below.

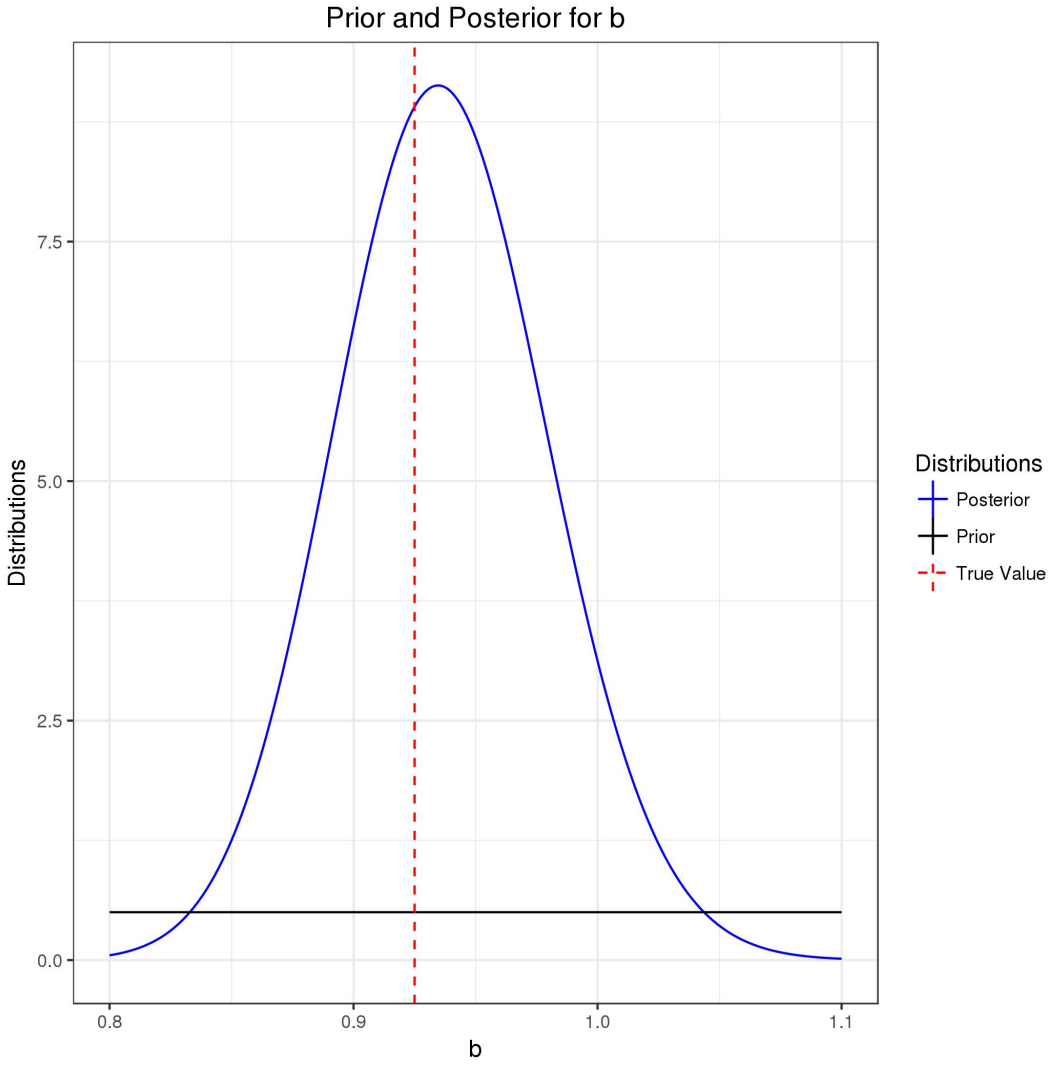

Figure 3.3: whatever

As we mentioned in chapter 2, the solution to an inverse problem in the Bayesian framework is the posterior. The question is: how can we use this

posterior to infer the value of b and the uncertainty associated to it? **Here talk about how this integral is untractable**.

To make inferences about the value of b we resort to a family of techniques from sampling probability distributions known as Markov Chain Monte Carlo (MCMC)**Cite algun libro del MCMC**.

After using the MCMC we get the following plot

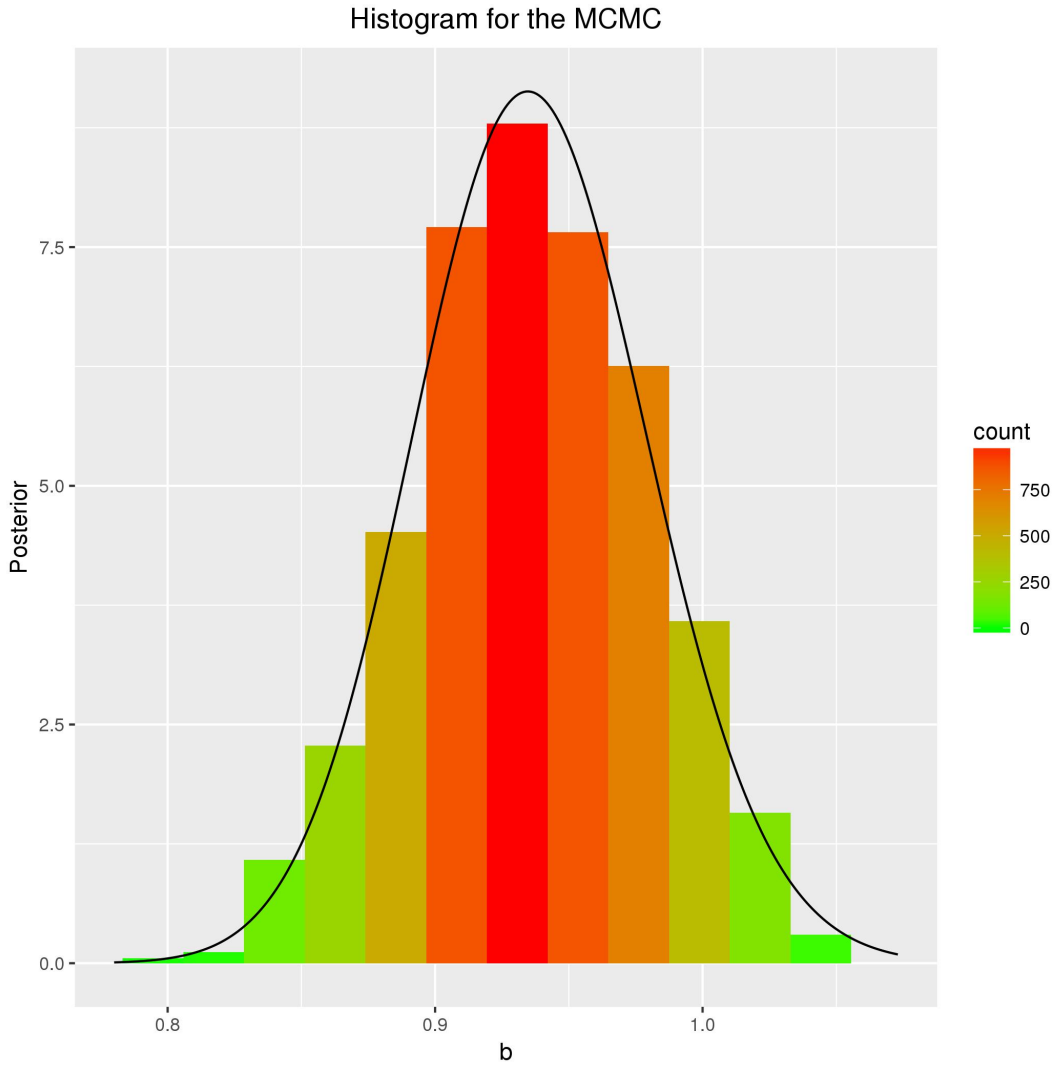

Figure 3.4: whatever

How important is the prior to make inferences?

It is constructive to go in some depth and explore how prominent is the role of the prior probability in making inferences about a problem. Let us consider the same problem as above, we want to estimate the value of the paramter b. For demonstration purposes, let us assume two things. The parameter b can be any real number (not just $0 < b \leq 2$ as before) and

$$b \sim \mathscr{N}(b^*, \sigma_b^2),$$

where b^* and σ_b are parameters to be set later. With this new prior the formula for the posterior becomes

$$\mathbb{P}_{post}(b|m) \propto \exp\left(-\frac{\|m - G(b)\|_2^2}{2\sigma^2} - \frac{(b-b^*)^2}{2\sigma_b^2}\right).$$

As before, assume that the true value of b is 0.925. To ilustrate the role that the prior has in the inference of the value of b given the experimental data m, let us assume that

$$b \sim \mathscr{N}(4, 2.5).$$

That is, the person that proposed this prior is confident that within 95% of confidence, the true value of b is in the interval $[1.8, 8.2]$. Let us see what happens with the posterior in equation (3.2) when we get more and more experimental data.

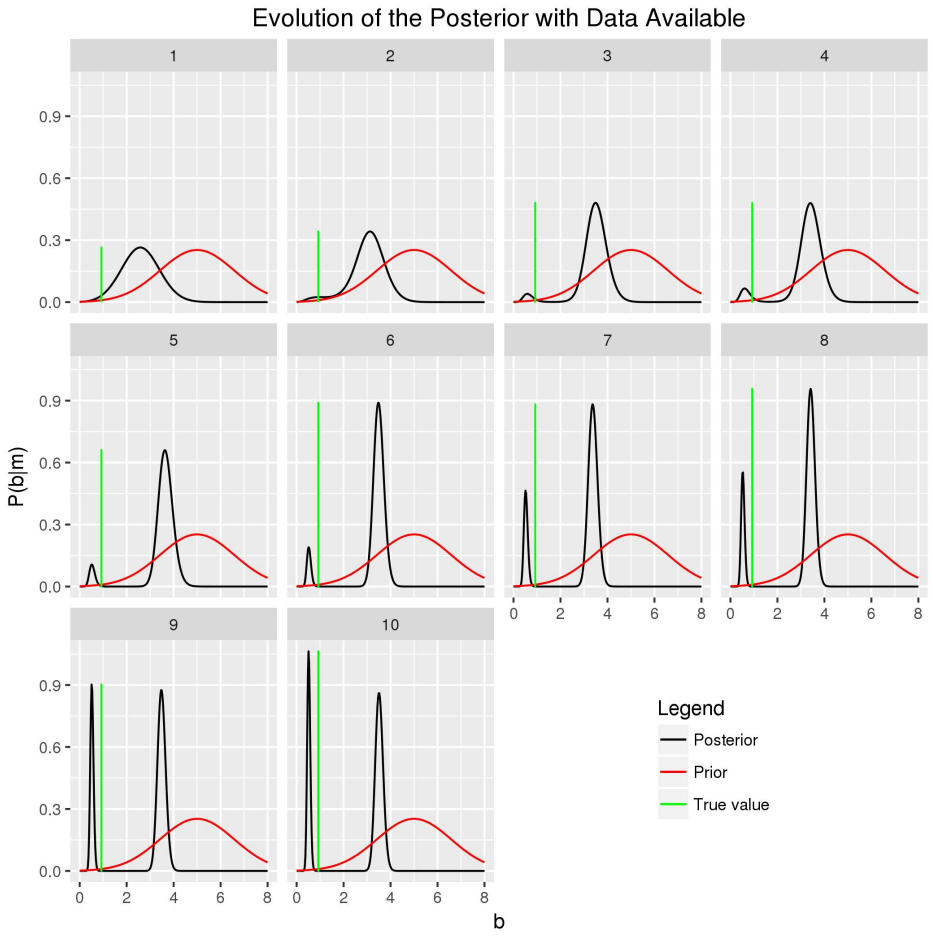

Figure 3.5: bla

Bibliography

[1] Vladimir Igorevich Arnol'd. *Mathematical methods of classical mechanics*, volume 60. Springer Science & Business Media, 2013.

[2] Alberto Bressan. *Lecture Notes on Functional Analysis*. American Mathematical Society, 1900.

[3] Richard M Dudley. *Real analysis and probability*, volume 74. Cambridge University Press, 2002.

[4] Delphine Dupuy, Céline Helbert, Jessica Franco, et al. Dicedesign and diceeval: Two r packages for design and analysis of computer experiments. *Journal of Statistical Software*, 65(11):1–38, 2015.

[5] Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

[6] Mark E Johnson, Leslie M Moore, and Donald Ylvisaker. Minimax and maximin distance designs. *Journal of statistical planning and inference*, 26(2):131–148, 1990.

[7] Marc C Kennedy and Anthony O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464, 2001.

[8] Leonid P Lebedev, Iosif I Vorovich, and Graham Maurice Leslie Gladwell. *Functional analysis: applications in mechanics and inverse problems*, volume 41. Springer Science & Business Media, 2012.

[9] Mikhail Lifshits. *Lectures on Gaussian processes*. Springer, 2012.

[10] Mikhail Anatolevich Lifshits. *Gaussian random functions*, volume 322. Springer Science & Business Media, 2013.

[11] Anthony OHagan. Bayesian analysis of computer code outputs: a tutorial. *Reliability Engineering & System Safety*, 91(10):1290–1300, 2006.

[12] Luc Pronzato and Werner G Müller. Design of computer experiments: space filling and beyond. *Statistics and Computing*, 22(3):681–701, 2012.

[13] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[14] Andrea Saltelli, Karen Chan, E Marian Scott, et al. *Sensitivity analysis*, volume 1. Wiley New York, 2000.

[15] Somersalo and Kaipio. *Statistical and computational inverse problems*. Springer-Verlag, 2005.

[16] James William Thomas. *Numerical partial differential equations: finite difference methods*, volume 22. Springer Science & Business Media, 2013.