

BASS as a Surrogate Model for Bayesian Optimization

Adrian TJ

September 2022

Contents

1	Stochastic Processes	3
1.1	General Definitions	3
1.2	Markov Chains	5
1.3	Gaussian Processes	10
1.4	Gaussian Process Regression	13
1.4.1	Preamble to Gaussian Process Regression: A Rehash on Regression Analysis	15
1.4.2	Theory of Gaussian Process Regression (from the func- tion space view)	16
2	Bayesian Optimization	19
2.1	Introduction	19
2.2	Components of BO	21
2.3	Formalization	22
3	BASS and their Origins	25
3.1	Decision Trees	26
3.2	MARS Models	28
3.3	BMARS Models	31
3.4	BASS Models	33

There is not a single theorem, lemma, or proposition in the whole paper. Have my senses taken leave of me? What, no asymptotics or results concerning the rate at which MARS approaches the “True Model” as the sample size goes to infinity? For one of the few times in its history, the Annals of Statistics has published an article based only on the fact that this may be a useful methodology. All is ad hoc; there is no maximum likelihood, no minimax, no rates of convergence, no distributional or function theory. Is nothing sacred? What kind of statistical science is this? My thanks go to the editor and the others involved in this sacrilegious departure.

- Leo Breiman, discussing the MARS model.

Chapter 1

Stochastic Processes

Before starting, we would like to note that sections of this chapter were constructed using different references as primary guides. The section on Markov chains primarily uses the references [Ži10] and [HR05].

1.1 General Definitions

Stochastic processes play a very important role in basically every part of this work. First, the basis of traditional Bayesian Optimization is Gaussian processes, which are a specific class of stochastic process in which all the points or vectors have a conjoined multi-normal distribution. This is not the only place where stochastic processes play a role, as BASS is a Bayesian method, and modern Bayesian methods generally rely on simulation strategies such as Markov Chain Monte Carlo (MCMC) to numerically approximate complex posterior distributions. We use a particular class of MCMC algorithms called reverse jump MCMC when dealing with our alternate surrogate function, but that is detailed much later when we talk about the Bayesian Adaptive Regression Spline method.

All in all, stochastic processes are generally important to the content of this work so we include some important results and definitions, starting with the definition of a stochastic process itself, taken from [Ži10, §3.0].

Definition 1. *Let $\Theta \subseteq \mathbb{R}^+$. A stochastic process $\{X_\theta, \theta \in \Theta\}$ is a collection of random variables, indexed by a parameter θ , such that θ belongs to some index set Θ . When $\Theta = \mathbb{N}$ (or $\Theta = \mathbb{N}_0$), then it is called a discrete-time process, and if $\Theta = \mathbb{R}^+$, then it is called a continuous-time process.*

To distinguish between a discrete and continuous time process, we use the index n for discrete-time processes and t for continuous-time processes. In

all applications of this work the indexes refer to time, as the function spaces \mathcal{X} on which the functions are evaluated are time dependent. This definition is very broad, as for example, if we take the case where $\Theta = \{1\}$, then the stochastic process $\{X_\theta, \theta \in \Theta\}$ is really just X_1 , a random variable. As with many other areas of mathematics though, the introduction of infinities in the values the indexes can take introduces a large amount of complexity, and things like vector distributions do not extend nicely to these new cases.

Another more formal definition of a stochastic process defines it as a function of the index and an element of the sample space, as follows:

Definition 2. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space¹ where Ω is a sample space, \mathcal{F} is a σ -algebra and $\mathbb{P} : \mathcal{F} \mapsto [0, 1]$ a probability measure. Taking Θ an arbitrary index space, a stochastic process is a function

$$X : \Omega \times \Theta \mapsto \mathbb{R}$$

such that for all $\theta \in \Theta$,

$$X_\theta : \omega \mapsto X(\omega, \theta) \equiv X_\theta : \Omega \mapsto \mathbb{R}$$

is a measurable function, which is equivalent to saying that X_θ is a random variable.

When ω is known, $\theta \mapsto X(\omega, \theta) : \Theta \mapsto \mathbb{R}$ is a sample function, also known as a realization of the stochastic process. These realizations are the trajectories or paths that are famous in diagrams such as the one shown in Figure 1.2. An example application of this type of structure are time series, where the fundamental properties of stochastic processes are used to generate a function whose realizations move through time.



As was mentioned earlier, the two main types of stochastic process that can be defined are ones where the realizations of the process occur in a

¹Billingsley's classic book [Bil12] has all the theoretical foundation of probability spaces and provides a thorough run down of these concepts.

discrete fashion, and the other is when they occur in a continuous fashion. It so happens that the two main structures relating to stochastic processes that we need to go into detail on are Markov chains, a discrete process, and Gaussian processes, which are continuous. We go in depth in the following sections into each of these types of stochastic processes, starting with Markov chains since Gaussian processes lead nicely onto the next chapter.

1.2 Markov Chains

Markov chains are ubiquitous in their applications, since they represent a relatively simple but very powerful concept; the mathematical structure on which we can define phase transitions of position changes that are governed by probabilistic rules. The fundamental property which will be explored later is that independently of the past, the only important factor in determining the future state of the process is the current state of it. We begin with a formal definition and take a constructive approach in building this model.

Definition 3. *Let $\{X_n, n \in \mathbb{N}_0\}$ be a stochastic process defined on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ such that*

$$X_n : \omega \mapsto X(\omega, n) \equiv X_n : \Omega \mapsto \mathcal{S},$$

where \mathcal{S} is a finite state space of cardinality k , such that $\mathcal{S} = \{s_1, s_2, \dots, s_k\}$. We call $\{X_n, n \in \mathbb{N}_0\}$ a Markov chain if

$$\mathbb{P}(X_n = s_n | X_{n-1} = s_{n-1}, X_{n-2} = s_{n-2}, \dots, X_0 = s_0) = \mathbb{P}(X_n = s_n | X_{n-1} = s_{n-1})$$

for all $n \in \mathbb{N}_0$ and all $s_0, s_1, \dots, s_n \in \mathcal{S}$ whenever the two conditional probabilities are well defined, this is, when $\mathbb{P}(X_n = s_n, X_{n-1} = s_{n-1}, \dots, X_0 = s_0) > 0$.

Intuitively, this means that the next state in the chain is only dependent on the current state of the chain, and that the history of how the chain got to the current state is irrelevant to the next step. This behaviour is known as the Markov property, and one very valuable insight that emerges from this behaviour is being able to completely determine all properties of the chain given only the initial state of it and the knowledge of how the chain behaves at all of its possible states.

The initial state or initial probability is self-explanatory, it tells us the expected behaviour of the first element in the chain, X_0 . Formally, we call $\mu^{(0)} = \{\mu_s^{(0)} : s \in \mathcal{S}\}$ with $\mu_s^{(0)} = \mathbb{P}(X_0 = s)$ the initial probability distribution of the process.

The other important component to fully describe a Markov chain is the transition probabilities from one state s_i to another s_j for all $s_i, s_j \in \mathcal{S}$. We identify this probability as

$$p_{i,j} = \mathbb{P}(X_n = s_j | X_{n-1} = s_i).$$

If we take all combinations of the states the chain can be in and map them to all other states, we get what is known as the transition matrix P , where each entry in the matrix denotes the probability of moving from one specific state to another. This matrix, because it is fundamentally a matrix made up of probabilities, has two interesting properties for the purposes of this work:

- $p_{i,j} \geq 0$ for all $i, j \in \{1, 2, \dots, n\}$
- $\sum_{i=1}^n p_{i,j} = 1$ for all $j \in \{1, 2, \dots, n\}$.

We note that these two properties are the definition of a stochastic matrix, and therefore, by construction, Markov chain transition matrices are stochastic matrices. We have only described that we can fully determine the Markov chain from its initial state and its transition probabilities, but we have not shown this concept concretely. The following theorem ties these loose ends together and formally describes this property.

Theorem 1. *Let $\{X_n, n \in \mathbb{N}_0\} : \Omega \mapsto \mathcal{S}$ be a stochastic process defined on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with \mathcal{S} a finite state space. Then, $\{X_n, n \in \mathbb{N}_0\}$ is a Markov chain if and only if there exists a stochastic matrix P such that*

$$\mathbb{P}(X_n = s_n | X_{n-1} = s_{n-1}, \dots, X_0 = s_0) = p_{n-1,n}$$

for all $n \in \mathbb{N}_0$ and all $s_0, s_1, \dots, s_n \in \mathcal{S}$ such that $\mathbb{P}(X_{n-1} = s_{n-1}, \dots, X_0 = s_0) > 0$.

This theorem is of great importance in the area of study of Markov chains, and we refer the interested reader to [Chu60, §1.2] for a detailed proof. This theorem establishes a bridge between the theory of Markov processes in general and the properties of linear algebra and matrix theory. One such interesting property is the use of eigenvectors and eigenvalues of the transition matrix used to define a stationary distribution, which is what we are ultimately building towards this section.

We have seen that the transition matrix P governs the behaviour and the changes that happen from one state of the Markov chain to the next. A natural follow up question arises from trying to understand the behaviour of the chain an arbitrary $k > 1$ steps in the future instead of a single step,

such as if we wanted to calculate $\mathbb{P}(X_{i+k} = s_{i+k} | X_i = s_i)$. We first note that this multi-step transition probability can be simplified, assuming that the Markov chain is homogenous, to the following:

$$\mathbb{P}(X_{i+k} = s_{i+k} | X_i = s_i) = \mathbb{P}(X_k = s_k | X_0 = s_0)$$

which we denote as $p_{i,k}^{(k)}$. Because we can define the individual probabilities of transitioning from one step to another after k steps, it stands to reason that there would be a transition matrix to describe this behaviour. The theorem that establishes the way these relationships work is the Chapman-Kolmogorov theorem and equation, which we present now. The formulation of this theorem presented in this work is taken from [Ži10, §8.3].

Theorem 2 (Chapman-Kolmogorov Theorem). *Let $n, m \in \mathbb{N}_0$ and $s_i, s_j \in \mathcal{S}$. Then, $p_{i,j}^{(n+m)}$, which is the probability that starting from state i at step n we end up in state j after m steps is given by:*

$$p_{i,j}^{(n+m)} = \sum_{k \in \mathcal{S}} p_{i,k}^{(n)} p_{k,j}^{(m)}.$$

Proof. We will prove the case when the Markov process is homogenous. We begin by noting that $p_{i,j}^{(n+m)} = \mathbb{P}(X_{n+m} = s_j | X_n = s_i) = \mathbb{P}(X_m = s_j | X_0 = s_i) = p_{i,j}^{(m)}$. Applying the law of total probability and using the Markov property,

$$\begin{aligned} p_{i,j}^{(n+m)} &= \\ &= \mathbb{P}(X_{n+m} = s_j | X_n = s_i) \\ &= \mathbb{P}(X_m = s_j | X_0 = s_i) \\ &= \sum_{k \in \mathcal{S}} \frac{\mathbb{P}(X_m = s_j, X_n = s_k, X_0 = s_i)}{\mathbb{P}(X_0 = s_i)} \\ &= \sum_{k \in \mathcal{S}} \mathbb{P}(X_m = s_j | X_n = s_k) \mathbb{P}(X_n = s_k | X_0 = s_i) \\ &= \sum_{k \in \mathcal{S}} p_{i,k}^{(n)} p_{k,j}^{(m-n)}. \end{aligned}$$

□

We now introduce the following lemma that takes advantage of this property to provide an alternative way of computing and understanding the transition matrix P .

Lemma 1. *Let P^k be the k -th order matrix power of the transition matrix P for a homogeneous Markov chain $\{X_n, n \in \mathbb{N}_0\}$. Then,*

$$p_{i,j}^{(k)} = (P^k)_{i,j} \text{ for } i, j \in \mathcal{S}.$$

There is a way to prove this lemma by induction directly, but we opt instead to use the Chapman-Kolmogorov Equation $n-1$ times to achieve the final result, as follows.

This lemma becomes useful in proving the following theorem.

Theorem 3. *Let $n, m \in \mathbb{N}_0$ and $i, j \in \mathcal{S}$. Then, $p_{i,j}^{(n+m)}$, which is the probability that starting from state i in step n we end up in state j after m steps is given by:*

$$p_{i,j}^{(n+m)} = \sum_{k \in \mathcal{S}} p_{i,k}^{(m)} \cdot p_{k,j}^{(n)}.$$

From here, we now present two definitions relating to the properties of matrices which will become important when further exploring the types of Markov chains we are interested in.

Definition 4. *A matrix $A \in \mathbb{R}^{n \times m}$ is called non-negative if $a_{i,j} \geq 0$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$.*

Before continuing, we note that with stochastic matrices being just glorified assortments of probabilities that are neatly ordered, they are always non-negative.

Definition 5. *Let A be a non-negative matrix. If there exists $N_0 \geq 1$ such that all elements of A^{n_0} are positive, then we call A a quasi-positive matrix.*

These two definitions are necessary building blocks to get to the property we are truly interested in, ergodicity.

Definition 6. *A Markov chain $\{X_n : n \in \mathbb{N}_0\}$ is called ergodic if the limit $\pi_j := \lim_{n \rightarrow \infty} p_{i,j}^{(n)}$*

- *Exists for all $j \in \{1, 2, \dots, k\}$.*
- *Is positive and does not depend on i .*
- *$\pi = (\pi_1, \pi_2, \dots, \pi_k)$ is a probability distribution on \mathcal{S} .*

Ergodicity is an incredibly powerful property that is not native to stochastic processes per say, but rather it is inherited from measure theory. In a sense, for a system to be ergodic it needs to be governed by constant or unchanging probability laws, irregardless of a change in state, or in the context of time-centric processes, changes in time. As a simple example, we consider throwing a fair coin in the air 100 times. We would (on average) get the same result if 100 people throw one coin compared to one person throwing the coin 100 times. This would be an ergodic system. On the other hand, if we change the coin to one that is imbalanced or unfair after 50 throws are completed, then the laws governing the change themselves would not remain constant as time passes or we change states, and this would be an example of a non-ergodic system.

The scope of this work does not cover this theory in as much detail as the author would like, but the interested reader is directed to sections 24 and 36 of [Bil12] for a much more complete rundown of Ergodicity looked at through the lenses of measure theory and stochastic processes. The result ergodicity provides that is the foundation for MCMC applications is the following theorem.

Theorem 4. *Let $\{X_n : n \in \mathbb{N}_0\}$ be an ergodic Markov chain, such that*

$$\pi_j = \lim_{n \rightarrow \infty} p_{i,j}^{(n)}, \text{ for all } j \in \mathcal{S}.$$

Then, the vector π is a unique solution of

$$\pi^T = \pi^T P$$

further, π is a probability distribution on \mathcal{S} .

Proof. Let j be an arbitrary member of \mathcal{S} . Then,

$$\begin{aligned} \pi_j &= \lim_{n \rightarrow \infty} p_{i,j}^{(n)} \\ &= \lim_{n \rightarrow \infty} p_{i,j}^{(n+1)} \\ &= \lim_{n \rightarrow \infty} p_{i,j}^{(n)} \cdot p_{i,j} \\ &= \pi_j \cdot p_{i,j}. \end{aligned}$$

Since this holds true for all $j \in \mathcal{S}$, then in particular it holds true for the vectorized version of this equation, this being

$$\begin{bmatrix} \pi_1 \\ \pi_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} \pi_1 & \pi_2 & \dots \end{bmatrix} \begin{bmatrix} p_{i,1} \\ p_{i,2} \\ \vdots \end{bmatrix}$$

Which is equivalent to saying that $\pi^T = \pi^T P$. □

If we construct a Markov chain in a way that it is ergodic and control the distribution π so that it is something that we want, we can guarantee that after a sufficiently large amount of iterations n we will be sampling from the stationary distribution of the process, which is invariant through time because of the property presented in the above theorem. This is the crux of understanding the Markov Chain Monte Carlo technique which is the central object of study of this section. Delving into the construction of the Metropolis-Hastings algorithm and the fundamental building blocks needed to construct these chains that eventually lead to interesting stationary distributions would make this work exceedingly long, but [RK16] and [RCC99] are good resources for a detailed rundown of these topics. We will return much later in this work when discussing the BASS method to MCMC and why it is so essential to this process. We will not look at pure MCMC though but an interesting modification to the base algorithm known as the Reversible Jump Markov Chain Monte Carlo (RJMCMC) algorithm.

For now though we move on to the other interesting sub-case of stochastic processes relevant to this work, the continuous-time Gaussian Process.

1.3 Gaussian Processes

A Gaussian process can be thought of as a generalization of the Gaussian (normal) distribution when viewed from the lens of stochastic processes. Stochastic processes, as we saw before, are indexed by a parameter, which itself belongs to an index set. The Gaussian process extends the definition of the Gaussian distribution from being valued on scalars or vectors to a more general index set, to a set $T \subseteq \mathbb{R}^+$. Note that this set can contain an infinite number of points, which is the main difference between the Gaussian distribution and the Gaussian process. This extension is not trivial though, and to specify the structure of these particular types of stochastic processes, we aid ourselves with finite dimensional distributions. We now provide a formal definition of these types of distributions.

Definition 7. *Let $\{t_i\}_{i \leq k}$ be a sequence of points² such that $\{t_i\}_{i \leq k} \subseteq T$. The k -dimensional random vector $(X_{t_1}, X_{t_2}, \dots, X_{t_k})$ has a distribution $\mu_{t_1, t_2, \dots, t_k}$ over \mathbb{R}^k . These measures $\mu_{t_1, t_2, \dots, t_k}$ are the finite dimensional distributions of the process.*

One logical follow up is to ask if the inverse of this definition is also true.

²Sometimes referred to as time points, considering this index not as an abstract set but a representation of the progress of time as indexed by a subset of the positive reals.

This would be, starting from a consistent³ joint distribution of the random variables $(X_{t_1}, X_{t_2}, \dots, X_{t_k})$, with $k \in \mathbb{N}$, then would it be possible to define a probability measure $\mu_{t_1, t_2, \dots, t_k}$ on a higher cardinality set, in this case \mathbb{R}^k that matches these distributions exactly? The positive answer to this question is the Kolmogorov Extension Theorem, an incredibly deep and powerful result that guarantees the existence of stochastic processes by only defining a finite set of distributions that share some consistency requirements.

Theorem 5 (Kolmogorov Existence Theorem). *Given a collection of random variables $\{X_{t_1}, X_{t_2}, \dots, X_{t_k}\}$, indexed by $t_1, t_2, \dots, t_k \in \mathbb{R}$, such that for any finite subset of time points $t_{i_1}, t_{i_2}, \dots, t_{i_m}$, the joint distribution of $(X_{t_{i_1}}, X_{t_{i_2}}, \dots, X_{t_{i_m}})$ is consistent, there exists a stochastic process $\{X_t\}_{t \in \mathbb{R}}$ on a higher-dimensional space \mathbb{R}^k such that for all $t_1, t_2, \dots, t_k \in \mathbb{R}$, the marginal distributions of $(X_{t_1}, X_{t_2}, \dots, X_{t_k})$ coincide with the original given distributions.*

The Kolmogorov Existence Theorem addresses the question of whether we can construct a stochastic process given only a finite set of joint distributions. These joint distributions need to satisfy a consistency condition, which means that any finite subset of random variables should have joint distributions that are compatible with each other.

The theorem states that if such consistent joint distributions exist, then it is possible to define a stochastic process on a higher-dimensional space, often denoted as \mathbb{R}^k , such that the original given distributions match the marginal distributions of this process. In essence, the Kolmogorov Existence Theorem provides a way to create a stochastic process that captures the dependencies and correlations between random variables at different time points, even when we only have limited information about their joint distributions.

In the case of Gaussian processes, this means that if we have a finite set of random variables following consistent joint Gaussian distributions, we can construct a Gaussian process on a continuous domain (typically a subset of \mathbb{R}) such that any finite subset of the process is jointly Gaussian with the desired covariance structure.

With this auxiliary distribution out of the way, we can now proceed to define the Gaussian process formally.

Definition 8. *Let $\{X_t : t \in \mathbb{R}^+\}$ be a continuous-time stochastic process. We call $\{X_t : t \in \mathbb{R}^+\}$ a Gaussian process if for every set $\{t_i\}_{i \leq k}$ and every finite k , the finite-dimensional distribution of the k -dimensional random vector*

³Consistency is a fuzzy term with a lot of meanings, but its meaning in this context will become clear with the theorems that follow.

$(X_{t_1}, X_{t_2}, \dots, X_{t_k})$ has a Gaussian multivariate distribution when $k > 1$ and a Gaussian univariate distribution when $k = 1$.

An important extension of this is that we can consider the Gaussian process a function f and evaluate it at certain points x , where $x \in \mathcal{X}^\Theta$, the input space. This is, for each x , there is a random variable $f(x)$ that represents the possible output of the function f at this location, or a realization of a certain Gaussian distribution at location x . This can also be understood as taking a sample from the distribution $f(x)$. As with any version of a Gaussian distribution, $f(x)$ can be completely be defined by it's mean and variance. Unlike the finite-dimensional cases though, by definition we can have infinitely many values in the index space T , and therefore the reasonable approach to defining the mean and variance of functions of elements in the index space. We denote the mean function of the process as:

$$m(x) = \mathbb{E}[f(x)]$$

and the covariance function as:

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

The main reason why the covariance function is denoted by k is that this name alludes to the fact that k is a kernel of the terms in the input space. We now present a formal definition of this type of function, followed by some intuition on its interpretation. With these definitions though, we write the distribution function for f as

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

Definition 9. Let \mathcal{X} be an input space like the ones we have previously defined (in the case of Gaussian processes, $\mathcal{X} = \{X_t : t \in T \subseteq \mathbb{R}^+\}$), and let V be a vector space equipped with an inner product. Also, let φ be a mapping such that $\varphi : \mathcal{X} \times \mathcal{X} \rightarrow V$. We call the function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ a kernel or kernel function when $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_V$.

What this kernel function allows us to do is basically borrow an inner product from a different space, in this definition V , and apply it to the transformed elements of our input or feature space. This allows us to cheat in some regard and compute dot products or metrics in the input space without knowing or needing to know the properties of that space. Tying back the kernel to Gaussian processes, we note that while the mean only controls the average value across realizations of the process f , the covariance or kernel function actually controls the underlying behavior of the realizations



or Gaussian processes. As an example of this, we borrow the following figure from [Gha11].

Basically, all major properties such as smoothness, differentiability, sparsity and range (to name only a few) are controlled by the choice of the kernel function.

While the kernel function can be basically anything that fulfils the above definition, there are some particularly useful or popular kernel functions when dealing with Gaussian processes. Chief among them is the squared exponential kernel, also known as the radial basis function. This kernel is defined as

$$k(x, x') = \sigma_f^2 \exp\left(\frac{-\ell(x - x')}{2}\right),$$

where σ_f^2 and ℓ are hyperparameters. While we do not go into too much detail as to why this is a good choice for a kernel, we refer the interested reader to [Gar23, §2.1].

The hyperparameters defined previously each control the general behaviour of the process when fitting data points. A more detailed run down of these points is given in the next section, which talks about Gaussian process regression. This is the method by which uncertainty is measured and function approximations are conducted in the traditional approach to Bayesian optimization.

1.4 Gaussian Process Regression

One of the applications of Gaussian processes is to use them as a method to perform supervised learning using it. The method generates from data a mean function and then uses that mean function to extrapolate or predict on points where there is no data observed. As we described in the last chapter, we can fully specify or determine a particular Gaussian process by specifying its mean $m(x) = \mathbb{E}[f(x)]$ and its kernel function $k(x, x')$. While in the last chapter we started with the Gaussian process and from that generate the

values on the rest of the input space, in this chapter we are going to be doing somewhat of the inverse of that. We will be starting from a set of data $\{x_i\}_{i \leq n} = \mathcal{D} \subset \mathcal{X}$, and adjusting the properties of the kernel and mean to achieve the best fit Gaussian process to the points and minimize the error.

One of the advantages of Gaussian process regression over other forms of supervised learning algorithms (such as linear regression) is the fact that Gaussian processes are a non-parametric model. Without delving too much into the specifics, this means that the number of parameters controlling the model is not fixed at a certain quantity, but rather is adaptive. The amount of parameters can grow and shrink depending on the data that is being fit.

This however does not mean that we do not have control over the behavior of the function we decide to adjust to the points. In the following figure, we can see how changing the hyperparameter ℓ in the squared exponential covariance function changes the relative smoothness of the fit function.



Figure 1.1: Gaussian Processes for ML, page 20.

We also note how radically the confidence intervals change from one adjustment to the next based on moving this hyperparameter around. Because we can define an infinite amount of Gaussian processes based on our choice of hyperparameters (and further even by changing the kernel function), **HOW TO STATE THIS AS A MINIMIZATION PROBLEM?** This problem translates to finding under the given assumptions the function that best adjusts to these points in the sense that it minimizes the uncertainty present. This will involve the selection of the best⁴ hyperparameters given the constraints of the problem and using that defined mean function to predict values not in the data set.

To get there, we will begin by going over regression in general, then make

⁴Best is a dangerous word in this context. Obviously depending on the problem and the minimization objectives, this best set of hyperparameters that define this function can change, but here we simply refer to the one or family of functions that solves the above minimization problem.

the link to the formal theory of Gaussian process regression to show how supervised learning can be performed by this method.

1.4.1 Preamble to Gaussian Process Regression: A Refresh on Regression Analysis

The central task of this section is regression, which is the process of estimating or generating understanding of a theoretical function $f : \mathcal{X} \rightarrow \mathbb{R}$ that describes the relationship between a response variable y and a group or single input variable described by the vector \mathbf{x} . While we do not know the exact behavior of f the function that describes this relationship⁵, what we do have is a set of observations $\mathcal{D} \subset \mathcal{X}$, where $\mathcal{D} = \{(x_i, y_i), i = 1, 2, \dots, n\}$ and $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. These observations constitute point-wise realizations of the function f . From that, for a set of new unknown points \mathbf{x}_* we want to predict or infer the quantity $f(\mathbf{x}_*) = y_*$, which would be point-wise estimates of these unseen or unknown realizations of f .

One example of another method of solving this problem is the well known and well loved linear regression, that assumes that the function f is of the general form

$$y = f(x) = x^T \beta + \epsilon$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. The error variance σ^2 and the coefficients β are generated by a process of minimizing the error of the outputs generated.

One of the disadvantages of these types of methods is that they only give one function as the output, but there may well be an infinite number of functions that fit a certain set of points. As an example, consider the following figure.



Figure 1.2: Gaussian Processes for ML, page 20.

⁵We will discuss this in later chapters, but this is actually one of the greatest advantages of Bayesian Optimization, in that it can work on so called *black-box* functions

Notice how for the training dataset \mathcal{D} , all of these functions have error zero! The only recourse we have is to restrict these functions to generate one function (or at least a finite family of functions), and not an infinity of them. As with many other areas of mathematics, there is a trade off that we have to choose to perform between how general a solution to a problem is and how many solutions to that problem we wish to get. **Expand on this.**

The key insight to understanding Gaussian process regression from a statistician's point of view is changing the way we think about regression. Instead of going the traditional way and providing restrictions to generate particular approximations, why don't we consider the space of possible functions that can fit these points as coming from a probability model? What if the observed functions in the past figure are realizations of a more general class of random variable that is coherent in the function space, rather than the discrete variable or vector space? From this groundwork is where Gaussian processes come into the picture. The only added restriction that we impose on the functions is that as a family, they come from a generalization to infinite dimensions of the multivariate normal distribution, and with that, we generate the entire conceptual framework from which we derive and generate the Gaussian process regression method. Having now set the expectation of what we wish to accomplish, we can now construct the theory of this different class of regression analysis.

1.4.2 Theory of Gaussian Process Regression (from the function space view)

This section of the text is heavily based on the work and structure of [RW⁺06] and to a much lesser extent [Wan20]. The book Gaussian Processes for Machine Learning is the go-to reference and source for modern applications of Gaussian processes in general and it would be a disservice to the reader to not follow that structure and argumentation.

We can split up the set \mathcal{D} into training and test, defining $\mathcal{D}_{train} = \{x_i, y_i\}_{i=1}^n$, and $\mathcal{D}_{test} = \{x_i^*, y_i^*\}_{i=1}^*$, with $x_i, x_i^* \in \mathbb{R}^d$ and $y_i, y_i^* \in \mathbb{R}$. As described before, what we want to achieve is basically generating out of sample predictions for the test set \mathcal{D}_{test} . Since the function $f = m$ defined is the mean of the process we defined before, we can simply evaluate the out of sample points x_i^* on f and we have an estimate for y_i^* . For simplicity and by convention, we denote the set of test outputs \mathbf{f}_* . Because of the structure we have defined, formally, the regression function f is really a normal distribution when conditioned by the training dataset \mathcal{D}_{train} , which would be

$$\mathbb{P}(\mathbf{f}|\mathbf{X}) = \mathcal{N}(f|\mu, k)$$

where $\mathbf{X} = [x_1, x_2, \dots, x_n]$, $\mathbf{f} = [f(x_1), f(x_2), \dots, f(x_n)]$, $\mu = [m(x_1), m(x_2), \dots, m(x_n)]$ and k the kernel or covariance function of the process. By convention, it is customary to set $m(\mathbf{X}) = 0$, but there is a reasoning behind this. When dealing with the type of data generally modelled by these types of functions, it is common (encouraged even) to normalize the data before applying these methods to it. Another important piece of shorthand notation we will use is $K = k(\mathbf{X}, \mathbf{X})$, $K_* = k(\mathbf{X}, \mathbf{X}_*)$, and $K_{**} = k(\mathbf{X}_*, \mathbf{X}_*)$. Note that since the number of training points need not equal the number of test points, then n has not attributable relationship with n^* and we cannot state any properties of the matrix K_* , much less say that it is a square matrix.

Since we have formally defined the distribution of \mathbf{f} and we know that \mathbf{f}_* is the same distribution but considering a different set of data points $\mathbf{X}_* = [x_1^*, \dots, x_n^*]$, we can define the joint distribution of these vectors as

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix} \right), \quad (1.1)$$

which is the joint probability distribution $\mathbb{P}(\mathbf{f}, \mathbf{f}_* | \mathbf{X}, \mathbf{X}_*)$. This is a massive step in constructing the regression model but we are still missing one component. When dealing with regression and prediction of values outside of the original training dataset, we do not need the joint probability, but the conditional probability of the test set given everything else we know. Formally, this means that we need to find the probability distribution of $\mathbb{P}(\mathbf{f}_* | \mathbf{X}, \mathbf{X}_*, \mathbf{f})$. Luckily, since we are dealing with Gaussian distributions, there is a very neat closed form expression that describes this probability. Referring to section A.2 of [RW⁺06], and noting that the mean for both \mathbf{f} and \mathbf{f}_* is 0, then

$$\mathbf{f}_* | \mathbf{f} \sim \mathcal{N}(K_*^T K^{-1} \mathbf{f}, K_{**} - K_*^T K^{-1} K_*).$$

This probability function is what is used to calculate out of sample predictions for the out of sample data \mathbf{X}_* .

Gaussian Process Regression Over Noisy Observations

When there is inherent randomness or noise in the signals being calculated, the structure of the problem itself changes. If we can assume that the noisy signals we are receiving are of the form $y = f(\mathbf{x}) + \varepsilon$ with the noise or inherent randomness captured by ε and these being independent, identically distributed observations of a normal or Gaussian distribution, the covariance of the observations therefore becomes

$$\text{cov}(\mathbf{y}) = K(\mathbf{X}, \mathbf{X}) + \sigma^2 I,$$

with I the identity matrix. In practical terms this means that the inherent variance of every observation is present as an added term on the diagonal of the covariance matrix (this is because of the assumption that the noise is independent). This change causes the joint distribution originally defined in equation 1.1 to

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K + \sigma^2 I & K_* \\ K_*^T & K_{**} \end{bmatrix} \right),$$

and the conditional distributions for prediction are therefore

$$\begin{aligned} \mathbf{f}_* | \mathbf{X}, \mathbf{y}, \mathbf{X}_* &\sim \mathcal{N}(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*)), \text{ where} \\ \bar{\mathbf{f}}_* &:= \mathbb{E}[\mathbf{f}_* | \mathbf{X}, \mathbf{y}, \mathbf{X}_*] = K_*^T [K + \sigma^2 I]^{-1} \mathbf{y} \\ \text{cov}(\bar{\mathbf{f}}_*) &= K_{**} - K_*^T [K + \sigma^2 I]^{-1} K_*. \end{aligned}$$

These equations are for the entire training and test set though. For a test point x_* , we can define the vector \mathbf{k}_* as the vector of covariances between this new test point and each of the points in the training dataset. This would be equivalent to taking the column associated with that single point from the test data matrix K_* . If we wanted just the closed form for this single out of sample observation, that would be

$$\begin{aligned} \bar{f}_* &= \mathbf{k}_*^T (K + \sigma^2 I)^{-1} \mathbf{y}, \\ \mathbb{V}(\bar{f}_*) &= k(x_*, x_*) - \mathbf{k}_*^T (K + \sigma^2 I)^{-1} \mathbf{k}_*. \end{aligned}$$

Chapter 2

Bayesian Optimization

This chapter (and the rest of this work, really) relies pretty heavily on the book [Gar23] as a source of consistency and terminology. While at the time of writing this book is still in pre-print, this might be the most approachable and complete treatment of Bayesian optimization the author has found. We rely on it for standardizing notation and for some of the proofs and concepts, as well as for a general sketch of how this chapter is set out.

2.1 Introduction

The necessity for Bayesian optimization comes from a relatively simple constraint problem, but not in the way we normally think of constraints when dealing with optimization. In traditional optimization problems, there are constraints to the structure of the function or the solutions that we can find. An example of this is integer optimization, where the objective function either only takes variables in a discrete form or we are interested only in the solutions to the problem that are integers. While we are finding local or global maxima or minima, what we generally want to solve when in the scope of traditional optimization is the number of steps we need to find an optimum element in a specific set to the function or set of functions we are evaluating. This remains true for Bayesian optimization, but there is an element that makes it unique- under this paradigm we consider the function that we are trying to find expensive to evaluate. What this means is that the nature and our approach to these types of problems has to change drastically, as we are not only concerned with finding the best solution to the problem, we are also interested in doing it in the most efficient way possible in terms of function evaluations. What this means generally is that we want to not only find the maxima or minima of the function, but we have this added restriction that

we have to do it in the least number of steps possible.

There are many scenarios where this paradigm is necessary because of the types of problems we are dealing with. One particular case which will become relevant to this work in later chapters is hyperparameter optimization in machine learning systems. When training large machine learning models as is becoming customary these days (look at the training time and size of for example any large language model), we want to do things in the most efficient manner possible. For many models, this means specifying things such as a learning rate in neural networks, which is the susceptibility of a certain model to change its weights depending on the error estimates generated. If this hyperparameter is set too high, and the model may be unstable in training and we may never reach a stable solution. If the weight is too low, then the model will change its weights too slowly and we will be left with either too much time dedicated to training or simply does not achieve good metrics or results since there is still learning to be done by it on the data that is available. These hyperparameters are generally chosen through expert judgement, transferred over from other similar models, or randomly shuffled through until trying to explore the space in which these hyperparameters exist. Bayesian optimization is a different approach, in which we assign probability of finding extrema in certain unexplored areas of the function space and iteratively search through this space until we either find these extrema or find a suitable candidate with a certain confidence that it is either the extrema or close to it.

As the models that we build and the problems we solve in general become more and more complicated, there has been a growing interest in this area of statistics as a solution to generating viable extrema candidates of expensive to evaluate functions. There has been extensive work on this field recently and in the past few decades, with examples such as [BYC⁺13a], [BKJ⁺20], and [ASY⁺19] that have impeded this field considerably. Hyperparameter optimization has recently been talked about as the "killer app" for Bayesian optimization, which revitalized interest in Gaussian Processes in the context of machine learning and Bayesian optimization in general. The interest in the field has led to a great amount of interest and therefore research and publications tackling the method and providing theoretical backing to it.

In this chapter we will show the basic theoretical structure of Bayesian optimization, the three central components that make it up, and the decisions we have to make to be able to approximate these functions from a Bayesian point of view.

2.2 Components of BO

Bayesian optimization is a sequential model-based approach to optimizing black-box functions that are expensive to evaluate. The goal is to find in some domain \mathcal{X} a value x that maximizes the objective function f with the least possible number of evaluations. Practically, this means numerically calculating the value $f(x)$ for the least amount of times possible. One important distinction from other areas of optimization is that we do not impose any restrictions on the function f , except continuity¹. Formally, let f be the objective function that we want to maximize, and let \mathcal{D} be the set of observations we have made so far, which consists of pairs (x_i, y_i) where x_i is the input and y_i is the corresponding output of the function $f(x_i)$. We want to find the input x that maximizes the function $f(x)$, i.e., $x^* = \arg \max_{\mathcal{X}} f(x)$.

The Bayesian optimization algorithm consists of three main components: a prior distribution over the objective function, an acquisition function, and a method for updating the prior distribution based on the observed data. The way this model works can be thought of as the following three steps:

1. Determining a probabilistic model that captures the observed behaviour of f , either by assuming a prior on the first step or going off data points $(x_i, f(x_i))$ for later iterations. We call this probabilistic mode of f the prior, since it is used basically as one under this scheme².
2. Optimizing the selected acquisition function that probabilistically estimates the best possible points for evaluation in the following iteration.
3. Evaluating the function at the best candidate for improvement, therefore getting new data and updating our probabilistic model of the behaviour of the function.

The prior distribution over the objective function $f(x)$ is usually assumed to be a Gaussian process, but that need not be the case. While these type of models are beyond the scope of this work, as examples, we refer to [SWG13] for the prior set to a t -Student distribution and [WLX⁺] for a case study of the Poisson prior. Given the set of observations \mathcal{D} , the selected prior distribution defines a posterior distribution that takes into account the uncertainty in the observations.

The acquisition function is a criterion that measures the expected utility of each point in the input space x for the purpose of optimization. The most

¹As we will see in later chapters, the entire point of this work will be relaxing this restriction.

²Another common way to jump-start this process is to begin by evaluating some random points in \mathcal{X} and continuing the process from there.

commonly used acquisition function is the expected improvement (EI), which measures the expected improvement in the objective function if we were to evaluate the function at a particular input x . Much like the case with the prior distributions, there are other choices such as Thompson sampling (TS) and Upper Confidence bound (UCF), references for each can be found in [KKM12] and [KCG12] respectively.

$$EI(x) = \mathbb{E}[\max(f(x) - f(x_{best}), 0)],$$

where x_{best} is the current best input found so far. The idea is to select the input x that maximizes the EI, which balances exploration (trying new inputs) and exploitation (focusing on inputs that are likely to be good) this balance is controlled by hyperparameters, so that is another layer of complexity that needs to be controlled. In a sense, what we are doing is differing solving the main optimization problem by generating a simpler to solve optimization problem in finding the relevant extrema of the acquisition function, thus allowing us to spend as little function evaluations as possible on evaluating the target function f .

The final component of BO is the method for updating the prior distribution based on the observed data. This is done under the overarching philosophy of Bayesian inference, which allows us to compute the posterior distribution over the objective function given the prior distribution and the observed data. The posterior distribution is then used as the prior distribution for the next iteration of the algorithm. What we wish to do is infer the value $\phi = f(x)$ for some unobserved point in the domain \mathcal{X} .

2.3 Formalization

This process begins by setting the prior distribution $\mathbb{P}(\phi|x)$ which is our belief of possible values for ϕ before observation of the unknown x .

Assuming we observe the objective function at x and measure y , our optimization model supposes that the distribution of this measurement is determined by the value of interest ϕ through the observation model $\mathbb{P}(y|x, \phi)$. With this new value y we can update our beliefs and generate the posterior function through the use of Bayes' theorem:

$$\mathbb{P}(\phi|x, y) = \frac{\mathbb{P}(\phi|x) \cdot \mathbb{P}(y|x, \phi)}{\mathbb{P}(y|x)}.$$

As with all applications of Bayes' theorem such as this one, the denominator is simply a constant term used to scale the probability. This means

that we can simplify the expression by only keeping the kernel of the distribution and dropping the normalizing term, and using the proportionality operator \propto , as follows.

$$\mathbb{P}(\phi|x, y) \propto \mathbb{P}(\phi|x) \cdot \mathbb{P}(y|x, \phi).$$

In the Bayesian framework, the posterior distribution is often not the final goal but a starting point for further tasks such as prediction or decision-making. This builds on the iterative process under which we are working in Bayesian optimization, where new information is added to the approximation of the function every time a function evaluation happens. As an example, consider the case where we want to predict the outcome of an independent, repeated noisy observation at x, y' . We can treat the outcome as a random variable and derive its distribution by integrating our posterior belief about ϕ against the observation model:

$$\mathbb{P}(y'|x, \mathcal{D}) = \int \mathbb{P}(\phi|x, \mathcal{D}) \cdot \mathbb{P}(y'|x, \phi) d\phi.$$

This equation represents the posterior predictive distribution for y' given we know x and a set of data \mathcal{D} . By integrating over all possible values of ϕ , weighted by their plausibility, the posterior predictive distribution naturally accounts for uncertainty in the unknown objective function value.

From here is where the link to Gaussian Processes lies. Gaussian Processes serve as a natural bridge between the Bayesian framework and Bayesian optimization. They provide a flexible, non-parametric prior of the form $\mathbb{P}(f)$ over a certain function f , and their posterior predictive distributions can be efficiently computed and used to guide the search for the optimal input value. The acquisition function, which is derived from the GP's posterior predictive distribution, plays a crucial role in balancing exploration and exploitation during the optimization process.

Continue here on the final part of chapter one of BO.

Chapter 3

BASS and their Origins

The main proposal in this work is the use of a BASS model as the surrogate model in the framework of Bayesian Optimization rather than a traditional Gaussian Process. To understand this method and why it is useful for this purpose, we develop first the theory that lead to the construction of this method, and then explain it's behaviour in some detail.

As with most other algorithms used for regression tasks, BASS is built upon a rich history of work that was previously done in the field. It begins with CART and decision trees, which MARS models are based on, and in turn BMARS (Bayesian MARS) uses the same model form as MARS but builds the model using the traditional Bayesian approach of defining a prior and constructing from there. Finally, BASS is a specific case of the BMARS model. This is much more clear when viewed as a diagram, as shown in the following figure.



Figure 3.1: The theoretical basis for BASS.

3.1 Decision Trees

Decision trees are a fundamental and versatile tool that comes in two main flavours: classification trees and regression trees. These two types of trees serve distinct purposes and are employed to make predictions in different contexts. The term "Classification and Regression Tree" (CART) analysis serves as an umbrella term that encompasses both classification and regression tree procedures. CART was first introduced by Breiman et al. in 1984 [Bre17], and it represents a unified framework for decision tree modelling, allowing for flexibility in handling both classification and regression tasks within the same algorithm. While classification and regression trees share common principles, such as recursive binary splitting and impurity measures (e.g., Gini impurity for classification and mean squared error for regression), they also exhibit some differences in terms of the criteria used to determine the best feature and split point at each node. The final predictions for a new data point are obtained by traversing the tree from the root node to a leaf node and returning the average (for regression) of the target variable for the training samples in that leaf node.



Figure 3.2: A decision tree.

In classification tree analysis, the primary objective is to predict the discrete class or category to which a given data point belongs. This technique finds relevance in numerous applications, including disease diagnosis, sentiment analysis, and fraud detection. The decision tree algorithm partitions the data into subsets based on input features and assigns each data point to a specific class by following a path through the tree.

On the other hand, regression tree analysis is employed when the predicted outcome is a continuous, real-valued number. Tasks like predicting housing prices, stock market trends, or a patient's length of stay in a hospital rely on regression trees. Similar to classification trees, regression trees partition the data based on input features. However, in this context, the goal is

to estimate a numeric value for each data point, typically by computing the average of the target variable within each leaf node.

In both cases, the decision tree serves as a transparent and interpretable model, making it valuable for exploratory data analysis and predictive modelling. The tree's structure allows for a straightforward understanding of how decisions are made, and predictions for new data points are obtained by navigating the tree from the root node to an appropriate leaf node, where the final prediction is based on the characteristics of the training samples within that leaf.

The versatility and comprehensibility of decision trees have contributed significantly to their widespread use in various fields, and the distinctions between classification and regression trees make them adaptable tools for addressing a wide range of data analysis and prediction tasks. One problem that these methods tend to have though is overfitting, since if we allow the model to keep generating new nodes or leafs, the tree ultimately becomes too complex and every data point that we would like to analyse is simply too narrow to be useful as an aggregation scheme.

While decision trees are useful on their own their real power comes from the versatility they provide, leading to a wide range of methods being born from the basic idea of splitting decisions into paths. These methods are sometimes called *ensemble* methods, because they involve the creation and combination of multiple models to form a more robust and accurate predictive model. As an interesting side note, the term ensemble is borrowed from the world of music and theatre. Instead of relying on a single predictive model, these models create an ensemble of multiple models, each of which may have its own strengths and weaknesses. These individual models, often referred to as “base models” or “weak learners”, are combined in a way that leverages their collective intelligence to make better predictions.

Ensemble methods work on the premise that combining diverse models can reduce the risk of overfitting, enhance generalization, and improve predictive accuracy. Each model in the ensemble may make errors on some portions of the data, but by aggregating their predictions, the ensemble can produce more accurate and stable results. Common ensemble methods include Random Forest, Gradient Boosting, Adaptive Boosting, Bagging, and many others. The difference in these models is how they combine these different decision trees to achieve a more stable model.

3.2 MARS Models

Now, let's shift our focus the following technique down the chain for us known as MARS, which stands for Multivariate Adaptive Regression Splines. MARS builds upon the foundation laid by decision trees but introduces a different paradigm for modelling complex relationships within data. Rather than using piecewise constant approximations, MARS leverages piecewise linear functions, thereby extending the toolbox of non-linear regression methods.

MARS is particularly valuable when the relationships between input features and the target variable are not strictly linear but can be approximated as a series of linear segments. Let's delve into the mathematical foundations of MARS to gain a deeper understanding.

For the following section, we will adopt the terminology and follow the development presented in [Fri91a] and [Fri91b].

MARS is defined using functions that are piecewise linear in the form of $(x - t)_+$ and $(t - x)_+$. These functions take the maximum between 0 and the value within the function, as follows:

$$(x - t)_+ = \max\{0, x - t\} = \begin{cases} x - t & \text{if } x > t, \\ 0 & \text{if } x \leq t. \end{cases}$$

Each function is piecewise linear with a slope change at the value t , which makes them linear splines. Each pair divided at the value t is referred to as a "reflected pair". The idea of the method is to form reflected pairs for each input variable X_j with slope changes at each observed value x_{ij} . Following this construction and considering a variable X_j , the collection of basis functions is:

$$\mathcal{C} = \{(X_j - t)_+, (t - X_j)_+\}, \text{ con } t \in \{x_{1,j}, x_{2,j}, \dots, x_{N,j}\}, \text{ y } j = 1, 2, \dots, p.$$

And each reflected pair appears as follows, and are shown in figure 3.3:

$$h_1(X) = h(X_j - x_{ij})$$

$$h_2(X) = h(x_{i,j} - X_j)$$

So, for each X_j , the set of candidate reflected pairs that exist at each of the x_{ij} points of that variable is shown in figure 3.4.

This means that if all input values are distinct, there are a total of $2Np$ basis functions, and $2N$ divisions in each of the splines for each variable. The model we use to combine all variables is an additive one in β :

$$f(X) = \beta_0 + \sum_{m=1}^M \beta_m h_m(X),$$

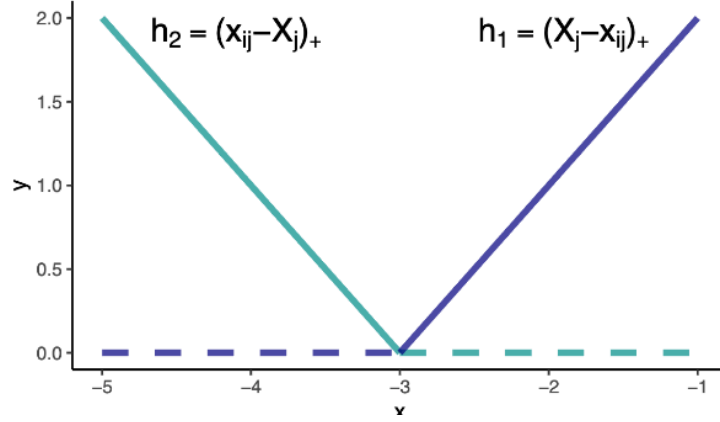


Figure 3.3: The constructed reflected pairs.

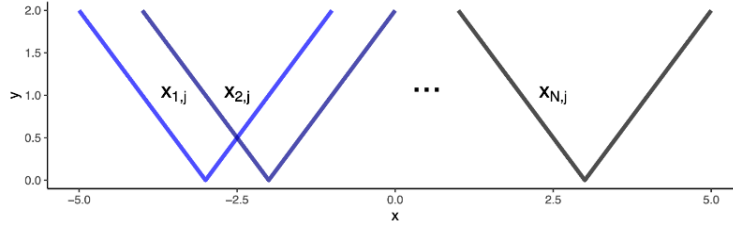


Figure 3.4: Collections of hinge functions.

where each function $h_m(X)$ is an element of \mathcal{C} or a linear combination of these basis functions. For this initial explanation, we will use only functions that do not involve interactions (the model with interactions works very similarly to CART decision trees).

The model construction process (denoted as \mathcal{M}) begins with the base model $\hat{f}(X) = \hat{\beta}_0 = h_0(X) = 1$, where a term functioning as an intercept at the origin is defined. To add terms incrementally, all functions in the set \mathcal{C} are candidates for entry into \mathcal{M} . For each observation x_{ij} , the new model involving these functions is fitted as follows:

$$\hat{f}(X) = \hat{\beta}_0 + \hat{\beta}_1 h_1(X) + \hat{\beta}_2 h_2(X).$$

Since this is a linear form in terms of each of the functions h_i , the adjustment of each parameter $\hat{\beta}_i$ is done by minimizing the sum of squares.

The reflected pair that is added is the one that minimizes the training error. This process is repeated for each of the remaining reflected pairs, solving OLS to determine the new β , and adding the one that continues to minimize the error. The stopping criterion can be either that none of the

remaining reflected pairs reduces the error enough (in absolute or relative terms) or until we have a predetermined number of variables in the model.

This leads to overfitting the data, but this is the goal in this part of the procedure when minimizing the training error. Once we have \mathcal{M} , we start the second part of model fitting, which is the pruning part. In this case, terms $h_i(X)$ are removed iteratively, starting with the one that produces the smallest increase in the residual sum of squares when removed. This procedure yields a better model for each λ size, denoted as \hat{f}_λ .

Several procedures can be used to estimate the optimal value of λ , such as cross-validation or bootstrap, but this involves a significant computational cost. To minimize this cost, MARS models generally use a generalized cross-validation (GCV) procedure. This criterion is defined as:

$$GCV(\lambda) = \frac{\sum_{i=1}^N (y_i - \hat{f}_\lambda(x_i))^2}{\left(\frac{1-M(\lambda)}{N}\right)^2},$$

where the value $M(\lambda)$ is the number of effective parameters in the model, depending on the number of terms plus the number of breakpoints used, penalized by a factor (this factor is 2 in the case for additive in X that we are explaining, and 3 when there are interactions).

Now that we have described the base model, we can go back and consider the differences when incorporating interaction terms. Instead of just adding terms additively, products between the existing h_ℓ functions in the model and between each reflected pair of each x_{ij} can be added. Note that this general case encompasses the previous one because if you want to add a reflected pair without interacting with the other variables, you simply multiply by $h_0 = 1$.

In this way, a model \mathcal{M} with M terms will be updated as follows after finding the combination $\hat{\beta}_{M+1}h_\ell(X) \cdot (X_j - t)_+ + \hat{\beta}_{M+2}h_\ell(X) \cdot (t - X_j)_+$ that best minimizes the training error:

$$\hat{f}(X) = \hat{\beta}_0 + \sum_{m=1}^M \hat{\beta}_m h_m(X) + \hat{\beta}_{M+1} h_\ell(X) \cdot (X_j - t)_+ + \hat{\beta}_{M+2} h_\ell(X) \cdot (t - X_j)_+,$$

where $h_\ell \in \mathcal{M}$ and $t = x_{ij}$ for any case. **Como transiciono de una idea a otra aqui?**

One interesting consideration is that the added flexibility of these models does not end with it being piecewise linear, as one small alteration that can be made is to replace the truncated linear functions with truncated cubic basis functions or other functions of a higher order, which provides allows the resulting model to be fully differentiable within its domain.

In conclusion, our exploration of Multivariate Adaptive Regression Splines (MARS) is a powerful and versatile technique for modelling complex relationships within data. Unlike traditional linear regression, MARS leverages piecewise linear functions, making it a valuable tool when dealing with non-linear relationships between input features and the target variable. The methodology behind MARS involves the creation of reflected pairs and the incremental addition of basis functions to the model, effectively adapting to the data's intricacies. The model construction process aims to minimize the training error by iteratively selecting the reflected pair that provides the most significant reduction in error. This leads to a model that can potentially overfit the data, but this is by design during the training phase. Subsequently, a pruning step is employed to refine the model, removing terms that do not contribute significantly to the model's predictive power.

3.3 BMARS Models

The key innovation that distinguishes BMARS from its predecessor lies in its Bayesian framework. While MARS primarily focuses on constructing piecewise linear models by iteratively selecting basis functions to minimize training error, BMARS introduces Bayesian methods to bring a probabilistic perspective into this process. This integration of Bayesian principles allows BMARS to provide not only point estimates of model parameters but also entire probability distributions, enabling a more comprehensive understanding of the uncertainty inherent in the modelling process.

As described in [DMS98b], which is the original paper where this method was proposed, the idea behind this model is to allow more flexibility in the traditional MARS model by looking at it from a Bayesian perspective. This is done by mimicking the MARS model with changes in framing, namely by considering the number of basis functions, along with their *type*, their coefficients and their form (the positions of the split points and the sign indicators) as random.

We take the definition of a basis type from the original paper and a small example from there to make the definition tangible, and include it here for completeness.

Definition 10. *We consider basis functions X_i and X_j to be of the same type if $[x(1, i), \dots, x(J_i, i)]$ is identical to some permutation of $[x(1, j), \dots, x(J_j, j)]$. Hence, with m predictor variables, there are N different types of basis func-*

tions, where N is given by:

$$N = \sum_{i=1}^m \binom{m}{i} = 2^m - 1. \quad (3.1)$$

Note that the sum does not include the constant basis function term (for which i would equal 0 in (3.1)), as this basis X_1 is always the sole constant basis function in the model, so it cannot be chosen as a candidate basis. Frequently, some maximum order of interaction I is assigned to the model, such that $J_i \leq I$ for $i = 1, \dots, k$. In which case, the sum in (3.1) only runs from 1 up to I . We let $T_i \in \{1, 2, \dots, N\}$ denote the type of basis function X_i ; thus, T_i effectively tells us which predictor variables we are splitting on, i.e., what the values of $x(1, i), \dots, x(J_i, i)$ are.

As an example, suppose we have a problem with just two predictors, i.e., $m = 2$. Then the types of basis functions that could be in the model (not including the constant one) are $[\pm(x_1 - *)]$ (say, type 1), $[\pm(x_2 - *)]$ (say, type 2), and $[\pm(x_1 - *)][\pm(x_2 - *)]$ (say, type 3). So for all those basis functions which split only on predictor x_2 , their types T_i are all equal to 2.

The key insight that this framework of construction is based on is that any MARS model can be uniquely defined by the following:

- The number of basis functions k ,
- the coefficients of the basis functions a_i ,
- the type of basis functions T_i ,
- the knot points associated with each interacting term t_{ji} ,
- the sign indicators associated with each interacting term s_{ji} .

and as such, a probability distribution over the possible MARS models can be constructed, enabling a more comprehensive understanding of the uncertainty inherent in the modelling process since the response is an entire probability distribution rather than a single model. This framework also provides the ability to do ANOVA and Sobol decompositions for sensitivity analysis. One important aspect of this framework for construction is that the number of parameters varies with the number of basis functions k , and as such, a traditional method such as Markov chain Monte Carlo cannot be used directly, as it assumes constant parameters. An extended version of MCMC is therefore used, called reversible jump MCMC (first introduced in [Gre95a]), that allows for a variable number of parameters. *Cual es el modelo? Tengo el paper pero no lo veo claro.*

3.4 BASS Models

Bayesian Adaptive Spline Surface (BASS) models are an extension of the family of non parametric regression models of which multivariate adaptive regression splines (MARS) was the first developed in 1991 and is the most basic [Fri91b]. MARS is an extension of linear models, where interactions between variables and non linearities are modelled automatically.

The extension of this model of interest is the BMARS model. As the authors state in [DMS98b], the aim of BMARS is to provide a Bayesian model that mimics the MARS procedure. This is done by considering the number of basis functions, along with their type, the coefficients and their form, random. Then, a suitable MCMC algorithm, in this case reversible jump monte carlo [Gre95a], is used to calculate a suitable posterior for the problem. The great advantage to RJMCMC over other MCMC techniques is that the number of parameters can be variable, and as such, is a great basis for the type of analysis that is necessary for models in the MARS class.

BASS models are an adaptation of BMARS models that include some efficiency improvements for posterior sampling alongside parallel tempering for better posterior exploration [FS20a]. For our purposes though, the most significant improvement is that it allows the use of categorical variables alongside some continuous variables. This is incredibly useful in the context of hyper-parameter optimization, where there are generally some categorical variables such as polynomial degree, smoothing, etc. While we remain in the one dimensional case and this improvement is not necessarily useful, the multi dimensional case will benefit greatly from this improvement.

A detailed rundown of the model, its priors, and its construction in general can be seen in [FS20a], but, we include the definition of the model presented in that work for completeness.

Let y_i denote the dependent variable and \mathbf{x}_i denote a vector of p independent variables, with $i = 1, \dots, n$. Without loss of generality, let each independent variable be scaled to be between zero and one. We model y_i as

$$y_i = f(\mathbf{x}_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

$$f(\mathbf{x}) = a_0 + \sum_{m=1}^M a_m B_m(\mathbf{x})$$

$$B_m(\mathbf{x}) = \prod_{k=1}^{K_m} g_{km} [s_{km}(\mathbf{x}_{v_{km}} - t_{km})]_+^\alpha$$

The only difference between this setup and that of MARS and BMARS is the inclusion of the constant g_{km} in each element of the tensor product.

This term is $g_{km} = [s_{km}(\mathbf{x}_{v_{km}} - t_{km})]_+^\alpha$, where $s_{km} \in \{-1, 1\}$ is referred to as the sign, $t_{km} \in [0, 1]$ is a knot, v_{km} selects a variable, K_m is the degree of interaction, and α determines the degree of the polynomial splines. This normalizes the basis functions so that the basis coefficients a_1, \dots, a_M are on the same scale, making computations more stable.

Bibliography

- [ASY⁺19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.
- [BBBK] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. page 9.
- [Bil12] P. Billingsley. *Probability and Measure*. Wiley Series in Probability and Statistics. Wiley, 2012.
- [BKJ⁺20] Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G Wilson, and Eytan Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization. *Advances in neural information processing systems*, 33:21524–21538, 2020.
- [Bre17] Leo Breiman. *Classification and regression trees*. Routledge, 2017.
- [BYC⁺13a] James Bergstra, Dan Yamins, David D Cox, et al. 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, 2013.
- [BYC⁺13b] James Bergstra, Dan Yamins, David D Cox, et al. 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, 2013.
- [Chu60] Kai Lai Chung. *Markov chains with stationary transition probabilities*. Springer, Berlin, 1960. OCLC: 682058891.

- [DMS98a] D. G. T. Denison, B. K. Mallick, and A. F. M. Smith. Bayesian mars. *Statistics and Computing*, 8(4):337–346, 1998.
- [DMS98b] David GT Denison, Bani K Mallick, and Adrian FM Smith. Bayesian mars. *Statistics and Computing*, 8:337–346, 1998.
- [DMTK20] Erik Daxberger, Anastasia Makarova, Matteo Turchetta, and Andreas Krause. Mixed-variable bayesian optimization. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, page 2633–2639, Jul 2020. arXiv:1907.01329 [cs, stat].
- [Fra18] Peter I. Frazier. A tutorial on bayesian optimization. (arXiv:1807.02811), Jul 2018. arXiv:1807.02811 [cs, math, stat].
- [Fri91a] Jerome H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1), Mar 1991.
- [Fri91b] Jerome H Friedman. Multivariate adaptive regression splines. *The annals of statistics*, 19(1):1–67, 1991.
- [FS20a] Devin Francom and Bruno Sansó. Bass: An r package for fitting and performing sensitivity analysis of bayesian adaptive spline surfaces. *Journal of Statistical Software*, 94(LA-UR-20-23587), 2020.
- [FS20b] Devin Francom and Bruno Sansó. Bass : An r package for fitting and performing sensitivity analysis of bayesian adaptive spline surfaces. *Journal of Statistical Software*, 94(8), 2020.
- [Gar23] Roman Garnett. *Bayesian Optimization*. Cambridge University Press, 2023. to appear.
- [Gha11] Zoubin Ghahramani. A tutorial on gaussian processes (or why i don’t use svms). In *Proc. MLSS Workshop Talk Gaussian Processes*, 2011.
- [GMHL20] Eduardo C. Garrido-Merchán and Daniel Hernández-Lobato. Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes. *Neurocomputing*, 380:20–35, Mar 2020. arXiv:1805.03463 [cs, stat].
- [Gre95a] Peter J Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82(4):711–732, 1995.

- [Gre95b] Peter J. Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82(4):711–732, 1995.
- [HR05] Johannes M Hohendorff and J Rosenthal. An introduction to markov chain monte carlo. *University of Toronto, Department of Statistics, supervised reading report (<http://www.probability.ca/jeff/grad.html>)*, 2005.
- [KCG12] Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. On bayesian upper confidence bounds for bandit problems. In *Artificial intelligence and statistics*, pages 592–600. PMLR, 2012.
- [KFB⁺17] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. (arXiv:1605.07079), Mar 2017. arXiv:1605.07079 [cs, stat].
- [KKM12] Emilie Kaufmann, Nathaniel Korda, and Rémi Munos. Thompson sampling: An asymptotically optimal finite-time analysis. In *International conference on algorithmic learning theory*, pages 199–213. Springer, 2012.
- [LGN⁺19] Phuc Luong, Sunil Gupta, Dang Nguyen, Santu Rana, and Svetha Venkatesh. *Bayesian Optimization with Discrete Variables*, volume 11919 of *Lecture Notes in Computer Science*, page 473–484. Springer International Publishing, Cham, 2019.
- [RCC99] Christian P Robert, George Casella, and George Casella. *Monte Carlo statistical methods*, volume 2. Springer, 1999.
- [RK16] Reuven Y Rubinstein and Dirk P Kroese. *Simulation and the Monte Carlo method*. John Wiley & Sons, 2016.
- [Ros96] Sheldon M. Ross. *Stochastic processes*. Wiley series in probability and statistics. Wiley, New York, 2nd ed edition, 1996.
- [RW⁺06] Carl Edward Rasmussen, Christopher KI Williams, et al. *Gaussian processes for machine learning*, volume 1. Springer, 2006.
- [SB] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Published: Retrieved September 10, 2022, from <http://www.sfu.ca/ssurjano>.

- [SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. (arXiv:1206.2944), Aug 2012. arXiv:1206.2944 [cs, stat].
- [SWG13] Amar Shah, Andrew G Wilson, and Zoubin Ghahramani. Bayesian optimization using student-t processes. In *NIPS Workshop on Bayesian Optimization*, 2013.
- [Wan20] Jie Wang. An intuitive tutorial to gaussian processes regression. *arXiv preprint arXiv:2009.10862*, 2020.
- [WLX⁺] Xiaoxing Wang, Jiaxing Li, Chao Xue, Wei Liu, Chaoyue Wang, Weifeng Liu, Xiaokang Yang, Junchi Yan, and Dacheng Tao. Poisson process for bayesian optimization.
- [Ži10] Gordan Žitković. Introduction to stochastic processes-lecture notes. *Department of Mathematics, The University of Texas at Austin*, 2010.