KU LEUVEN

FACULTY OF SCIENCE

Machine Learning In Quantitative Finance

A kernel of truth

Lennert VAN DER SCHRAELEN

Supervisor: Prof. W. Schoutens
Department Mathematics, Statistics & Risk Section

Mentor: *S. Reyners*
Department Mathematics, Statistics & Risk Section

Thesis presented in
fulfillment of the requirements
for the degree of Master of Science
in Mathematics

Academic year 2019-2020

Contents

Acknowledgement

I wish to show my gratitude to my supervisor, professor Schoutens Wim, and my mentor, Reyners Sofie, proposing this interesting subject and providing a pleasant working atmosphere by giving support when necessary.

I wish to show my respect to professor Quaegebeur Johan, who learned me to understand mathematics.

Next, I wish to thank my mother, Jannsens Jo, my father, Van der Schraelen Danny, and my brother, Van der Schraelen Arne, since they are just always there for me.

I would like to pay my special regards to my classmates of Mathematics, which are responsible for an unforgettable 5 years. Special thanks are given to Andries Hannah, Bouwen Ben, Dompas Annabel, Gielis Simon, Nevelsteen Stephanie, Op de Beeck Nicolas, Schruers Dorien, Torfs Sofie, Van den Borne Bram, Vanmechelen Pieter, Van Kruijsdijck Gregory, Vermeiren Stephanie and Wolfs Jasper.

Finally, I wish to show my gratitude to Ardenoy Arno, Olbrechts Ruben and Stuyck Raf. I hope we stay lifelong friends despite the different paths we may take.

> But what wisdom is there within us
> To live based on the feeling of our hearts
> How many times has instinct let us down
> Never to be thought through, never to be questioned
>
> (As I Lay Dying: 'The sound of truth')

Symbols And Notation

We try to use notations and abbreviations in a consistent way. Before using a non-prominent notation or abbreviation for the first time in the text, we always repeat its definition. A list of abbreviations can be found in table 1.

BBMM	Black-box matrix matrix
BM	Bayesian methods
BS	Black and Scholes
DGP	Deep Gaussian process
DKL	Deep kernel learning
GP	Gaussian process
GPR	Gaussian process regression
HMC	Hamiltonian Monte Carlo
KISS-GP	Kernel interpolation for scalable structured Gaussian processes
MAP	Maximum a posteriori
MCMC	Markov chain Monte Carlo
MMM	Matrix matrix multiplication
MVM	Matrix vector multiplication
NUTS	No U-turn sampler
PR	Polynomial regression
RKHS	Reproducing kernel Hilbert space
RR	Ridge regression
SDE	Stochastic differential equation
SKI	Structured kernel interpolation
SKIP	Structured kernel interpolation for products
SoR	Subset of regressors
SVG	Stochastic variational GPR
VFE	Variational free energy / variational GPR
VG	Variance gamma

Table 1: List of abbreviations.

Next, we list all the specific notations we use in this thesis which can be found in table 2. Matrices are capitalized and vectors are in bold type. A quantity corresponding with a test set is denoted with a superscript asterix.

$\setminus$	Left matrix divide: $X = A \setminus B$ is the solution of $AX = B$
$:=$	Equality which is a definition
$\circ$	Element-wise multiplication
$\sim$	Is distributed as
$(x)^+$	The maximum of x and zero
$\tilde{A}$	A small modification (on A)
$\|\|.\|\|_{\mathcal{H}}$	RKHS norm
$< ., . >_{\mathcal{H}}$	RKHS inner product
$\|A\|$	Determinant of A
$\boldsymbol{y}^t$	the transpose of $\boldsymbol{y}$
∇	Gradient
$\bigotimes$	Kronecker product
B	Banach space
$\boldsymbol{c}$	Cluster center
C	Cluster
δ_x	Evaluation functional
$\mathbb{E}$	Expectation
$\boldsymbol{f}$	Vector of function values $\boldsymbol{f} = (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n))^t$ (GPR)
$\boldsymbol{f}^*$	Posterior prediction
γ	Gain or learning rate
Γ	For brevity
H	Barrier
$H()$	Hamiltonian function
$\mathcal{H}$	Hilbert space (Functional analysis), model structure (Bayesian analysis)
I_n	$n \times n$ Identity matrix
κ	The speed of mean reverting in the Heston model
k	Kernel (GPR), log strike price (finance)
$k_{\boldsymbol{x},\boldsymbol{z}}$	Validation of kernel with input $\boldsymbol{x}, \boldsymbol{z}$
K	Gram matrix (GPR), strike price (finance)
$K(X, X) = K_{X,X}$	$n \times n$ Covariance (or Gram) matrix
$K_{X,Z}$	$n \times m$ Covariance (or Gram) matrix
$\hat{K}_{X,X}$	Equals $\tilde{K}_{X,X} + \sigma^2 I_n$
KL	Kullback-Leiber divergence
$\mathcal{L}$	Variational lower bound
λ_i	ith Eigenvalue
Λ	For brevity (sparse GPR)

m	Number of inducing points
$\boldsymbol{m}$	Mean of the inducing points
n^*	Number of predictions
n	Number of observations in the model / which we want to predict
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2
$O(\cdot)$	Big oh notation
$p(\boldsymbol{y}\|\boldsymbol{x})$	The density of $\boldsymbol{y}$ given $\boldsymbol{x}$
q	Dividend yield (finance), Approximate distribution (GPR)
$q_{s_T}(s)$	Risk neutral density of log stock price at maturity
$q_{\boldsymbol{x}_i,\boldsymbol{x}_j}$	Validation of the SoR approximation kernel with input $\boldsymbol{x}_i, \boldsymbol{x}_j$
Q	Risk-neutral pricing measure
$Q_{A,B} = K_{A,Z}K_{Z,Z}^{-1}K_{Z,B}$	For brevity (sparse GPR)
ρ	Correlation parameter
r	Interest rate
$\mathbb{R}$	The real numbers
σ	Volatility (finance) or noise variance (GPR)
Σ	Variance or for brevity (sparse GPR)
S	Variance of the inducing points
S_t	Price of the stock at time t
θ	The vol-of-vol in het Heston model
$\boldsymbol{\theta}$	Vector of Hyperparameters
Θ	Parameter space
t	Time
T	Maturity
$\mathrm{Tr}\,(A)$	Trace of A
$\boldsymbol{u}$	Response vector for Z
$\phi(\cdot)$	Feature map
$\phi_T(\cdot)$	Characteristic function of $q_{s_T}(s)$
$\varphi(\cdot)$	Eigenfunction
$\boldsymbol{v}$	(Eigen)vector
$\mathbb{V}$	Variance
v_t	Variance at time t Heston model
W	Sparse matrix
X	Design matrix containing the covariates
X_t	Stochastic process
$\mathcal{X}$	Input space
$\boldsymbol{y}$	Response vector for X
$(\Omega, \mathcal{F}, \mathcal{P})$	probability space
w	Weights (for example for regression)
W_t	A brownian motion
$\boldsymbol{z}$	Latent variables (sparse GPR), Samples from a normal distributed r.v.
Z	Parameters corresponding to the inducing values $\boldsymbol{u}$ (sparse GPR)

Table 2: List of notations.

Chapter 1

Abstract

This thesis is the continuation of (De Spiegeleer et al., 2018), which is a paper written by De Spiegeleer Jan, Madan Dilip B, Reyners Sofie (my mentor) and Schoutens Wim (my supervisor). Both documents aim to deploy machine learning in the context of traditional quant problems. We focus on the fast pricing of derivatives using Gaussian process regression (GPR). Instead of executing time expensive calculations in order to price derivatives, we aim to train a model which we can consult achieving a speed-up. However, the price we have to pay for this speed-up is some loss in accuracy.

We start with discussing the necessary financial knowledge concerning pricing derivatives. Thereafter, we give a short summary concerning the necessary mathematical pre-knowledge which enables us to understand Gaussian processes. Next, we discuss the basic algorithm for GPR which has a time complexity of $O(n^3)$, $O(n^2)$ for training and fitting respectively. Subsequently, we try to modify/improve this algorithm obtaining a speed up for training and prediction and discuss when these new algorithms are suitable. We discuss different techniques which among others exploit the structure of the data, approximate our problem and/or use advanced numerical methods. Some of these modifications achieve a considerable speed-up for training and/or prediction with only a little loss of accuracy. Thereafter, we include a section discussing Bayesian and deep techniques for completeness. We end with a data study pricing different types of derivatives testing different GPR models.

We include a quite extensive appendix to which we often refer. Hence, it is recommended to print or to use two files which enables easy swapping. A handy scheme giving an overview of the thesis and the methods we discuss can be found in section 10.1.

Chapter 2

Some Financial Pre-knowledge

In this chapter, we discuss some financial derivatives which we try to price in our data study. We also give a very short introduction to models modelling the time evolution of a stock.

2.1 Options

An option is a derivative which gives the investor the right, not an obligation, to buy or to sell a stock satisfying some predetermined terms and conditions. The buyer of a call option has the right to buy the stock from the seller at a certain time for the strike price (K). The buyer of a put option has the right to sell the stock to the seller at a certain time for the strike price. The two simplest types of options are European calls (EC) and puts (EP) which are also known as vanillas. These options can only be executed at the maturity (T). We denote with Q the risk-neutral measure or pricing measure and with $+$ the maximum of a value and zero. We obtain the following expressions concerning their initial price

$$\text{Initial price EC}(K,T) := \exp(-rT)\mathbb{E}_Q[(S(T)-K)^+] \tag{2.1}$$
$$\text{Initial price EP}(K,T) := \exp(-rT)\mathbb{E}_Q[(K-S(T))^+]. \tag{2.2}$$

To obtain the pay-off, we just need to omit the expectation and the $\exp(-rT)$ (discounting) factor. Subsequently, we try to price a down-and-out barrier put. If the stock price remains above a barrier H during its lifetime T, it has the same structure as a European put with strike K. We also try to price a down-and-in barrier put, which has the same structure as a European put if the option goes beneath a barrier at least one time. The initial prices for these derivatives are given by

$$\text{Initial price DOBP} := \exp(-rT)\mathbb{E}_Q[(K-S(T))^+]1(\min_{0\le t\le T} S(t) > H) \tag{2.3}$$
$$\text{Initial price DIBP} := \exp(-rT)\mathbb{E}_Q[(K-S(T))^+]1(\min_{0\le t\le T} S(t) <= H). \tag{2.4}$$

Notice that a lot of other barrier options are possible. We end with pricing American options. For this type of option, the holder can exercise his rights during the entire life-time of the option and not only at the maturity. In the next section we try to model the time evolution of a stock. We begin with the well-known Black and Scholes (BS) model.

2.2 The Black And Scholes Model

We follow the approach of (Schoutens, 2008), where the interested reader can find more details.

Definition 1 (Geometric Brownian Motion) *Denote with W_t a Brownian motion (see appendix section 11.1). Furthermore, denote with S_t the price of the stock at time t, with r the interest, q the dividend yield and with σ the volatility. The stochastic differential equation*

$$dS_t = S_t((r-q)dt + \sigma dW_t), \tag{2.5}$$

has a unique solution given by

$$S_t = S_0 \exp\left(\left((r-q) - \frac{\sigma^2}{2}\right)t + \sigma W_t\right), \tag{2.6}$$

which is called the geometric Brownian motion or the Black and Scholes model.

The expression of the geometric Brownian motion can be found using Itô's lemma. Notice that the logarithm of S_t has a normal distribution and thus S_t has a log-normal distribution. By using r and q in this way, we are working in a risk neutral world.

Due to the simple expression of the stock price, one can price many derivatives very easily and even closed form solutions for different options exist. One can even derive the BS model as the limit of a tree model. For derivative pricing using tree models, see appendix section 11.2.

However, the model is often too simple and has a lot of shortfalls. The log returns of empirical data typically do not behave according to a normal distribution and show most of the time negative skewness and excess kurtosis. Hence, the Black-Scholes model can not model realistically extreme events. Notice that the model is continuous and shows no jumps, which is not the case in the financial market. Finally, the volatility parameter, which is the (only!) parameter, is assumed to be constant. However, the parameters of uncertainty change stochastically over time and are clustered in real life.

Hence, the BS model not always suffices. In this thesis, we work with two well-known models in the financial world which are more flexible and have better properties.

2.3 The Heston And Variance Gamma Model

In this section, we state some important results from (Heston, 1993) for the Heston model and from (Madan et al., 1998) and (Schoutens, 2008) for the Variance gamma model.

Definition 2 (Heston stochastic volatility model) *Let us denote with $v_t > 0$ the variance (volatility) of the model at time t and with $\sigma_0 > 0$ the standard deviation at time 0. The paramater $\theta > 0$ is the volatility of the variance (vol-of-vol) and η represents the long time variance of the model. The speed of the mean reverting of v_t to η is represented by $\kappa > 0$. The Brownian motion $\tilde{W}_t$ is correlated (parameter $-1 < \rho < 1$) with the Brownian motion W_t. The Heston stochastic volatility model is given by*

$$\begin{aligned} dS_t &= (r-q)dt + \sqrt{v_t}dW_t \\ dv_t &= \kappa(\eta - v_t)dt + \theta\sqrt{v_t}d\tilde{W}_t \qquad v_0 = \sigma_0^2 \geq 0. \end{aligned} \tag{2.7}$$

Notice that we have included stochastic volatility in the model which was not the case in the BS model. For equity markets, ρ is typically negative since the volatility rises if the stock drops. Notice that this model does not include jumps, thus we discuss another model which does include jumps.

Before we introduce the Variance gamma model, we define the Variance Gamma distribution and state the definition of a Variance Gamma process.

Definition 3 (Variance Gamma distribution) *Suppose that the r.v. $X \sim Gamma(C, M)$ and $Y \sim Gamma(C, G)$ are independent. Then $\Gamma_1 \sim X - Y$ is Variance Gamma distributed i.e. $\Gamma_1 \sim VG(C, G, M)$.*

One can also define the Variance Gamma distribution by mixing a normal and a gamma distribution.

Definition 4 (Variance Gamma process) *A stochastic process $X = \{X_t, t \geq 0\}$ is a Variance-Gamma process with parameters (C, G, M) on some probability space $(\Omega, \mathcal{F}, \mathcal{P})$, if*

1. *$X_0 = 0$ a.s.*
2. *X has independent increments*
3. *X has stationary increments*
4. *$X_{t+v} - X_t \sim VG(Cv, G, M)$.*

Notice the strong similarity with a Brownian motion. However, a VG process is a jump process. In order to obtain a martingale (the expectation of the next value is equal to the current value), we shift the VG process and find the following model.

Definition 5 (Variance Gamma model) *Denote with $X = \{X_t, t \geq 0\}$ a Variance Gamma process with parameters (C, G, M) and write*

$$\begin{aligned} \nu &= 1/C \\ \sigma^2 &= 2C/(MG) \\ \theta &= C(G-M)/(MG) \\ \omega &= \nu^{-1}\log(1 - \frac{1}{2}\sigma^2\nu - \theta\nu). \end{aligned}$$

The Variance Gamma model of the stock price (risk-neutral) is given by

$$S_t = S_0 \exp\left((r - q + w)\, t + \sigma X_t\right). \tag{2.8}$$

Using these models one can price derivatives. For vanillas we use the Fast Fourier Transform which can be found in the appendix section 11.3. For barrier options we execute Monte Carlo simulations (see appendix section 11.4) and for American options a tree model (see appendix section 11.2).

Chapter 3

Some Mathematical Pre-knowledge

Our main goal is to train a model using the training dataset and to make (fast) predictions. However, if we do not make any assumptions, every function which is consistent with the training data would be equally valid. We can for example only use one class of functions (for example only linear functions). However, it is possible that this family is too simple and thus not sufficient. One also needs to pay attention for making the model to complex (overfitting).

We use an approach where we give a prior probability to every function. A higher probability is given to functions which we consider to be more likely, like smoother functions. In order to do so, one needs some knowledge about Bayesian - and functional analysis.

3.1 Bayesian Analysis

In this section, we follow (Rasmussen and Williams, 2006) and (Camilla, 2018) and focus on Bayesian model selection. At the lowest level, we consider $\boldsymbol{f}$ (vector of function values) and at the second level the hyperparameters $\boldsymbol{\theta}$ which control the distribution of $\boldsymbol{f}$. At the top level, we have a (discrete) set of possible model structures $\mathcal{H}_i$. We denote with X the covariates and with $\boldsymbol{y}$ the response. From Bayes theorem (see appendix equation (11.16) in the appendix), we find that the posterior of $\boldsymbol{f}$ at the lowest level is given by

$$p(\boldsymbol{f}|\boldsymbol{y}, X, \boldsymbol{\theta}, \mathcal{H}_i) = \frac{p(\boldsymbol{y}|X, \boldsymbol{f}, \mathcal{H}_i)p(\boldsymbol{f}|\boldsymbol{\theta}, \mathcal{H}_i)}{p(\boldsymbol{y}|X, \boldsymbol{\theta}, \mathcal{H}_i)}, \tag{3.1}$$

where $p(\boldsymbol{f}|\boldsymbol{\theta}, \mathcal{H}_i)$ is the prior distribution of $\boldsymbol{f}$, $p(\boldsymbol{y}|X, \boldsymbol{f}, \mathcal{H}_i)$ the likelihood function and $p(\boldsymbol{y}|X, \boldsymbol{\theta}, \mathcal{H}_i)$ the normalising constant. If we condition on $\boldsymbol{f}$, we find that

$$p(\boldsymbol{y}|X, \boldsymbol{\theta}, \mathcal{H}_i) = \int p(\boldsymbol{y}|X, \boldsymbol{f}, \mathcal{H}_i)p(\boldsymbol{f}|\boldsymbol{\theta}, \mathcal{H}_i)d\boldsymbol{f}. \tag{3.2}$$

Subsequently, the posterior distribution of the hyperparameters $\boldsymbol{\theta}$ is given by

$$p(\boldsymbol{\theta}|\boldsymbol{y}, X, \mathcal{H}_i) = \frac{p(\boldsymbol{y}|X, \boldsymbol{\theta}, \mathcal{H}_i)p(\boldsymbol{\theta}|\mathcal{H}_i)}{p(\boldsymbol{y}|X, \mathcal{H}_i)}, \tag{3.3}$$

where $p(\boldsymbol{\theta}|\mathcal{H}_i)$ is the prior distribution of $\boldsymbol{\theta}$, $p(\boldsymbol{y}|X, \boldsymbol{\theta}, \mathcal{H}_i)$ the likelihood function and $p(\boldsymbol{y}|X, \mathcal{H}_i)$ the normalising constant. If we condition on $\boldsymbol{\theta}$, we find that

$$p(\boldsymbol{y}|X, \mathcal{H}_i) = \int p(\boldsymbol{y}|X, \boldsymbol{\theta}, \mathcal{H}_i)p(\boldsymbol{\theta}|\mathcal{H}_i)d\boldsymbol{\theta}. \tag{3.4}$$

At the top level, the posterior of the model is calculated

$$p(\mathcal{H}_i|\boldsymbol{y}, X) = \frac{p(\boldsymbol{y}|X, \mathcal{H}_i)p(\mathcal{H}_i)}{p(\boldsymbol{y}|X)}. \tag{3.5}$$

Depending on the model, some of the integrals cannot be solved in an analytic way. Different ways to handle this problem can be found in the sections (4.1) and (4.3).

3.2 Functional Analysis

Although one can perfectly understand the machine learning techniques from a Bayesian or frequentist point of view, it is very interesting to approach our problems from another perspective. We briefly discuss some important items found in (Rasmussen and Williams, 2006) and (Sejdinovic and Gretton, 2012). For the Representer theorem, we follow (Schölkopf et al., 2001).

Definition 6 (Banach space) *A Banach space is a complete normed space.*

Definition 7 (Hilbert space) *A Hilbert space is a complete inner product space. In other words, it is a Banach space with inner product.*

Definition 8 (Functional) *Let B be a Banach space. Linear maps from B to $\mathbb{C}$ are called functionals.*

Definition 9 (Evaluation Functional) *Let $\mathcal{H}$ be a Hilbert space of functions $\mathcal{X} \to \mathbb{R}$, defined on a non-empty set $\mathcal{X}$. For a fixed $\boldsymbol{x} \in \mathcal{X}$, the map $\delta_{\boldsymbol{x}} : \mathcal{H} \to \mathbb{R} : f \mapsto f(\boldsymbol{x})$ is called the evaluation functional at $\boldsymbol{x}$.*

Notice that evaluation functionals are always linear. However, they are not always continuous thus we define the motion of reproducing kernel Hilbert space (RKHS), which behaves particularly well.

Definition 10 (Reproducing kernel Hilbert space) *A Hilbert space $\mathcal{H}$ of functions $\mathcal{X} \to \mathbb{R}$, defined on a non-empty set $\mathcal{X}$ is said to be a reproducing kernel Hilbert space (RKHS) if $\delta_{\boldsymbol{x}}$ is continuous $\forall \boldsymbol{x} \in \mathcal{X}$.*

Before we discuss what a RKHS have to do with kernels and machine learning, we give a rigorous definition of a kernel and a reproducing kernel.

Definition 11 (Kernel) *Let $\mathcal{X}$ be a non-empty set. The function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is said to be a kernel if there exists a real Hilbert space $\mathcal{H}$ and a map $\phi : \mathcal{X} \to \mathcal{H}$ such that $\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$,*

$$k(\boldsymbol{x}, \boldsymbol{x}') := \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle_{\mathcal{H}}. \tag{3.6}$$

The map ϕ is called a feature map to the feature space $\mathcal{H}$.

Definition 12 (Reproducing kernel) *Let $\mathcal{H}$ be a Hilbert space of $\mathbb{R}$-valued functions defined on a non-empty set $\mathcal{X}$. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a reproducing kernel of $\mathcal{H}$ if it satisfies*

- $\forall \boldsymbol{x} \in \mathcal{X}, k(\cdot, \boldsymbol{x}) \in \mathcal{X}$
- $\forall \boldsymbol{x} \in \mathcal{X}, \forall f \in \mathcal{H}, \langle f, k(\cdot, \boldsymbol{x}) \rangle_{\mathcal{H}} := f(\boldsymbol{x})$ *(the reproducing property).*

Notice that all reproducing kernels are kernels (take $\phi(\boldsymbol{x}) = k(\cdot, \boldsymbol{x})$). One can also prove that kernels, and thus Reproducing kernels, are positive definite. Remember that every inner product is a positive definite function. We state two theorems from which we omit the doable proofs.

Theorem 1 (Uniqueness of the reproducing kernel) *Let $\mathcal{H}$ be a Hilbert space of $\mathbb{R}$-valued functions defined on a non-empty set $\mathcal{X}$. If a the reproducing kernel exists, it is unique.*

Theorem 2 (Existence of the reproducing kernel) *$\mathcal{H}$ is an RKHS if and only if $\mathcal{H}$ has a reproducing kernel.*

We denote the space of $\mathbb{R}$-valued functions defined on a non-empty set $\mathcal{X}$ with $\mathbb{R}^{\mathcal{X}}$. Now we are ready to state the famous theorem of Moore-Aronszajn.

Theorem 3 (Moore-Aronszajn theorem) *Let $\mathcal{X}$ be the non-empty input domain. Then for every positive definite function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists a unique RKHS $\mathcal{H} \subset \mathbb{R}^{\mathcal{X}}$ of $\mathbb{R}$-valued functions defined on a non-empty set $\mathcal{X}$ with reproducing kernel k.*

Hence, from Moore-Aronszajn, we find that every positive definite function is a reproducing kernel. We already knew that every kernel is positive definite and that every reproducing kernel is a kernel. Hence, these three notions are equivalent.

From the fact that kernels are positive definite functions, we find the following theorem

Theorem 4 (Sum and scaling of kernels) *If k, k_1 and k_2 are kernels and $\alpha > 0$ is a scalar, then αk, $k_1 + k_2$ are kernels.*

Notice that the difference of kernels is not necessarily a kernel since it is not allowed to have that $k_1(\boldsymbol{x}, \boldsymbol{x}) - k_2(\boldsymbol{x}, \boldsymbol{x}) < 0$. In this case, for the feature map $\phi : \mathcal{X} \to \mathcal{H}$, the inproduct $\langle \phi(x), \phi(\boldsymbol{x}) \rangle_{\mathcal{H}}$ is smaller than zero which is not possible. For the product of kernels we find the following

Theorem 5 (Product of kernels) *Let k_1 and k_2 be kernels on $\mathcal{X}$ and $\mathcal{Z}$ respectively. We have that*

$$k((\boldsymbol{x}, \boldsymbol{z}), (\boldsymbol{x}', \boldsymbol{z}')) := k_1(\boldsymbol{x}, \boldsymbol{x}') k_2(\boldsymbol{z}, \boldsymbol{z}') \tag{3.7}$$

is a kernel on $\mathcal{X} \times \mathcal{Y}$. This type of kernel is called a product kernel. In addition, if k_1 and k_2 are both kernels on $\mathcal{X}$, we find that

$$k(\boldsymbol{x}, \boldsymbol{x}') := k_1(\boldsymbol{x}, \boldsymbol{x}')k_2(\boldsymbol{x}, \boldsymbol{x}') \tag{3.8}$$

is a kernel on $\mathcal{X}$.

Hence, from the linear kernel $k_{lin}(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$ and the previous two theorems we can construct polynomial kernels $k_{\text{poly}}(\boldsymbol{x}, \boldsymbol{x}') = (\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + c)^m$, with $c > 0$. Using Taylor series with non-negative coefficients, we can construct the exponential kernel $k_{\exp}(\boldsymbol{x}, \boldsymbol{x}') = \exp(2\sigma\langle \boldsymbol{x}, \boldsymbol{x}' \rangle)$ with $\sigma > 0$. Due to its importance we derive how to construct the Gaussian kernel. Let $\phi : \mathbb{R}^d \to \mathbb{R} : \boldsymbol{x} \mapsto \exp(-\sigma||\boldsymbol{x}||^2)$. If we define the ordinary inner product on $\mathbb{R}$, we know that $\mathbb{R}$ is a Hilbert space. Subsequently, using definition (11), we find that $\hat{k}(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})\phi(\boldsymbol{x}') = \exp(-\sigma||\boldsymbol{x}||^2)\exp(-\sigma||\boldsymbol{x}'||^2)$ is a kernel on $\mathbb{R}^d$. Hence, using Theorem (5), we find the Gaussian kernel on $\mathbb{R}^d$ given by

$$\begin{aligned} k_{gauss}(\boldsymbol{x}, \boldsymbol{x}') &:= \hat{k}(\boldsymbol{x}, \boldsymbol{x}')k_{\exp}(\boldsymbol{x}, \boldsymbol{x}') \\ &= \exp\left(-\sigma(||\boldsymbol{x}||^2 + ||\boldsymbol{x}'||^2 - 2\langle \boldsymbol{x}, \boldsymbol{x}' \rangle)\right) \\ &= \exp\left(-\sigma||\boldsymbol{x} - \boldsymbol{x}'||^2\right). \end{aligned} \tag{3.9}$$

Of course, we can classify our kernels. A stationary kernel is a function of $\boldsymbol{x} - \boldsymbol{x}'$, which is invariant to translations in the input space. Even stronger, if the kernel is only a function of $||\boldsymbol{x} - \boldsymbol{x}'||$, it is called isotropic and thus invariant to all rigid motions. An example is k_{gauss}. Note that the Gaussian kernel on $\mathbb{R}^d$ is a product kernel since it can be written as a product of d Gaussian kernels defined on a scalar input space.

Let us now consider one of the most important theorems of this section called the Representer theorem, which generalises a large class of optimisation problems.

Theorem 6 (Representer theorem) *Suppose we are given a nonempty set $\mathcal{X}$, a positive definite real valued kernel k on $\mathcal{X} \times \mathcal{X}$, a training set $(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_n, y_n) \in \mathcal{X} \times \mathbb{R}$, a strictly monotonically increasing real-valued function g on $[0, \infty[$, an arbitrary cost function $c : (\mathcal{X} \times \mathbb{R}^2)^n \to \mathbb{R} \cup \{\infty\}$ and a class of functions*

$$\mathcal{F} = \left\{ f \in \mathbb{R}^{\mathcal{X}} | f(\cdot) = \sum_{i=1}^{\infty} \beta_i k(\cdot, \boldsymbol{z}_i), \beta_i \in \mathbb{R}, \boldsymbol{z}_i \in \mathcal{X}, ||f|| < \infty \right\}. \tag{3.10}$$

Here, $||\cdot||_{\mathcal{H}}$ is the norm in the RKHS $\mathcal{H}_k$ associated with kernel k, i.e. for any $\boldsymbol{z}_i \in \mathcal{X}, \beta_i \in \mathbb{R} (i \in \mathbb{N})$ we have that

$$\left|\left| \sum_{i=1}^{\infty} \beta_i k(\cdot, \boldsymbol{z}_i) \right|\right|_{\mathcal{H}}^2 = \sum_{i,j=1}^{\infty} \beta_i \beta_j k(\boldsymbol{z}_i, \boldsymbol{z}_j). \tag{3.11}$$

Then, any $f \in \mathcal{F}$ minimizing the regularized risk functional

$$c((\boldsymbol{x}_1, y_1, f(\boldsymbol{x}_1)), \cdots, (\boldsymbol{x}_n, y_m, f(\boldsymbol{x}_n))) + g(||f||), \tag{3.12}$$

admits a representation of the form

$$f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\cdot, \boldsymbol{x}_i). \tag{3.13}$$

The first term of equation (3.12) is called the risk and is used for fitting the data. The second term is called the regularizer and serves as smoothness assumptions on f. We continue this section by introducing some notation and discussing a very handy trick.

Definition 13 (Gram matrix) *Given a kernel k and $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \in \mathcal{X}$, the $n \times n$ matrix K for which $K(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ (with $1 \leq i, j \leq n$) is called the Gram matrix of k with respect to $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.*

A Gram matrix corresponding to a kernel function as defined above is positive semi-definite. In order to obtain a non-linear version of an algorithm, one often replace $\boldsymbol{x}$ by $\phi(\boldsymbol{x})$. Thus we actually execute a mapping in a feature space. In other words, for $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{X}$, $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ is replaced by $\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = K(i,j)$. This called the Kernel Trick. From a computational point of view, by replacing $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ with $K(i,j)$, we turn a linear procedure into a non-linear procedure without adding much computation.

Now we discuss Mercer's theorem which allows us to express a kernel k in terms of eigenvalues and eigenfunctions. We denote with T_k an integral operator defined as

$$T_k(f) := \int k(\boldsymbol{x}, \boldsymbol{x}') f(\boldsymbol{x}') d\mu(\boldsymbol{x}') \tag{3.14}$$

with μ a measure. A function φ satisfying

$$\int k(\boldsymbol{x}, \boldsymbol{x}') \varphi(\boldsymbol{x}) d\mu(\boldsymbol{x}) = \lambda \varphi(\boldsymbol{x}') \tag{3.15}$$

is an eigenfunction of kernel k with eigenvalue λ with respect to the measure μ. Now we are ready to state the following theorem.

Theorem 7 (Mercer's theorem) *Let $(\mathcal{X}, \mu)$ be a finite measure space and $k \in L_\infty(\mathcal{X}^2, \mu^2)$ be a kernel such that $T_k : L_2(\mathcal{X}, \mu) \to L_2(\mathcal{X}, \mu)$ is positive definite. Let $\varphi_i \in L_2(\mathcal{X}, \mu)$ be the normalised eigenfunctions of T_k associated with the eigenvalues $\lambda_i > 0$. We then have that*

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{\infty} \lambda_i \varphi_i(\boldsymbol{x}) \varphi_i(\boldsymbol{x}') \tag{3.16}$$

holds μ^2 almost everywhere where the series converges absolutely and uniformly μ^2 almost everywhere.

Next, we try to numerically approximate the eigenfunctions. Let $d\mu(\boldsymbol{x}) = p(\boldsymbol{x}) d(\boldsymbol{x})$ and denote with $\boldsymbol{x}_l$ a sample from $p(\boldsymbol{x})$. Hence we can use the approximation

$$\lambda_i \varphi_i(\boldsymbol{x}') = \int k(\boldsymbol{x}, \boldsymbol{x}') p(\boldsymbol{x}) \varphi_i(\boldsymbol{x}) d\boldsymbol{x} \approx \frac{1}{n} \sum_{l=1}^{n} k(\boldsymbol{x}_l, \boldsymbol{x}') \varphi_i(\boldsymbol{x}_l). \tag{3.17}$$

Now plugging in $\boldsymbol{x}' = \boldsymbol{x}_l$ for $l = 1, \ldots, n$, denoting with K the Gram matrix and with λ_i^{mat} the ith matrix eigenvalue, we obtain the following matrix eigenproblem

$$K \boldsymbol{v}_i = \lambda_i^{\text{mat}} \boldsymbol{v}_i. \tag{3.18}$$

For fixed i, it is well-known that $\frac{1}{n} \lambda_i^{\text{mat}} \to \lambda_i$ as $n \to \infty$. Hence, we see that $\varphi_i(\boldsymbol{x}_j) = \sqrt{n} (\boldsymbol{v}_i)_j$. Plugging this in the right-hand side of equation (3.17) and dividing both sides by λ_i, we obtain the approximation

$$\varphi_i(\boldsymbol{x}') \approx \frac{\sqrt{n}}{\lambda_i^{\text{mat}}} (k(\boldsymbol{x}_1, \boldsymbol{x}'), \ldots, k(\boldsymbol{x}_n, \boldsymbol{x}')) \boldsymbol{v}_i. \tag{3.19}$$

Chapter 4

Gaussian Process Regression

In the first two sections of this chapter, we try to convince ourself why Gaussian process regression (GPR) makes sense by looking at it in different ways. Next, we discuss how we can find the hyperparameters of the kernel and we finish by discussing the algorithm of standard GPR.

4.1 Bayesian View

Let us introduce some notation. Suppose we have a training set of n observations $\boldsymbol{x}_i$ with $i \in (1, \ldots, n)$ having dimension d. Hence, the design matrix X containing the covariates, has dimension $n \times d$. The response vector is denoted by $\boldsymbol{y}$. For the prediction set we use the $n^* \times d$ matrix X^* containing the observations $\boldsymbol{x}_i^*$ with $i \in (1, \ldots, n^*)$.

In this section, we will follow (Rasmussen and Williams, 2006). Consider the following non-parametric regression model

$$Y = f(X) + \boldsymbol{\varepsilon}. \tag{4.1}$$

Now, considering a linear model, we take $f(\boldsymbol{x}_i) = \boldsymbol{x}_i \boldsymbol{w}$ with $\boldsymbol{w}$ the weights. We assume that the noise $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2 I_n)$. In order to do Bayesian analysis, one requires a prior over the parameters. We choose the Gaussian prior $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \Sigma)$ over the weights. From Bayes theorem, and noting that $p(\boldsymbol{w}|X)$ is independent of X, we find that

$$p(\boldsymbol{w}|\boldsymbol{y}, X) = \frac{p(\boldsymbol{y}|X, \boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{y}|X)}. \tag{4.2}$$

The denominator is the marginal likelihood

$$p(\boldsymbol{y}|X) = \int p(\boldsymbol{y}|X, w)p(\boldsymbol{w})d\boldsymbol{w} \tag{4.3}$$

which is just a normalising constant in equation (4.2). Notice the analogy with equations (3.1) and (3.2) without the hyperparameters since they are not present at this moment. Due to the fact that the noise is Gaussian, we find that the likelihood is given by

$$p(\boldsymbol{y}|X,\boldsymbol{w}) = \prod_{i=1}^{n} p(y_i|X,\boldsymbol{w}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(y_i - \boldsymbol{x}_i\boldsymbol{w})^2}{2\sigma^2}\right)$$
$$= \frac{1}{(2\pi\sigma)^{\frac{n}{2}}} \exp\left(\frac{1}{2\sigma^2}(\boldsymbol{y} - X\boldsymbol{w})^2\right) = \mathcal{N}(X\boldsymbol{w}, \sigma^2 I_n). \tag{4.4}$$

If we define with $\bar{\boldsymbol{w}} = \sigma^{-2}(\sigma^{-2}X^tX + \Sigma^{-1})$, we find that the posterior is proportional to

$$p(\boldsymbol{w}|\boldsymbol{y},X) \propto \exp\left(-\frac{1}{2\sigma^2}(\boldsymbol{y} - Xw)(\boldsymbol{y} - Xw)\right) \exp\left(-\frac{1}{2}\boldsymbol{w}^t\Sigma^{-1}\boldsymbol{w}\right)$$
$$\propto \exp\left(-\frac{1}{2}(\boldsymbol{w} - \bar{\boldsymbol{w}})^t\left(\frac{1}{\sigma^2}X^tX + \Sigma^{-1}\right)(\boldsymbol{w} - \bar{\boldsymbol{w}})\right), \tag{4.5}$$

due to the fact that the denominator is constant in equation (4.2) and by "completing the square". Hence, denoting with $A = (\sigma^{-2}X^tX + \Sigma^{-1})$, we find that the posterior has the following distribution

$$p(\boldsymbol{w}|\boldsymbol{y},X) \sim \mathcal{N}(\bar{\boldsymbol{w}} = \frac{1}{\sigma^2}A^{-1}X^t\boldsymbol{y}, A^{-1}). \tag{4.6}$$

The predictive distribution at $\boldsymbol{x}^*$ can be seen as averaging the output of all linear models with respect to the Gaussian posterior. Hence, we find that

$$p(f(\boldsymbol{x}^*)|\boldsymbol{x}^*,X,\boldsymbol{y}) = \int p(f(\boldsymbol{x}^*)|\boldsymbol{x}^*,\boldsymbol{w},X,\boldsymbol{y})p(\boldsymbol{w}|X,\boldsymbol{y})d\boldsymbol{w}$$
$$= \mathcal{N}\left(\frac{1}{\sigma^2}\left(\boldsymbol{x}^*\right)^t A^{-1}X\boldsymbol{y}, \left(\boldsymbol{x}^*\right)^t A^{-1}\boldsymbol{x}^*\right). \tag{4.7}$$

Since $f(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{w}$, the predictive distribution at $\boldsymbol{x}^*$ is given by $p(f(\boldsymbol{x}^*)|x^*,X,\boldsymbol{y}) = \boldsymbol{x}^*p(\boldsymbol{w}|X,\boldsymbol{y})$ from which expression (4.7) can easily derived using lemma (6) in the appendix.

Let us now map the d-dimensional input vector to a D dimensional feature space. Thus the vector of parameters has now dimension D. We use the same notation as in section 3.2. Hence our model is now given by $f(\boldsymbol{x}) = \phi(\boldsymbol{x})\boldsymbol{w}$. In other words, we can replace $\boldsymbol{x}^*$ by $\phi(\boldsymbol{x}^*)$ and X by $\Phi(X)$, where the latter is the matrix consisting of $\phi(\boldsymbol{x})$. Notice that we now need to invert the matrix A of size $D \times D$ which can be time consuming. Furthermore, we use the matrix inversion lemma (see appendix section 11.6). We obtain that

$$f^*|\boldsymbol{x}^*,X,\boldsymbol{y} \sim \mathcal{N}(\phi\left(\boldsymbol{x}^*\right)\Sigma\Phi(X)^t(\Phi(X)\Sigma\Phi(X)^t + \sigma^2 I_n)^{-1}\boldsymbol{y},$$
$$(\phi(\boldsymbol{x}^*)\Sigma\phi\left(\boldsymbol{x}^*\right)^t(\Phi(X)\Sigma\Phi(X)^t + \sigma^2 I_n)^{-1}\Phi(X)\Sigma\phi\left(\boldsymbol{x}^*\right)^t). \tag{4.8}$$

Notice that $\phi(\boldsymbol{x})\Sigma\phi(\boldsymbol{x}')^t$ is an inner product with respect to Σ. If Σ is positive definite and thus $\Sigma^{1/2}$ exists, we can define $\psi(\boldsymbol{x}) = \phi(\boldsymbol{x})\Sigma^{1/2}$. Hence, we obtain the following dot product $k(\boldsymbol{x}, \boldsymbol{x}') = \psi(\boldsymbol{x}) \cdot \psi(\boldsymbol{x})^t$. This enables us to execute the kernel trick and write $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})\Sigma\phi(\boldsymbol{x}')^t$. We denote with $K(X, X')$ the Gram matrix consisting of elements $k(\boldsymbol{x}, \boldsymbol{x}')$. Remark that some hyperparameters concerning the kernel come into play. Notice that we now need to invert a matrix of size $n \times n$ which can come at handy when $n < D$. Finally, we form a matrix $\Phi(X^*)$ consisting of $\phi(\boldsymbol{x}^*)$ and write $\boldsymbol{f}^*$ for the predictive distribution

$$\boxed{\begin{aligned}\boldsymbol{f}^*|X^*, X, \boldsymbol{y} \sim \mathcal{N}(&K(X^*, X)[K(X, X) + \sigma^2 I]^{-1}\boldsymbol{y}, \\ &K(X^*, X^*) - K(X^*, X)[K(X, X) + \sigma^2 I]^{-1}K(X, X^*)).\end{aligned}} \tag{4.9}$$

This is actually the weight space view of the predictive version equation (3.1).

4.2 Function Space View

In this section, we follow (Rasmussen and Williams, 2006) and (De Spiegeleer et al., 2018). We try to obtain the same results as before, though from another point of view. We consider inference directly in the function space.

A Gaussian process (GP), is just a collection of (possible infinite) random variables where any finite subset of it has a joint Gaussian distribution. We use the same notation as in formula (4.1) with $f(\boldsymbol{x})$ a Gaussian process and $\boldsymbol{\varepsilon}$ is the noise in the data with mean zero and variance $\sigma^2 I_n$. The process $f(\boldsymbol{x})$ is completely determined by its mean and covariance function. We define the mean function of $f(\boldsymbol{x})$ as $m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})]$ and the covariance as $k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}))]$. Notice that the kernel has to be symmetric. We have that

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')). \tag{4.10}$$

Notice that GPR is a Bayesian method, with a Gaussian process prior. As discussed before, the prior takes care of the fact that we prefer smoother functions. Combining this with the data, it is essentially a matter of conditioning a joint distribution on the observations. One often assumes a zero mean function. We again denote with $K(X, X')$ the Gram matrix consisting of elements $k(\boldsymbol{x}, \boldsymbol{x}')$ and find that

$$\begin{bmatrix}\boldsymbol{y} \\ \boldsymbol{f}^*\end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix}K(X, X) + \sigma^2 I_n & K(X, X^*) \\ K(X^*, X) & K(X^*, X^*)\end{bmatrix}\right). \tag{4.11}$$

Notice that if $\sigma^2 = 0$, we have clean observations of the underlying process and we see that $\boldsymbol{y} = \boldsymbol{f}$. Conditioning on the joint distribution on the observations (see appendix section 11.7), we find the following predictive equations

$$\boxed{\begin{aligned}\boldsymbol{f}^*|X^*, X, \boldsymbol{y} \sim \mathcal{N}(&K(X^*, X)[K(X, X) + \sigma^2 I]^{-1}\boldsymbol{y}, \\ &K(X^*, X^*) - K(X^*, X)[K(X, X) + \sigma^2 I]^{-1}K(X, X^*),\end{aligned}} \tag{4.12}$$

which is the GP posterior in the test inputs. Notice that this is the same formula as before and thus the function space view of the predictive version of equation (3.1).

We already obtained the solution using two different methods. However, we can obtain even more insight using the Representer theorem (see theorem (6)). Let us consider the following functional

$$\mathcal{J}(f) = \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - f(\boldsymbol{x}_i))^2 + \frac{1}{2}||f||^2_{\mathcal{H}}. \tag{4.13}$$

Notice that the negative log likelihood of the Gaussian noise model with variance σ, which can be found in equation (4.4), uses the same squared error data fit term. Now, substituting $f(\cdot) = \sum_{i=1}^{n}\alpha_i k(\cdot, \boldsymbol{x}_i)$ and using $\langle k(\cdot, \boldsymbol{x}_i), k(\cdot, \boldsymbol{x}_j)\rangle_{\mathcal{H}} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, we find that

$$\begin{aligned} J(\boldsymbol{\alpha}) &= \frac{1}{2}\boldsymbol{\alpha}^t K\boldsymbol{\alpha} + \frac{1}{2\sigma^2}|\boldsymbol{y} - K(X,X)\boldsymbol{\alpha}|^2 \\ &= \frac{1}{2}\boldsymbol{\alpha}^t\left(K(X,X) + \frac{1}{\sigma^2}(K(X,X))^2\right)\boldsymbol{\alpha} - \frac{1}{\sigma^2}\boldsymbol{y}^t K(X,X)\boldsymbol{\alpha} + \frac{1}{2\sigma^2}\boldsymbol{y}^t\boldsymbol{y}. \end{aligned} \tag{4.14}$$

If we differentiate $J(\boldsymbol{\alpha})$ to $\boldsymbol{\alpha}$ and set this derivative to zero we obtain a minimum (due to the convexity) which is given by $\hat{\boldsymbol{\alpha}} = (K + \sigma^2 I)^{-1}\boldsymbol{y}$. Hence, we find that

$$\boxed{f(\boldsymbol{x}^*) = k(\boldsymbol{x}^*, X)(K(X,X) + \sigma^2 I)^{-1}\boldsymbol{y}}, \tag{4.15}$$

which is again the same equation. Be aware of the strong similarity between GPR and kernel ridge regression. If we choose the hyperparameters in an appropriate way and if we use the same kernel, kernel ridge regression and GPR are equal. However, for GPR, we decide to find the hyperparameters by maximizing the marginal likelihood were for kernel ridge regression one always uses a variation of cross-validation. More information concerning the similarities can be found in (Welling, 2013).

4.3 Hyperparameter Selection

In this section, we follow (Rasmussen and Williams, 2006) and (Camilla, 2018).

For determining the (kernel) hyperparameter(s), one proposes to use cross-validation, maximise the marginal likelihood or sample them. Since the computational burden of cross-validation is greater than maximising the marginal likelihood, we discard the first. Sampling the hyperparameters is possible although this can also be very computationally demanding. These Bayesian techniques are quite popular and a clear explanation concerning these methods can be found in chapter (8). In the Bayesian case, there is also a prior over the hyperparameters like in equation (3.3). Since the integral in (3.4) can be not analytically tractable, one could require analytical approximations or Bayesian methods.

For the other approach, we maximise the marginal likelihood, which is also called the empirical Bayesian approach. As seen in section 4.2, we put a prior on $\boldsymbol{f}$ which equals $\boldsymbol{f}|X \sim \mathcal{N}(\mathbf{0}, K(X,X))$. However, we do not give a prior distribution on the hyperparameters in this case. In other words, we omit the difficult integral (3.4) and actually do kind of an approximation (known as type II-likelihood or evidence).

From equation (4.4), we find that the likelihood $\boldsymbol{y}|\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{f}, \sigma^2 I_n)$. Hence, by marginalization over $\boldsymbol{f}$ (that's why it is called the marginal likelihood), we find that

$$p(\boldsymbol{y}|X) = \int p(\boldsymbol{y}|\boldsymbol{f}, X)p(\boldsymbol{f}|X)d\boldsymbol{f}. \tag{4.16}$$

This equals equation (3.2) with the hyperparameters fixed. Often, these integrals are analyticaly intractable. However, we are lucky since for Gaussian process regression with white noise, this integral can be solved analytically. Solving this equation using the lemma (9) in the appendix, one finds equation

$$\log p(\boldsymbol{y}|X) = -\frac{1}{2}\boldsymbol{y}^t(K(X,X)+\sigma^2 I)^{-1}\boldsymbol{y} - \frac{1}{2}\log|K(X,X)+\sigma^2 I_n| - \frac{n}{2}\log(2\pi). \tag{4.17}$$

Alternatively, by observing that $\boldsymbol{y} \sim \mathcal{N}(\mathbf{0}, K(X,X)+\sigma I_n)$, we find again equation (4.17). By maximizing this equation, we can find the (kernel) hyperparameters.

4.4 The Algorithm

In this section, we follow (Rasmussen and Williams, 2006), (De Spiegeleer et al., 2018) and (Chen and Wang, 2018). We actually need to do three delicate things. First we can decide to take care of the zero mean function, next we determine the hyperparameter(s) of the kernel and afterwards we calculate equation (4.15).

In order to take care of the zero mean function, one can first execute a polynomial fit of low degree and afterwards we model the residuals using a Gaussian process. Another approach is to include an extra mean function in our model. However, we need to optimise more hyperparameters in this case, which slows down our calculations.

As discussed before, we can find the hyperparameters by maximizing equation (4.17) (the marginal likelihood or evidence). We normally use the L-BFGS method, which belongs to quasi-Newton methods (see appendix section 11.8). However, for many kernels, the equation is not convex with respect to the hyperparameters. Therefore, the optimisation algorithm may converge to a local optimum. In order to avoid this, one can generate multiple random initial values from a simple prior distribution (for example uniform) and do the corresponding optimisation on them. Notice that one needs to do the optimisation multiple times using this approach, which is very computational demanding. Other (Bayesian) approaches converging to a global optimum like simulated annealing are discussed in chapter 8.

In our algorithm, we will use the Gaussian kernel (which is also called the squared exponential kernel). Notice that we already encountered this kernel in section 3.2.

However, we include one extra hyperparameter σ_f (the signal variance) and together with the length-scale l we find that

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left(\frac{-\sum_{i=1}^{d} |x_i - x_i'|^p}{2l^2}\right). \tag{4.18}$$

The higher the length-scale the less the model varies across the parameters. We denote with $\boldsymbol{\theta} = (\sigma_f, l)$ the vector containing the hyperparameters. Other kernels are also possible, but we do not dwell on it any further in this thesis.

Notice that we need to calculate inverses in equations (4.17) and in (4.15). In our algorithm, we use the Cholesky decomposition. More information about this can be found in the appendix section 11.9. From the properties of the Cholesky decomposition the determinant in equation (4.17) can be calculated easily. This algorithm is $O(n^3)$ in runtime (due to the Cholesky decomposition) and $O(n^2)$ in memory (matrix storage). The pseudo-code can be found in algorithm 1. We use the notation $x = A \setminus b$ for the solution of $AX = b$ and we only focus on calculating the predictive mean. Notice that the value of the kernel depends on $\boldsymbol{\theta}$. For small σ^2 it is possible that the matrix $(K(X, X^*) + \sigma^2 I)$ is not invertible. In this case, it is recommended to gradually increase σ^2. Actually, the parameter σ pushes the eigenvalues of the positive definite Gram matrix away from zero, making it invertible. In our basic standard GPR algorithm, we take σ^2 as input parameter, however this parameter can also be optimised and automatically increased if necessary.

Algorithm 1 Standard GPR

input: $X, \boldsymbol{y}, k, \sigma^2, X^*, \boldsymbol{\theta}_0$ (initial hyperparameter values)
output: $\mathbb{E}[\boldsymbol{f}^*]$

Optimise *LogMarginalLlh*($\boldsymbol{\theta}$) :
Calculate $K(X, X)$
$L = cholesky(K(X, X) + \sigma^2 I_n)$
$\boldsymbol{\alpha} = L^t \setminus L \setminus \boldsymbol{y}$
$\mathbb{E}[\boldsymbol{f}^*] = K(X^*, X)\boldsymbol{\alpha}$

function: *LogMarginalLlh*($\boldsymbol{\theta}$)
    $L = cholesky(K(X, X) + \sigma^2 I_n)$
    $\boldsymbol{\alpha} = L^t \setminus L \setminus \boldsymbol{y}$
    $\log(p(\boldsymbol{y}|X)) = -\frac{1}{2}\boldsymbol{y}^t\boldsymbol{\alpha} - \sum_i \log(L_{ii}) - \frac{n}{2}\log(2\pi)$
    return: $\log(p(\boldsymbol{y}|X))$.

Theorem 8 (Computation cost prediction exact GPR) *The computation cost to find the predictive mean for one observation is $O(n)$.*

Proof: The vector $(K(X, X) + \sigma^2 I)^{-1}\boldsymbol{y}$ has been calculated during training. Hence, the construction and vector multiplication with $k(x^*, X)$ takes $O(n)$ time. □

Chapter 5

Exploiting The Structure

Notice that we generate our dataset ourself in order to train option values, thus it makes sense that we exploit this insight. For particular datasets, we are able to obtain a tremendous speed-up. Nevertheless, the methods we discuss are often not applicable since we obtain constraints on the dimension of the covariates.

5.1 Kronecker Gaussian Process Regression

In this paragraph, we mainly follow (Saatçi, 2012) combined with some thoughts from (Wilson and Nickisch, 2015). We generate data which lies on a rectilinear grid (see figure (5.1)). We need the Kronecker product from which extra information and properties can be found in the appendix section 11.10.

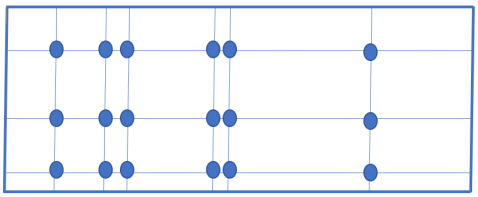


Figure 5.1: A 2-dimensional rectilinear grid.

In order to use the handy properties of the Kronecker product, we work with a product kernel consisting of the product of the kernels k_c (see equation (3.7)) over scalar inputs. We define K the $n \times n$ Gram matrix over X and K_c the $e_c \times e_c$ matrix defined over the vector of scalar inputs $\boldsymbol{x}^{(c)}$. Notice that $n = \prod_{c=1}^{d} e_c$. Imposing the same constraints concerning the indices as in definition (16) in the appendix, we find that

$$K(i,j) = K_1(i^{(1)}, j^{(1)})K_2(i^{(2)}, j^{(2)}) \cdots K_d(i^{(d)}, j^{(d)}). \tag{5.1}$$

In other words, we have that $K = \bigotimes_{c=1}^{d} K_c$. First let us try to determine the hyperparameters. We again try to optimise the marginal likelihood (4.17). If the noise is zero, the quantity $K^{-1}\boldsymbol{y}$ can be found using the Kronecker product. Hence, we only need to compute the inverses of the smaller matrices K_c and execute the corresponding

Kronecker product. Calculating the inverses of the smaller matrices has time complexity $\mathrm{O}(d \cdot \max_{c\in\{1,\cdots,d\}} e_c^3)$ and the Kronecker product which is, using the standard matrix-vector multiplication, $O(n^2)$ in runtime and memory. If we calculate the inverses with a Cholesky decomposition, one can again easily compute the determinant of the smaller matrices. Using the corresponding property of determinants of Kronecker products, one can also compute the determinant in a very efficient way.

However, a small detour has to be made if the noise σ is not equal to zero. Despite we can write K^{-1} as a Kronecker product, this is not necessarily the case for $(K+\sigma^2 I)^{-1}$. Luckily, using the eigendecompositions $K = Q\Lambda Q^t$ of K and $K_c = Q_c \Lambda_c Q_c^t$ of K_c, we find that

$$\begin{aligned}(K+\sigma I)^{-1}\boldsymbol{y} &= Q(\Lambda+\sigma^2 I)^{-1}Q^t\boldsymbol{y} \\ &= \left(\bigotimes_{c=1}^{d} Q_c\right)\left(\left(\bigotimes_{c=1}^{d}\Lambda_c\right)+\sigma^2 I\right)^{-1}\left(\bigotimes_{c=1}^{d} Q_c^t\right)\boldsymbol{y}. \end{aligned} \tag{5.2}$$

Notice that calculating the inverse of is not computationally demanding any more since it is a diagonal matrix. Of course, one needs to calculate the eigenvalues and eigenvectors of the smaller matrices K_c which has time complexity $\mathrm{O}(d \cdot \max_{c\in\{1,\cdots,d\}} e_c^3)$ and the Kronecker product. In (Saatçi, 2012), one proposes a different approach using tensors for calculating matrix vector products. The necessary basic knowledge of tensors can be found in chapter 26 of (Riley et al., 2006). This was further clarified in the appendix of (Wilson, 2014). One found that, supposing the d Kronecker matrices have the same dimension $e = n^{1/d}$, the matrix vector product

$$K\boldsymbol{y} = \bigotimes_{c=1}^{d} K_c\boldsymbol{y} \tag{5.3}$$

can be calculated in $O\left(d \cdot n^{\frac{d+1}{d}}\right)$ in runtime and $O(dn^{\frac{2}{d}})$ in storage. For the determinant we find that

$$|K+\sigma I| = |Q(\Lambda+\sigma^2 I)Q^t| = |\Lambda+\sigma^2 I| = \prod_{i=1}^{n}(\Lambda_{ii}+\sigma^2), \tag{5.4}$$

which is not computationally demanding since we already calculated the eigendecomposition. However, note that computational and memory demands scale in dimension, not to mention about the curse of dimensionality. For example, using a dataset with dimension $d = 10$ and we only want 3 different values for each dimension. We have only little information for each dimension although we need $3^{10} = 59049$ data points. Hence, it is not recommended using this technique for datasets with $d > 5$. Thus, the running time (and feasible accuracy) is very dependent on the dimension, what was not the case before.

5.2 Toeplitz Gaussian Process Regression

Notice that for Kronecker GPR, there is no possible efficiency gain for one-dimensional inputs. However, for one-dimensional inputs, one can impose a Toeplitz structure on the Gram matrix. This seems not relevant for our subject, since we work with options depending on a lot of variables. Nevertheless, it will be useful in the future. An algorithm about fast inverting a Toeplitz matrix can for example be found in(Wilson, 2014). We are able to convert a Toeplitz matrix in $O(n \log(n))$ time. In fact we are able to do a matrix vector multiplication in $O(n \log(n))$ time, from which we can find the inverse in the same time complexity. We will discuss this later in chapter (7).

Chapter 6

Approximations

In this chapter, we try to approximate our problem enabling faster algorithms. However, this approach reduces the expressiveness of our Gaussian process. We start with the most basic sparse models and try to solve its downfalls. At the end of this chapter, using stochastic techniques, we are able to obtain a huge speed-up.

6.1 Sparse Gaussian Process Regression

In this section, we mainly follow (Quiñonero-Candela and Rasmussen, 2005) and (Snelson and Ghahramani, 2006) using some statements found in (Rasmussen and Williams, 2006). The first gives a unifying view of sparse Gaussian process regression.

We try to find $m < n$ inducing points $\boldsymbol{u}$ which are values of the Gaussian process corresponding to a set of input locations Z. These points are used to approximate the full covariance matrix over all n training inputs. This kind of methods are called sparse methods since they are sparse in the data. The determination of the locations of the inducing points will be discussed later. An advantage is that we will be able to reduce the running time complexity to $O(nm^2 + m^3)$, and memory requirements to $O(mn + m^2)$.

As has been specified in (Quiñonero-Candela and Rasmussen, 2005), we can see our algorithms as exact inference with an approximated prior. However, in other papers, approximate inference with an exact prior was used. We explain both in order to obtain the necessary intuition. The most basic model we discuss can also be derived using functional analysis. Our Bayesian view in section 4.1 was very extensive and we really started from scratch; we even did not include kernels. Let us shortly put in order what we have done. We found the predictive distribution by conditioning on $\boldsymbol{f}$ and used Bayes. We calculated

$$p(\boldsymbol{f}^*|\boldsymbol{y}, \boldsymbol{x}^*, X) = \int p(\boldsymbol{f}^*|\boldsymbol{y}, \boldsymbol{f}, X, \boldsymbol{x}^*) p(\boldsymbol{f}|\boldsymbol{y}, X) d\boldsymbol{f} \tag{6.1}$$

$$= \int p(\boldsymbol{f}^*|\boldsymbol{y}, \boldsymbol{f}, X, \boldsymbol{x}^*) \frac{p(\boldsymbol{y}|\boldsymbol{f}, X) p(\boldsymbol{f}|X)}{p(\boldsymbol{y}|X)} d\boldsymbol{f}. \tag{6.2}$$

Now, conditioning on $\boldsymbol{u}$, we find that

$$p(\boldsymbol{f}^*|\boldsymbol{y}, \boldsymbol{x}^*, X, Z) = \int p(\boldsymbol{f}^*|\boldsymbol{y}, \boldsymbol{u}, X, \boldsymbol{x}^*, Z)p(\boldsymbol{u}|\boldsymbol{y}, X, Z)d\boldsymbol{u} \tag{6.3}$$

$$= \int p(\boldsymbol{f}^*|\boldsymbol{y}, \boldsymbol{u}, X, \boldsymbol{x}^*, Z)\frac{p(\boldsymbol{y}|\boldsymbol{u}, X, Z)p(\boldsymbol{u}|Z)}{p(\boldsymbol{y}|Z, X)}d\boldsymbol{u}. \tag{6.4}$$

For the view of approximate inference with an exact prior where we replace the likelihood $p(\boldsymbol{y}|\boldsymbol{u}, X, Z)$ by an approximation $q(\boldsymbol{y}|\boldsymbol{u}, X, Z)$. We take $p(\boldsymbol{u}|Z) = \mathcal{N}(\boldsymbol{0}, K_{Z,Z})$ (reasonable since we expect the pseudo data to be distributed as the real data). Calculating this integral gives the predictive distribution.

Next, we discuss the same using exact inference with an approximated prior. From now on, we also omit the extensive notation including all the covariates for brevity. Using $p(\boldsymbol{f}^*|\boldsymbol{f})p(\boldsymbol{f}) = p(\boldsymbol{f}^*, \boldsymbol{f})$ we find that

$$p(\boldsymbol{f}^*|\boldsymbol{y}) = \frac{1}{p(\boldsymbol{y})}\int p(\boldsymbol{f}^*, \boldsymbol{f})p(\boldsymbol{y}|\boldsymbol{f})d\boldsymbol{f}. \tag{6.5}$$

As of now, we denote $K(X, X^*) = K_{X,X^*}$, which corresponds better with the more concise notation. From before, we know that the likelihood $p(\boldsymbol{y}|\boldsymbol{f}) = \mathcal{N}(\boldsymbol{f}, \sigma^2 I_n)$ and that

$$p(\boldsymbol{f}, \boldsymbol{f}^*) = \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} K_{X,X} & K_{X,X^*} \\ K_{X^*,X} & K_{X^*,X^*} \end{bmatrix}\right). \tag{6.6}$$

which we call the GP prior. For brevity, we denote with $Q_{A,B} = K_{A,Z}K_{Z,Z}^{-1}K_{Z,B}$. All the sparse approximations which we are going to discuss can be seen as approximations of the training- and test conditional $p(\boldsymbol{f}|\boldsymbol{u})$ and $p(\boldsymbol{f}^*|\boldsymbol{u})$ given by

$$\begin{aligned} p(\boldsymbol{f}|\boldsymbol{u}) &= \mathcal{N}(K_{X,Z}K_{Z,Z}^{-1}\boldsymbol{u}, K_{X,X} - Q_{X,X}) \\ p(\boldsymbol{f}^*|\boldsymbol{u}) &= \mathcal{N}(K_{X^*,Z}K_{Z,Z}^{-1}\boldsymbol{u}, K_{X^*,X^*} - Q_{X^*,X^*}). \end{aligned} \tag{6.7}$$

Compare this with equation (4.12) and convince yourself that these expressions make sense. The matrix $K_{X,Z}K_{Z,Z}^{-1}$ can be seen as some projection matrix absorbing the information of n points in m points. We denote with $q(\boldsymbol{f}|\boldsymbol{u})$ and $q(\boldsymbol{f}^*|\boldsymbol{u})$ their approximations. Approximating the joint prior by assuming that $\boldsymbol{f}^*$ and $\boldsymbol{f}$ are conditionally independent given $\boldsymbol{u}$, we find that

$$p(\boldsymbol{f}^*, \boldsymbol{f}) = \int p(\boldsymbol{f}^*, \boldsymbol{f}|\boldsymbol{u})p(\boldsymbol{u})d\boldsymbol{u} \simeq \int q(\boldsymbol{f}^*|\boldsymbol{u})q(\boldsymbol{f}|\boldsymbol{u})p(\boldsymbol{u})d\boldsymbol{u} = q(\boldsymbol{f}^*, \boldsymbol{f}). \tag{6.8}$$

Notice that due to the assumption of conditional independence, information from $\boldsymbol{f}$ can only be transferred to $\boldsymbol{f}^*$ via $\boldsymbol{u}$. In our further discussion, we omit the tedious derivations concerning the joint prior or using the approximate likelihood. This will be redundant since in section 6.2, we implicitly do the same calculations. We first discuss the most basic sparse model.

6.1.1 Subset Of Regressors

In this subsection, we discuss the Subset of regressors (SoR) approximation. The name of this method becomes clear in a moment. For this approximation, we use the approximate priors given by

$$q_{SoR}(\boldsymbol{f}|\boldsymbol{u}) = \mathcal{N}(K_{X,Z}K_{Z,Z}^{-1}\boldsymbol{u}, \mathbf{0}) \quad \text{and} \quad q_{SoR}(\boldsymbol{f}^*|\boldsymbol{u}) = \mathcal{N}(K_{X^*,Z}K_{Z,Z}^{-1}\boldsymbol{u}, \mathbf{0}), \tag{6.9}$$

which are Dirac shaped distributions. The joint prior is given by

$$q_{SoR}(\boldsymbol{f}, \boldsymbol{f}^*) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} Q_{X,X} & Q_{X,X^*} \\ Q_{X^*,X} & Q_{X^*,X^*} \end{bmatrix}\right). \tag{6.10}$$

In this case, the approximate priors are very restrictive. Only a very limited family of functions will be plausible under the posterior, given enough data. This will lead to overconfident predictive variances. However, we are mainly interested in the predictive mean for pricing options so this is not a huge problem. Denoting $\Lambda = \sigma^2 I_n$, the predictive distribution is given by

$$q_{SoR}(\boldsymbol{f}^*|\boldsymbol{y}) = \mathcal{N}(Q_{X^*,X}(Q_{X,X} + \Lambda)^{-1}\boldsymbol{y}, Q_{X^*,X^*} - Q_{X^*,X}(Q_{X,X} + \Lambda)^{-1}Q_{X,X^*}). \tag{6.11}$$

This can easily be seen replacing $\boldsymbol{f}$ by $\boldsymbol{y}$ in equation (6.10) and by conditioning the joint distribution on the observations (see appendix section 11.7). For the next sparse methods, this derivation is always the same and we will not elaborate.

Notice the strong similarities with equation (4.9). In fact, we have only replaced the matrix $K_{X,X}$ with the matrix $Q_{X,X}$. We actually replace the kernel/covariance function using an approximation, which we discuss further in a moment. However, notice that we still have to invert a $n \times n$ matrix. By applying the matrix inversion lemma, we can obtain an expression for the predictive distribution which is more economical to compute. If we denote with $\Sigma = (\Lambda^{-1}K_{Z,X}K_{X,Z} + K_{Z,Z})^{-1}$, the predictive distribution is given by

$$\boxed{q_{SoR}(\boldsymbol{f}^*|\boldsymbol{y}) = \mathcal{N}(\Lambda^{-1}K_{X^*,Z}\Sigma K_{Z,X}\boldsymbol{y}, K_{X^*,Z}\Sigma K_{Z,X^*}).} \tag{6.12}$$

Hence, we only need to invert a $m \times m$ matrix (the Σ term). We follow (Rasmussen and Williams, 2006) for the derivation of SoR using functional analysis. We take a subset of regressors (which we also denote with Z) and use this to approximate the ith eigenfunction. Using equation (3.19) and denoting with $\lambda_i^{(m)}$ and $u_i^{(m)}$ the eigenvalues and eigenvectors of matrix $K_{Z,Z}$, we find that

$$\tilde{\varphi}_i(\boldsymbol{x}) = \frac{\sqrt{m}}{\lambda_i^{(m)}} k_{\boldsymbol{x},Z}\boldsymbol{u}_i^{(m)}. \tag{6.13}$$

Hence, using $\lambda_i \approx \lambda_i^{(m)}/m$, we can obtain a approximation for the kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{n} \lambda_i \varphi_i(\boldsymbol{x})\varphi(\boldsymbol{x}')$. We find that

$$\begin{aligned}\tilde{k}_{SoR}(\boldsymbol{x}, \boldsymbol{x}') &= \sum_{i=1}^{m} \frac{\lambda_i^{(m)}}{m} \tilde{\varphi}_i(\boldsymbol{x}) \tilde{\varphi}_i(\boldsymbol{x}') \\ &= \sum_{i=1}^{m} \frac{\lambda_i^{(m)}}{m} \frac{m}{(\lambda_i^{(m)})^2} k_{\boldsymbol{x},Z} \boldsymbol{u}_i^{(m)} (\boldsymbol{u}_i^{(m)})^t k_{Z,\boldsymbol{x}'} \\ &= k_{\boldsymbol{x},Z} K_{Z,Z}^{-1} k_{Z,\boldsymbol{x}'}. \end{aligned} \tag{6.14}$$

This is the covariance function used in the SoR. The new covariance matrix has rank (at most) m (due to the properties of rank and matrix product) and thus this model is a degenerate GP only admitting m degrees of freedom. Now, equation (6.12) can even be derived fulling in the approximations in (4.7), for which we refer to (Rasmussen and Williams, 2006) (chapter 8). Now, we are ready to discuss the deterministic training conditional (DTC) approximation.

6.1.2 Deterministic Training Conditional

DTC is an improvement of the SoR approximation concerning the predictive uncertainties although it has the same predictive mean. Originally, this method was called projected latent variables (PLV) or projected process approximation (PPA) and seen as a likelihood approximation method where

$$p(\boldsymbol{y}|\boldsymbol{f}) \simeq q(\boldsymbol{y}|\boldsymbol{u}) = \mathcal{N}(K_{X,Z} K_{Z,Z}^{-1} \boldsymbol{u}, \sigma^2 I_n). \tag{6.15}$$

Notice that we use the projection matrix $K_{X,Z} K_{Z,Z}^{-1}$. However, looking at it from the view of approximate priors, one imposes a deterministic training conditional (variance 0) and an exact test conditional (thats why it is called DTC), given by

$$q_{DTC}(\boldsymbol{f}|\boldsymbol{u}) = \mathcal{N}(K_{X,Z} K_{Z,Z}^{-1} \boldsymbol{u}, \boldsymbol{0}), \quad \text{and} \quad q_{DTC}(\boldsymbol{f}^*|\boldsymbol{u}) = p(\boldsymbol{f}^*|\boldsymbol{u}). \tag{6.16}$$

The joint prior implied by DTC is given by

$$q_{DTC}(\boldsymbol{f}, \boldsymbol{f}^*) = \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} Q_{X,X} & Q_{X,X^*} \\ Q_{X^*,X} & K_{X^*,X^*} \end{bmatrix}\right). \tag{6.17}$$

Denoting $\Lambda = \sigma^2 I_n$, one gets that the predictive distribution is given by

$$q_{DTC}(\boldsymbol{f}^*|\boldsymbol{y}) = \mathcal{N}(Q_{X^*,X}(Q_{X,X} + \Lambda)^{-1} \boldsymbol{y}, K_{X^*,X^*} - Q_{X^*,X}(Q_{X,X} + \Lambda)^{-1} Q_{X,X^*}). \tag{6.18}$$

Notice the strong similarities with equation (4.9) and equation (6.11) despite a small change in the variance. However, we again have to invert a $n \times n$ matrix. By applying the matrix inversion lemma and using the same notation as before, the predictive distribution is given by

$$\boxed{q_{DTC}(\boldsymbol{f}^*|\boldsymbol{y}) = \mathcal{N}(K_{X^*,Z}\Sigma K_{Z,X}\Lambda^{-1}\boldsymbol{y}, K_{X^*,X^*} - Q_{X^*,X^*} + K_{X^*,Z}\Sigma K_{Z,X^*}).} \tag{6.19}$$

Hence, we only need to invert a $m \times m$ matrix. It is interesting to notice that the term $K_{X^*,X^*} - Q_{X^*,X^*}$ equals $\text{Cov}(\boldsymbol{f}^*|\boldsymbol{u})$. This can easily be seen looking at equation (6.16) and (6.7). Thus the predictive variance of DTC equals the sum of the predictive variance of the SoR and the predictive variance of $p(\boldsymbol{f}^*|\boldsymbol{u})$, which means that the predictive variance of DTC is never smaller than that of SoR . As a remark, we state that the DTC approximation does not exactly corresponds to a Gaussian process since the covariance between latent values depends on whether we work with training or test values as can be seen in equation (6.16).

6.1.3 Fully Independent Training Condition

Similarly, one can also deduce the fully independent training condition (FITC) approximation. In earlier literature, one called this method sparse Gaussian processes using pseudo-inputs (SGPP) and deduced it from a likelihood approximation

$$p(\boldsymbol{y}|\boldsymbol{f}) \simeq q(\boldsymbol{y}|\boldsymbol{u}) = \mathcal{N}(K_{X,Z}K_{Z,Z}^{-1}\boldsymbol{u}, \text{diag}[K_{X,X} - Q_{X,X}] + \sigma^2 I_n), \tag{6.20}$$

where one included a diagonal term for the covariance. We can again study it from the view of approximated priors. For the FITC we impose an extra independence assumption for the training conditional (that's why it is called FITC). Using equation (6.7), we find that

$$\begin{aligned} q_{FITC}(\boldsymbol{f}|\boldsymbol{u}) &= \prod_{i=1}^{n} p(f_i, \boldsymbol{u}) = \mathcal{N}(K_{X,Z}K_{Z,Z}^{-1}\boldsymbol{u}, \text{diag}[K_{X,X} - Q_{X,X}]) \\ q_{FITC}(\boldsymbol{f}^*|\boldsymbol{u}) &= p(\boldsymbol{f}^*|\boldsymbol{u}). \end{aligned} \tag{6.21}$$

The joint prior implied by FITC is given by

$$q_{FITC}(\boldsymbol{f}, \boldsymbol{f}^*) = \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} Q_{X,X} - \text{diag}[Q_{X,X} - K_{X,X}] & Q_{X,X^*} \\ Q_{X^*,X} & K_{X^*,X^*} \end{bmatrix}\right). \tag{6.22}$$

We denote with $\Lambda = \text{diag}[K_{X,X} - Q_{X,X} + \sigma^2 I_n]$. The predictive distribution is given by

$$q_{FITC}(\boldsymbol{f}^*|\boldsymbol{y}) = \mathcal{N}(Q_{X^*,X}(Q_{X,X} + \Lambda)^{-1}\boldsymbol{y}, K_{X^*,X^*} - Q_{X^*,X}(Q_{X,X} + \Lambda)^{-1}Q_{X,X^*}). \tag{6.23}$$

Notice the strong similarities with equations (4.9) and (6.18). Rewriting it with $\Sigma = (K_{Z,Z} + K_{Z,X}\Lambda^{-1}K_{X,Z})^{-1}$ and using the matrix inversion lemma, we obtain that

$$\boxed{q_{FITC}(\boldsymbol{f}^*|\boldsymbol{y}) = \mathcal{N}(K_{X^*,Z}\Sigma K_{Z,X}\Lambda^{-1}\boldsymbol{y}, K_{X^*,X^*} - Q_{X^*,X^*} + K_{X^*,Z}\Sigma K_{Z,X^*}).} \tag{6.24}$$

This is again more economical to compute. Notice that FITC does not correspond to a GP model since the training and test covariance are not computed the same. If we extend the additional factorizing assumption to the test conditional, we obtain the fully independent conditional (FIC) approximation which is a non-degenerate Gaussian process. The joint prior implied by FIC is given by

$$q_{FIC}(\boldsymbol{f}, \boldsymbol{f}^*) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} Q_{X,X} - \text{diag}[Q_{X,X} - K_{X,X}] & Q_{X,X^*} \\ Q_{X^*,X} & Q_{X^*,X^*} - \text{diag}[Q_{X^*,X^*} - K_{X^*,X^*}] \end{bmatrix}\right). \quad (6.25)$$

We give a short overview of the approximate kernels we encountered.

$$\tilde{k}_{SoR}(\boldsymbol{x}_i, \boldsymbol{x}_j) = k_{\boldsymbol{x}_i,Z} K_{Z,Z}^{-1} k_{Z,\boldsymbol{x}_j} \quad (6.26)$$

$$\tilde{k}_{FIC}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tilde{k}_{SoR}(\boldsymbol{x}_i, \boldsymbol{x}_j) + \delta_{i,j}\left(k(\boldsymbol{x}_i, \boldsymbol{x}_j) - \tilde{k}_{SoR}(\boldsymbol{x}_i, \boldsymbol{x}_j)\right). \quad (6.27)$$

6.1.4 Selecting The Inducing Points And Time Complexities

Now, we are ready to discuss how to select the inducing variables. We discuss three methods although there are much more different approaches possible. One can just use a subset of the data. However, this is often not sufficient thus one can decide to use K-means (see appendix section 11.11) such that each latent variable represents a cluster of the data. The time complexity of LLoyd's algorithm with p iterations is $O(nmdp)$. For large datasets, it is recommended to use mini-batching. Another approach is maximising the marginal likelihood (evidence) with respect to Z like in (Snelson and Ghahramani, 2006). For the different models, using the right Λ, the evidence is given by

$$\log(q(\boldsymbol{y}|Z, X)) = -\frac{1}{2}\log|Q_{X,X} + \Lambda| - \frac{1}{2}\boldsymbol{y}^t(Q_{X,X} + \Lambda)^{-1}\boldsymbol{y} - \frac{n}{2}\log(2\pi). \quad (6.28)$$

Theorem 9 (Computation and storage costs evidence sparse methods) *The computational cost of equation (6.28) is $O(nm^2 + m^3)$ and the storage demands equals $O(nm + m^3)$.*

Proof: The matrix inversion lemma deals with the major part of the work. As one can suspect, the first term comes from a matrix matrix multiplication with dimensions $m \times m$ and $m \times n$ and the second term comes from a matrix inversion of dimension $m \times m$. Let us give the derivation. By using Cholesky and writing $K_{Z,Z} = L_Z L_Z^t$ $O(m^3)$, we find that

$$\begin{aligned}(Q_{X,X} + \Lambda)^{-1} &= (K_{X,Z}K_{Z,Z}^{-1}K_{Z,X} + \Lambda)^{-1} \\ &= (K_{X,Z}(L_Z^t)^{-1}I_m(L_Z)^{-1}K_{Z,X} + \Lambda)^{-1} \\ &= \Lambda^{-1} - \Lambda^{-1}K_{X,Z} \\ &\quad (L_Z^t)^{-1}(I_m + L_Z^{-1}K_{Z,X}\Lambda^{-1}K_{X,Z}(L_Z^t)^{-1})^{-1}L_Z^{-1}K_{Z,X}\Lambda^{-1}.\end{aligned} \quad (6.29)$$

Now, we again need to take the inverse of a $m \times m$ matrix which requires $O(m^3)$ time to evaluate. In order to construct this matrix, we need $O(nm^2)$ time. Using Cholesky, we find that

$$(I_m + L_Z^{-1} K_{Z,X} \Lambda^{-1} K_{X,Z} (L_Z^t)^{-1}) = \mathcal{L}_Z \mathcal{L}_Z^t \tag{6.30}$$

and thus we can write

$$(Q_{X,X} + \Lambda)^{-1} = \Lambda^{-1} - \Lambda^{-1} K_{X,Z} (L_Z^t)^{-1} (\mathcal{L}_Z^t)^{-1} (\mathcal{L}_Z)^{-1} L_Z^{-1} K_{Z,X} \Lambda^{-1}. \tag{6.31}$$

Hence, we only need to invert a diagonal matrix of dimension $n \times n$ and two matrices of dimension $m \times m$. Multiplying this expression with $\boldsymbol{y}$ results in a time complexity of $O(nm)$. Using Sylvester's determinant theorem (see appendix section 11.12), one can see that the determinant can be calculated in a very economical way. We find that

$$\begin{aligned} \det(Q_{X,X} + \Lambda) &= \det(\Lambda) \det(\Lambda^{-1} Q_{X,X} + I_n) \\ &= \det(\Lambda) \det(\Lambda^{-1} K_{X,Z} (L_Z^t)^{-1} L_Z^{-1} K_{Z,X} + I_n) \\ &= \det(\Lambda) \det(I_m + (L_Z)^{-1} K_{Z,X} \Lambda^{-1} K_{X,Z} (L_Z^t)^{-1}). \end{aligned} \tag{6.32}$$

Notice that the first determinant is just the product of the diagonal elements of Λ and that the second term equals equation (6.30). Since, we already calculated its Cholesky decomposition, we just need to calculate the product of squares of the diagonal elements. However, we need to calculate the logarithm of this determinant, thus we just need to calculate two times the sum of the logarithms of these diagonal elements. As a remark, one only needs the diagonal elements of $K_{X,X}$ (and $Q_{X,X}$ for some sparse models) in order to calculate Λ.

Since one has to construct the matrix $K_{Z,X}$, its transpose and some $m \times m$ matrices, the storage demands are $O(nm + m^2)$. □

Theorem 10 (Computation cost prediction sparse methods) *The computational cost to find the predictive mean for one observation using sparse methods is $O(m)$.*

Proof: The vector $\Sigma K_{Z,X} \Lambda^{-1} \boldsymbol{y}$ has been calculated during training. Hence, the construction and vector multiplication with $k_{\boldsymbol{x}^*,Z}$ only takes $O(m)$ time. □

Notice that we now have to fit $md + \#\,\boldsymbol{\theta}$ parameters and overfitting can occur. In other words, the locations Z form a set of extra hyperparameters. Furthermore, optimising more variables requires more time. Hence, using K-means is recommended. However, we find a more rigorous solution solving overfitting in the next section.

6.2 Variational Gaussian Process Regression

One can decide to optimise inducing point locations as part of the model like in (Titsias, 2009) and (Hensman et al., 2013). The inducing points positions are variational parameters rather than model parameters and are thus protected from overfitting.

Readers which are not familiar with variational learning are referred to section 11.13 in the appendix.

Like before, we consider the following

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{f}) &= \mathcal{N}(\boldsymbol{y}|\boldsymbol{f}, \sigma^2 I_n) \\ p(\boldsymbol{f}|\boldsymbol{u}) &= \mathcal{N}(\boldsymbol{f}|K_{X,Z}K_{Z,Z}^{-1}\boldsymbol{u}, K_{X,X} - Q_{X,X}) \\ p(\boldsymbol{u}) &= \mathcal{N}(\boldsymbol{u}|\boldsymbol{0}, K_{Z,Z}). \end{aligned} \tag{6.33}$$

First, we try to obtain a lower bound of $p(\boldsymbol{y}|\boldsymbol{u})$, which is also known as the variational or evidence lower bound (ELBO), but now given the inducing points. Using Bayes on $p(\boldsymbol{f}|\boldsymbol{y}, \boldsymbol{u})$ and after an algebraic manipulation, we find that

$$p(\boldsymbol{y}|\boldsymbol{u}) = \frac{p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{u})p(\boldsymbol{f}|\boldsymbol{u})}{p(\boldsymbol{f}|\boldsymbol{y}, \boldsymbol{u})}. \tag{6.34}$$

Taking the natural logarithm, we obtain that

$$\log\left(p(\boldsymbol{y}|\boldsymbol{u})\right) = \log\left(p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{u})\right) + \log\left(\frac{p(\boldsymbol{f}|\boldsymbol{u})}{p(\boldsymbol{f}|\boldsymbol{y}, \boldsymbol{u})}\right). \tag{6.35}$$

Since the left hand side is not a function of $\boldsymbol{f}$, we find by taking the expectation to $p(\boldsymbol{f}|\boldsymbol{u})$ that

$$\log\left(p(\boldsymbol{y}|\boldsymbol{u})\right) = \mathbb{E}_{p(\boldsymbol{f}|\boldsymbol{u})} \log\left(p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{u})\right) + \mathbb{E}_{p(\boldsymbol{f}|\boldsymbol{u})}\left[\log\left(\frac{p(\boldsymbol{f}|\boldsymbol{u})}{p(\boldsymbol{f}|\boldsymbol{y}, \boldsymbol{u})}\right)\right]. \tag{6.36}$$

Note that the last term is the Kullback Leibler (KL) divergence between the posterior of $\boldsymbol{f}$ given the data and the inducing variables and the posterior of $\boldsymbol{f}$ given the inducing variables only. From now, we denote with $\tilde{p}(\boldsymbol{y}|\boldsymbol{u}) = \mathbb{E}_{p(\boldsymbol{f}|\boldsymbol{u})} \log\left(p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{u})\right)$ (it is independent of $\boldsymbol{f}$). Hence we can write that

$$\log\left(p(\boldsymbol{y}|\boldsymbol{u})\right) = \tilde{p}(\boldsymbol{y}|\boldsymbol{u}) + \mathrm{KL}[p(\boldsymbol{f}|\boldsymbol{u})||p(\boldsymbol{f}|\boldsymbol{y}, \boldsymbol{u})]. \tag{6.37}$$

Since $p(\boldsymbol{y}|\boldsymbol{f})$ is separable across the data

$$p(\boldsymbol{y}|\boldsymbol{f}) = \prod_{i=1}^{n} p(y_i|f_i), \tag{6.38}$$

the following lemma can be found.

Lemma 1 (Expression of $\exp(\tilde{p}(\boldsymbol{y}|\boldsymbol{u}))$) *We find that*

$$\begin{aligned}\exp(\tilde{p}(\boldsymbol{y}|\boldsymbol{u})) &= \mathcal{N}(\boldsymbol{y}|K_{X,Z}K_{Z,Z}^{-1}\boldsymbol{u}, \sigma^2 I_n)\exp\left(-\frac{1}{2\sigma^2}\operatorname{Tr}(K_{X,X}-Q_{X,X})\right)\\ &= \prod_{i=1}^{n}\mathcal{N}(y_i|k_{\boldsymbol{x}_i,\boldsymbol{z}}K_{Z,Z}^{-1}\boldsymbol{u}, \sigma^2)\exp\left(-\frac{1}{2\sigma^2}(k_{\boldsymbol{x}_i,\boldsymbol{x}_i}-k_{\boldsymbol{x}_i,\boldsymbol{z}}K_{Z,Z}^{-1}k_{\boldsymbol{z},\boldsymbol{x}_i})\right). \quad (6.39)\end{aligned}$$

Proof: Using $p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{u}) = p(\boldsymbol{y}|\boldsymbol{f})$, we calculate

$$\begin{aligned}\tilde{p}(\boldsymbol{y}|\boldsymbol{u}) &= \mathbb{E}_{p(\boldsymbol{f}|\boldsymbol{u})}\log\left(p(\boldsymbol{y}|\boldsymbol{f})\right)\\ &= \int p(\boldsymbol{f}|\boldsymbol{u})\log(p(\boldsymbol{y}|\boldsymbol{f})d\boldsymbol{f}\\ &= \int p(\boldsymbol{f}|\boldsymbol{u})\left(-\frac{n}{2}\log(2\pi\sigma^2)-\frac{1}{2\sigma^2}\operatorname{Tr}\left((\boldsymbol{y}-\boldsymbol{f})^2\right)\right)d\boldsymbol{f}\\ &= -\frac{n}{2}\log(2\pi\sigma^2)\int p(\boldsymbol{f}|\boldsymbol{u})d\boldsymbol{f} - \frac{1}{2\sigma^2}\int p(\boldsymbol{f}|\boldsymbol{u})\operatorname{Tr}\left(\boldsymbol{y}\boldsymbol{y}^t - 2\boldsymbol{y}\boldsymbol{f}^t - \boldsymbol{f}\boldsymbol{f}^t\right)d\boldsymbol{f} \quad (6.40)\end{aligned}$$

Now, we denote with $\boldsymbol{\alpha} = \mathbb{E}(\boldsymbol{f}|\boldsymbol{u})$. Noting that the variance equals the difference between the second moment and the first moments squared, we find that

$$\begin{aligned}\tilde{p}(\boldsymbol{y}|\boldsymbol{u}) &= -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\operatorname{Tr}\left(\boldsymbol{y}\boldsymbol{y}^t - 2\boldsymbol{y}\boldsymbol{\alpha}^t - \boldsymbol{\alpha}\boldsymbol{\alpha}^t + K_{X,X} - Q_{X,X}\right)\\ &= \log\mathcal{N}(\boldsymbol{y}|\boldsymbol{\alpha}, \sigma^2 I) - \frac{1}{2\sigma^2}\operatorname{Tr}(K_{X,X}-Q_{X,X}). \quad (6.41)\end{aligned}$$

We find the solution by taking the exponential. □

Hence, equation (6.41) is a lower bound of $p(\boldsymbol{y}|\boldsymbol{u})$ which is separable across $\boldsymbol{y}$ and consists of a likelihood approximation and a penalty term. The goal of variational inference is to minimize the KL divergence. Hence, instead of modifying the exact GP model, we minimize the distance between the exact posterior GP and a variational approximation. In this case, this means that we want that $p(\boldsymbol{f}|\boldsymbol{u})$ is similar to $p(\boldsymbol{f}|\boldsymbol{y},\boldsymbol{u})$.

Let us first discuss the case were the inducing variables are placed at the training data locations ($X = Z$). The penalty term in equation (6.41) equals one and the KL divergence equals zero. When $m < n$ we can minimize the KL with respect to Z. This ensures that Z is distributed amongst the training data X such that the penalty term is small. Using this, we can find a variational lower bound for the true log marginal likelihood $\log\left(p(\boldsymbol{y}|X)\right)$ (evidence) which was derived by (Titsias, 2009).

Theorem 11 *A variational lower bound for the evidence is given by*

$$\begin{aligned}\log(p(\boldsymbol{y}|X)) &= \log\left(\int p(\boldsymbol{y}|\boldsymbol{u})p(\boldsymbol{u})d\boldsymbol{u}\right) \geq \log\left(\int\exp(\tilde{p}(\boldsymbol{y}|\boldsymbol{u}))p(\boldsymbol{u})d\boldsymbol{u}\right)\\ &= \log\mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, \sigma^2 I_n + Q_{X,X}) - \frac{1}{2\sigma^2}\operatorname{Tr}(K_{X,X}-Q_{X,X}). \quad (6.42)\end{aligned}$$

Proof: The inequality follows from equation (6.37) and the fact that $KL(\cdot) \geq 0$. Using lemma (9) in the appendix section 11.7, we find that

$$\begin{aligned}
&\log\left(\int \exp(\tilde{p}(\boldsymbol{y}|\boldsymbol{u}))p(\boldsymbol{u})d\boldsymbol{u}\right) \\
&= \log\left(\int \mathcal{N}(\boldsymbol{y}|\boldsymbol{\alpha}, \sigma^2 I_n)\mathcal{N}(\boldsymbol{u}|\boldsymbol{0}, K_{Z,Z})d\boldsymbol{u}\right) - \frac{1}{2\sigma^2}\operatorname{Tr}(K_{X,X} - Q_{X,X}) \\
&= \log\left(\int \mathcal{N}(\boldsymbol{y}|\boldsymbol{\alpha}, \sigma^2 I_n)\mathcal{N}(\boldsymbol{\alpha}|\boldsymbol{0}, Q_{X,X})d\boldsymbol{\alpha}\right) - \frac{1}{2\sigma^2}\operatorname{Tr}(K_{X,X} - Q_{X,X}) \\
&= \log\left(N(\boldsymbol{y}|\boldsymbol{0}, \sigma^2 I_n + Q_{X,X})\right) - \frac{1}{2\sigma^2}\operatorname{Tr}(K_{X,X} - Q_{X,X}). \qquad (6.43)
\end{aligned}$$

□

Notice that the first term is the same as equation (6.28). However, the variational method is completely different than the other inducing point methods concerning the selection of the inducing inputs and the kernel hyperparameters due to the extra regularisation term in equation (6.42). As has been said before, the inducing point locations Z can be seen as variational parameters instead of an extra set of hyperparameters. The method optimising equation (6.42) is called the VFE (variational free energy) method.

The matrix $(K_{X,X} - Q_{X,X})$ equals $\operatorname{Cov}(\boldsymbol{f}|\boldsymbol{u})$, which corresponds to the squared error of prediction the training values $\boldsymbol{f}$ from the inducing variables $\boldsymbol{u}$ which equals zero if $X = Z$. Titsias stated that this trace term protect against overfitting takes care of selecting good inducing points. The lower bound (6.42) has again a computational complexity of $O(nm^2 + m^3)$ and storage demands of $O(nm + m^3)$. In order to compute the trace, we only need the diagonal elements of $K_{X,X}$ and we do not need to calculate the whole kernel.

Next, using $p(\boldsymbol{u})$ and the approximation $\exp(\tilde{p}(\boldsymbol{y}|\boldsymbol{u}))$ for $p(\boldsymbol{y}|\boldsymbol{u})$, we find that

$$\begin{aligned}
q(\boldsymbol{u}|\boldsymbol{y}) &\propto \exp(\tilde{p}(\boldsymbol{y}|\boldsymbol{u}))p(\boldsymbol{u}) \\
&\propto \mathcal{N}(\boldsymbol{y}|\boldsymbol{\alpha}, \sigma^2 I_n)p(\boldsymbol{u}) \\
&= c\exp\left(\frac{1}{2\sigma^2}\boldsymbol{u}^t K_{Z,Z}^{-1}K_{Z,X}K_{X,Z}K_{Z,Z}^{-1}\boldsymbol{u} + \frac{1}{\sigma^2}\boldsymbol{y}^t K_{X,Z}K_{Z,Z}^{-1}\boldsymbol{u} + \frac{1}{2\sigma^2}\boldsymbol{u}^t K_{Z,Z}^{-1}\boldsymbol{u}\right), \qquad (6.44)
\end{aligned}$$

where c is a constant. By completing the square and denoting with $\Sigma = (\Lambda^{-1}K_{Z,X}K_{X,Z} + K_{Z,Z})$, we recognise a Gaussian and find that

$$q(\boldsymbol{u}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{u}|\sigma^{-2}K_{Z,Z}\Sigma K_{Z,X}\boldsymbol{y}, K_{Z,Z}\Sigma K_{Z,Z}^{-1}). \qquad (6.45)$$

Now, using equation (6.3), we find that

$$\begin{aligned}\mathbb{E}[q(\boldsymbol{f}^*|\boldsymbol{y})] &= \int \mathbb{E}[p(\boldsymbol{f}^*|\boldsymbol{u})]q(\boldsymbol{u}|\boldsymbol{y})d\boldsymbol{u} \\ &= \int K_{X^*,Z}K_{Z,Z}^{-1}\boldsymbol{u}q(\boldsymbol{u}|\boldsymbol{y})d\boldsymbol{u} \\ &= K_{X^*,Z}K_{Z,Z}^{-1}\mathbb{E}[q(\boldsymbol{u}|\boldsymbol{y})] \\ &= \sigma^{-2}K_{X^*,Z}\Sigma K_{Z,X}\boldsymbol{y}, \end{aligned} \tag{6.46}$$

which is the same as for DTC. This makes sense since $\exp(\tilde{p}(y|u))$ is equal to equation (6.15). In the previous section, we mainly focussed on deriving the predictive distribution with approximate priors, although this can also be done using likelihood approximations as has been shown here. A logical extension will be a variational extension for the FITC. For the DTC we have that $\Lambda = \sigma^2 I_n$ and for FITC we have that $\Lambda = \text{diag}[K_{X,X} - Q_{X,X} + \sigma^2 I_n]$. Hence replacing $\sigma^2 I_n$ in equation (6.33) by Λ, we find a generalisation which may be suitable if one wants to include input-dependent variance.

We omit the derivation for the predictive variance since we are mainly interested for the predictions and the calculations are tedious. Despite the approach is different in both (Titsias, 2009) and (Rasmussen and Williams, 2006) (chapter 8), some concise calculations can be found which can help. One finds that

$$\mathbb{V}[q(\boldsymbol{f}^*|\boldsymbol{y})] = K_{X^*,X^*} - Q_{X^*,X^*} + K_{X^*,Z}\Sigma K_{X^*,Z}^t, \tag{6.47}$$

with Σ defined as before.

A thorough discussion concerning the difference in behaviour of the FITC and the standard VFE approximation can be found in (Bauer et al., 2016) which is definitely worth a read. We give a short resume of this paper since the behaviour of the FITC and VFE can be explained thanks to these results which comes at handy in our data study. Suppose we also train the noise variance parameter σ. We have that the evidence/lower bound for FITC and VFE is given by

$$\text{Objective} = \underbrace{-\frac{1}{2}\log|Q_{X,X} + \Lambda|}_{\text{Complexity penalty}} \underbrace{-\frac{1}{2}\boldsymbol{y}^t(Q_{X,X} + \Lambda)^{-1}\boldsymbol{y}}_{\text{Data Fit}} \underbrace{-\frac{1}{2\sigma^2}\text{Tr}(T)}_{\text{Trace Term}} - \frac{n}{2}\log(2\pi). \tag{6.48}$$

were the interpretation of the parameters may be different for each method. The trace term T equals 0 for the FITC for example.

First, FITC can severely underestimate the noise variance and VFE tends to overestimate it. The diagonal term $\text{diag}(K_{X,X} - Q_{X,X})$ is zero close to an inducing point and grows to the prior variance away from an inducing point. It can be seen as a heteroscedastic noise term. For FITC and data points far from the mean and the inducing points, the data fit term is small since the heteroscedastic noise term has increased. Hence, σ^2 loses its meaning and can be decreased to reduce the complexity penalty. In extreme cases, σ^2 can be estimated to be almost zero. Next, VFE tends to over-estimate the noise variance because both the data fit and trace term can be reduced by increasing σ^2.

Furthermore, VFE always improves by including additional inducing inputs although FITC may ignore them. Adding an inducing point can be seen as a rank 1 update of $Q_{X,X}$ and is always advantageous for the VFE bound. For FITC, the heteroscedastic noise term decreases at all points which reduces the complexity penalty but worsens the data fit penalty.

By placing the inducing points on every training input, VFE and FITC both recover the true posterior. This equals the global optimum for VFE since the KL gap is zero. In this case, the lower bound (6.42) just becomes the evidence. However, the objective function of FITC can still be improved with clumping of inducing points.

Finally, in (Bauer et al., 2016), they stated that the objective function of VFE is harder to optimise than FITC due to more amenable local optima.

Although this is maybe the best that we have found so far we can still obtain a faster algorithm.

6.3 Stochastic Variational Gaussian Process Regression

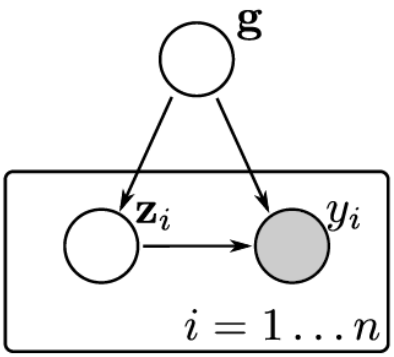


Figure 6.1: Requirements for SVI.

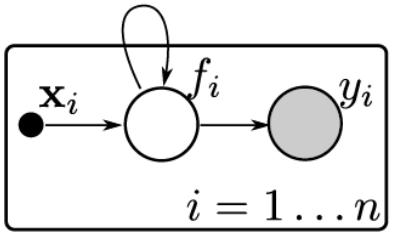


Figure 6.2: Standard GPR.

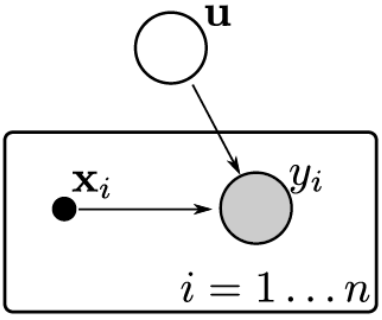


Figure 6.3: Variational GPR.

All plots are from (Hensman et al., 2013).

More information and assumptions of stochastic variational inference can be found in the appendix section 11.14. As has been stated in (Hensman et al., 2013) and in the appendix, we must have a set of global variables and we must have that $\boldsymbol{y}$ factorises in the latent variables (figure (6.1)). In figure (6.2)), one can see a plot of standard GPR were the connectivity between the values of $\boldsymbol{f}$ is denoted by a loop. In stochastic variational GPR, the variables $\boldsymbol{u}$ fulfil the role of global variables (figure (6.3)), which will enable stochastic variational inference. We use the term $\tilde{p}(\boldsymbol{y}|\boldsymbol{u})$ which act as $\log(p(y_i|\boldsymbol{u}, \boldsymbol{x}_i)$.

We introduce a variational distribution $q(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{m}, S)$. Now we are able to find another lower bound $\mathcal{L}_1$ of $p(\boldsymbol{y}|X)$ (the evidence). By writing $p(\boldsymbol{u}, \boldsymbol{y}) = p(\boldsymbol{u}|\boldsymbol{y})p(\boldsymbol{y})$, we find that

$$\mathrm{KL}[q(\boldsymbol{u})||p(\boldsymbol{u}|\boldsymbol{y})] = \mathbb{E}_{q(\boldsymbol{u})}[\log q(\boldsymbol{u}))] - \mathbb{E}_{q(\boldsymbol{u})}[\log(p(\boldsymbol{u}, \boldsymbol{y}))] + \log(p(\boldsymbol{y}|X)). \tag{6.49}$$

Then defining $\mathcal{L}_1$ as

$$\mathcal{L}_1 = -\mathbb{E}_{q(\boldsymbol{u})}[\log(q(\boldsymbol{u}))] + \mathbb{E}_{q(\boldsymbol{u})}[(\log(p(\boldsymbol{u}, \boldsymbol{y}))], \tag{6.50}$$

we find that

$$\log\left(p(\boldsymbol{y}|X)\right) = \mathrm{KL}[q(\boldsymbol{u})||p(\boldsymbol{u}|\boldsymbol{y})] + \mathcal{L}_1. \tag{6.51}$$

Since that $KL(\cdot) \geq 0$, we find that $\mathcal{L}_1$ is indeed a lower bound for the evidence. Hence, maximising this bound makes the KL-divergence between $p(\boldsymbol{u}|\boldsymbol{y})$ and $q(\boldsymbol{u})$ minimal. In other words, $q(\boldsymbol{u})$ will be similar to the true posterior $p(\boldsymbol{u}|\boldsymbol{y})$. Next, we use $p(\boldsymbol{u}, \boldsymbol{y}) = p(\boldsymbol{y}|\boldsymbol{u})p(\boldsymbol{u})$ and find that

$$\begin{aligned}\mathcal{L}_1 &= -\mathbb{E}_{q(\boldsymbol{u})}[\log(p(\boldsymbol{u}))] + \mathbb{E}_{q(\boldsymbol{u})}[(\log(p(\boldsymbol{y}|\boldsymbol{u}))] + \mathbb{E}_{q(\boldsymbol{u})}[\log(q(\boldsymbol{u}))] \\ &= \mathbb{E}_{q(\boldsymbol{u})}[(\log(p(\boldsymbol{y}|\boldsymbol{u}))] - \mathrm{KL}[q(\boldsymbol{u})||p(\boldsymbol{u})]\end{aligned} \tag{6.52}$$

From equation (6.37), we see that $\mathcal{L}_1$ is dependent on $\boldsymbol{f}$. In order to execute stochastic variational inference, this may not be the case. Hence, we find a lower bound independent of $\boldsymbol{f}$ for the evidence given by

$$\log\left(p(\boldsymbol{y}|X)\right) \geq \mathcal{L}_1 \geq \mathbb{E}_{q(\boldsymbol{u})}[\tilde{p}(\boldsymbol{y}|\boldsymbol{u})] - \mathrm{KL}[q(\boldsymbol{u})||p(\boldsymbol{u})] = \mathcal{L}_2, \tag{6.53}$$

which we will try to maximise. The previous derivation of the lower bound $\mathcal{L}_2$ was derived focussing on the independence of $\boldsymbol{f}$. Now, we derive the same expression using a more intuitive approach. In variational inference, we always want to minimise the KL-divergence between approximate variational posterior q and the true posterior p. Notice that $\boldsymbol{f}$ and $\boldsymbol{u}$ are the free parameters of the model. Now, choosing a variational posterior

$$q(\boldsymbol{f}, \boldsymbol{u}) = p(\boldsymbol{f}|\boldsymbol{u})q(\boldsymbol{u}), \tag{6.54}$$

and noticing that $KL(\cdot) \geq 0$, we find that

$$
\begin{aligned}
0 &\geq -\mathrm{KL}[q(\boldsymbol{f},\boldsymbol{u})||p(\boldsymbol{f},\boldsymbol{u}|\boldsymbol{y})] \\
&= \mathbb{E}_{q(\boldsymbol{f},\boldsymbol{u})} \log\left(\frac{p(\boldsymbol{f},\boldsymbol{u}|\boldsymbol{y})}{q(\boldsymbol{f},\boldsymbol{u})}\right) \\
&= \mathbb{E}_{q(\boldsymbol{f},\boldsymbol{u})} \log\left(\frac{p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{u})p(\boldsymbol{f}|\boldsymbol{u})p(\boldsymbol{u})}{p(\boldsymbol{y})p(\boldsymbol{f}|\boldsymbol{u})q(\boldsymbol{u})}\right) \\
&= \int\int \log\left(\frac{p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{u})p(\boldsymbol{u})}{p(\boldsymbol{y})q(\boldsymbol{u})}\right) q(\boldsymbol{f},\boldsymbol{u}) d\boldsymbol{f} d\boldsymbol{u} \\
&= \int\int \log\left(\frac{p(\boldsymbol{u})}{q(\boldsymbol{u})}\right) q(\boldsymbol{u},\boldsymbol{f}) d\boldsymbol{f} d\boldsymbol{u} + \int\int \log\left(\frac{p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{u})}{p(\boldsymbol{y})}\right) q(\boldsymbol{f},\boldsymbol{u}) d\boldsymbol{f} d\boldsymbol{u} \\
&= \int \log\left(\frac{p(\boldsymbol{u})}{q(\boldsymbol{u})}\right) q(\boldsymbol{u}) d\boldsymbol{u} + \int\int \log(p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{u}))p(\boldsymbol{f}|\boldsymbol{u})q(\boldsymbol{u}) d\boldsymbol{f} d\boldsymbol{u} - \log(p(\boldsymbol{y})) \\
&= -\mathrm{KL}[q(\boldsymbol{u})||p(\boldsymbol{u})] + \mathbb{E}_{q(\boldsymbol{u})}[\mathbb{E}_{p(\boldsymbol{f}|\boldsymbol{u})} \log(p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{u})] - \log(p(\boldsymbol{y})) \\
&= -\mathrm{KL}[q(\boldsymbol{u})||p(\boldsymbol{u})] + \mathbb{E}_{q(\boldsymbol{u})}[\tilde{p}(\boldsymbol{y}|\boldsymbol{u})] - \log(p(\boldsymbol{y})).
\end{aligned} \tag{6.55}
$$

Hence we find that

$$
\log(p(\boldsymbol{y}|X)) \geq -\mathrm{KL}[q(\boldsymbol{u})||p(\boldsymbol{u})] + \mathbb{E}_{q(\boldsymbol{u})}[\tilde{p}(\boldsymbol{y}|\boldsymbol{u})] = \mathcal{L}_2. \tag{6.56}
$$

The following lemma can be found.

Lemma 2 *Denoting with* $\Gamma_i = \frac{1}{\sigma^2} K_{Z,Z}^{-1} k_{Z,\boldsymbol{x}_i} k_{\boldsymbol{x}_i,Z} K_{Z,Z}^{-1}$, $\mathcal{L}_2$ *has the following explicit expression*

$$
\begin{aligned}
\mathcal{L}_2 = \sum_{i=1}^{n} \left\{ \log \mathcal{N}(y_i | k_{\boldsymbol{x}_i,Z} K_{Z,Z}^{-1} \boldsymbol{m}, \sigma^2) - \frac{1}{2\sigma^2}(k_{\boldsymbol{x}_i,\boldsymbol{x}_i} - q_{\boldsymbol{x}_i,\boldsymbol{x}_i}) - \frac{1}{2}\operatorname{Tr}(S\Gamma_i) \right\} \\
-KL[q(\boldsymbol{u})||p(\boldsymbol{u})]
\end{aligned} \tag{6.57}
$$

Proof: The origin of the Kullback-Leiber component in equation (6.57) is obvious. We find that

$$
\begin{aligned}
\mathbb{E}_{q(\boldsymbol{u})}[\tilde{p}(\boldsymbol{y}|\boldsymbol{u})] &= \int q(\boldsymbol{u}) \tilde{p}(\boldsymbol{y}|\boldsymbol{u}) d\boldsymbol{u} \\
&= \int q(\boldsymbol{u}) \left(\log\left(\mathcal{N}(\boldsymbol{y}|\boldsymbol{\alpha}, \sigma^2 I_n)\right) - \frac{1}{2\sigma^2}\operatorname{Tr}(K_{X,X} - Q_{X,X}) \right) d\boldsymbol{u} \\
&= \int q(\boldsymbol{u}) \left(\log\left(\mathcal{N}(\boldsymbol{y}|\boldsymbol{\alpha}, \sigma^2 I_n)\right) \right) d\boldsymbol{u} - \sum_{i=1}^{n} \frac{1}{2\sigma^2}(k_{\boldsymbol{x}_i,\boldsymbol{x}_i} - q_{\boldsymbol{x}_i,\boldsymbol{x}_i}),
\end{aligned} \tag{6.58}
$$

since the integral of a density function equals one. Hence, the trace term is equal to the middle term in equation (6.57). By omitting this term we find

$$\begin{aligned}\int q(\boldsymbol{u}) \log\left(\mathcal{N}(\boldsymbol{y}|\boldsymbol{\alpha}, \sigma^2 I_n)\right) d\boldsymbol{u} &= \int q(\boldsymbol{u}) \left(-\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\operatorname{Tr}\left((\boldsymbol{y}-\boldsymbol{\alpha})^2\right)\right) d\boldsymbol{u} \\ &= -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\int q(\boldsymbol{u}) \operatorname{Tr}\left(\boldsymbol{y}\boldsymbol{y}^t - 2\boldsymbol{y}\boldsymbol{\alpha}^t - \boldsymbol{\alpha}\boldsymbol{\alpha}^t\right) d\boldsymbol{u}.\end{aligned} \tag{6.59}$$

Now, remember that $q(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{m}, S)$ and denote with $\boldsymbol{\beta} = K_{X,Z}K_{Z,Z}^{-1}\boldsymbol{m}$. Noting that the variance equals the difference between the second moment and the first moment squared and that matrices in a trace of a product can be switched, we find that

$$\begin{aligned}\mathbb{E}_{q(\boldsymbol{u})}[\tilde{p}(\boldsymbol{y}|\boldsymbol{u})] &= -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\operatorname{Tr}\left(\boldsymbol{y}\boldsymbol{y}^t - 2\boldsymbol{y}\boldsymbol{\beta}^t + \boldsymbol{\beta}\boldsymbol{\beta}^t\right) - \frac{1}{2\sigma^2}\operatorname{Tr}\left(SK_{Z,Z}^{-1}K_{Z,X}K_{X,Z}K_{Z,Z}^{-1}\right) \\ &= \log\mathcal{N}(\boldsymbol{y}|\boldsymbol{\beta}, \sigma^2 I_n) - \sum_{i=1}^{n}\frac{1}{2\sigma^2}\operatorname{Tr}(S\Gamma_i) \\ &= \sum_{i=1}^{n}\left(\log\mathcal{N}(y_i|k_{\boldsymbol{x}_i,Z}K_{Z,Z}^{-1}\boldsymbol{m}, \sigma^2 I_n) - -\sum_{i=1}^{n}\frac{1}{2\sigma^2}\operatorname{Tr}(S\Gamma_i)\right).\end{aligned} \tag{6.60}$$

We find the result by putting everything together. □

Next, we can calculate the gradients of $\mathcal{L}_2$ with respect to the parameters of $q(\boldsymbol{u})$. Denoting with $\Lambda = \frac{1}{\sigma^2}K_{Z,Z}^{-1}K_{Z,X}K_{X,Z}K_{Z,Z}^{-1} + K_{Z,Z}^{-1}$,we find that

$$\begin{aligned}\frac{\partial\mathcal{L}_2}{\partial\boldsymbol{m}} &= \frac{1}{\sigma^2}K_{Z,Z}^{-1}K_{Z,X}\boldsymbol{y} - \Lambda\boldsymbol{m} \\ \frac{\partial\mathcal{L}_2}{\partial S} &= \frac{1}{2}S^{-1} - \frac{1}{2}\Lambda.\end{aligned} \tag{6.61}$$

Setting these gradients to zero, we can compute the optimal variational location and scale for the variational Gaussian process given by $S = \Lambda^{-1}$ and $\boldsymbol{m} = \frac{1}{\sigma^2}\Lambda^{-1}K_{Z,Z}^{-1}K_{Z,X}\boldsymbol{y}$. It can be shown that $\mathcal{L}_2$ is less restrictive than equation (6.42). However, at this unique maximum, the bounds are equal.

We use the closed form optimum for the variational location and scale parameters as described above. The key-property of this bound is that we can write this sum as n terms of the input-output pair $\{\boldsymbol{x}_i, y_i\}$. This is necessary to use stochastic gradient methods as discussed in the appendix section 11.14. Thus, using mini-batching, we only need to calculate a number of terms equal to the batch size.

First, we calculate the value m and all of its components in $O(nm + m^3)$ time. Hence, we found the inverses and the Cholesky decompositions of Λ and $K_{Z,Z}$. There exists an exact expression for the KL between two multivariate Gaussians (see appendix lemma (11)) which can be evaluated very fast ($O(m^2)$) using these quantities. Next, using our

previous results, a batch of the bound (6.57) can be evaluated in $O(m^2)$. The storage is again of order $O(m^3 + nm)$. Note that we still need to optimise the inducing points, the kernel hyperparameters (and eventually σ) which can be done by performing stochastic gradient methods.

Notice that, using the optimal variational location and scale, we find that

$$\begin{aligned} q(\boldsymbol{u}|\boldsymbol{y}) &= \mathcal{N}(\boldsymbol{u}|\sigma^{-2}\Lambda^{-1}K_{Z,Z}^{-1}K_{Z,X}\boldsymbol{y}, \Lambda^{-1}) \\ &= \mathcal{N}(\boldsymbol{u}|\sigma^{-2}(\sigma^{-2}K_{Z,Z}^{-1}K_{Z,X}K_{X,Z}K_{Z,Z}^{-1} + K_{Z,Z}^{-1})^{-1}K_{Z,Z}^{-1}K_{Z,X}\boldsymbol{y}, \\ &\qquad\qquad (\sigma^{-2}K_{Z,Z}^{-1}K_{Z,X}K_{X,Z}K_{Z,Z}^{-1} + K_{Z,Z}^{-1})^{-1}) \\ &= \mathcal{N}(\boldsymbol{u}|\sigma^{-2}K_{X^*,Z}\Sigma K_{Z,X}\boldsymbol{y}, K_{Z,Z}\Sigma K_{Z,Z}). \end{aligned} \tag{6.62}$$

which is the same as equation (6.45). Hence we can find the predictive distribution with the same calculation as in equation (6.46), which is again the predictive distribution of the DTC.

The posterior predictive distribution can also be expressed in $\boldsymbol{m}$ and S. For the predictive mean we find that

$$\begin{aligned} \mathbb{E}[q(\boldsymbol{f}^*|\boldsymbol{y})] &= \sigma^{-2}K_{X^*,Z}\Sigma K_{Z,X}\boldsymbol{y} \\ &= K_{X^*,Z}K_{Z,Z}^{-1}\frac{1}{\sigma^2}(\frac{1}{\sigma^2}K_{Z,Z}^{-1}K_{Z,X}K_{X,Z}K_{Z,Z}^{-1} + K_{Z,Z}^{-1})^{-1}K_{Z,Z}^{-1}K_{Z,X}\boldsymbol{y} \\ &= K_{X^*,Z}K_{Z,Z}^{-1}\frac{1}{\sigma^2}\Lambda^{-1}K_{Z,Z}^{-1}K_{Z,X}\boldsymbol{y} \\ &= K_{X^*,Z}K_{Z,Z}^{-1}\boldsymbol{m}^*. \end{aligned} \tag{6.63}$$

We find that

$$\begin{aligned} q(\boldsymbol{f}^*|\boldsymbol{y}) &= \int p(\boldsymbol{f}^*|\boldsymbol{u})q(\boldsymbol{u}|\boldsymbol{y})d\boldsymbol{u} \\ &= \mathcal{N}(K_{X^*,Z}K_{Z,Z}^{-1}\boldsymbol{m}^*, K_{X^*,X^*} - K_{X^*,Z}K_{Z,Z}^{-1}(K_{Z,Z} - SS^t)(K_{X^*,Z}K_{Z,Z}^{-1})^t). \end{aligned} \tag{6.64}$$

A very interesting side road can be taken by using natural gradients (see appendix section 11.16) and section 3.2 in (Hensman et al., 2013) where one recovers the same solutions as before.

Chapter 7

Approximations And Numerical Techniques

In this chapter, we open a can full of numerical techniques which are often applicable on or related to the methods of the previous chapters. However, it will be again a bumpy road where we encounter problems concerning the dimension of the data and the numerical stability of the algorithms. In the end, we will be able to speed-up the calculations with only a little loss in accuracy.

7.1 Structured Kernel Interpolation

In this section, we follow (Wilson and Nickisch, 2015), (Wilson et al., 2015) and (Dong et al., 2017) discussing structured kernel interpolation (SKI). Extra information about the interpolations we use, can be found in the appendix section 11.17. To gather some intuition, we first consider one data point, one dimension and linear interpolation. Suppose that the inducing points z_a and z_b are most close to x_i with $z_a \leq x_i \leq z_b$. Denoting with w_a and $w_b = 1 - w_a$ the linear interpolation weights which represents the relative distances from x_i to z_a and z_b. We can make the following approximation

$$\tilde{k}_{\boldsymbol{x}_i,\boldsymbol{z}_j} = w_a k_{\boldsymbol{z}_a,\boldsymbol{z}_j} + w_b k_{\boldsymbol{z}_b,\boldsymbol{z}_j}. \tag{7.1}$$

Let us go one step further doing a more general interpolation and denote with w_i the interpolation weight corresponding with z_i. In the general case, we find that

$$\tilde{k}_{x_i,\boldsymbol{z}_j} = \sum_{i=1}^{m} w_i k_{\boldsymbol{z}_i,\boldsymbol{z}_j}. \tag{7.2}$$

Hence, denoting with W a $n \times m$ matrix, we can make the approximation

$$K_{X,Z} \approx W K_{Z,Z}. \tag{7.3}$$

If the inducing points are on a rectilinear grid with regular spacing, one can use local cubic interpolation for greater accuracy. For more general grids, inverse distance weighting can be used. If one chooses only 2 non-zero entries per row, we can take as many inducing points as we want, without increasing computation time. Interpolation can be used for essentially every inducing point method. For the SoR (also used in DTC/VFE), we find that

$$K_{X,X}^{\text{SoR}} = Q_{X,X} = K_{X,X}K_{Z,Z}^{-1}K_{Z,X} \approx WK_{Z,Z}K_{Z,Z}^{-1}K_{Z,Z}W^t = WK_{Z,Z}W^t = \tilde{K}_{X,X}, \quad (7.4)$$

The popular FITC/FIC method, we do a diagonal correction of the (SoR). In this case, we find that

$$K_{X,X}^{\text{FIC}} \approx WK_{Z,Z}W^t + \text{diag}(K_{X,X} - WK_{Z,Z}W^t) = \tilde{K}_{X,X}. \quad (7.5)$$

The diagonal term can be computed in $O(n)$ time for sparse W. Notice that we denoted both approximations of the Gram matrices by $\tilde{K}_{X,X}$.

Theorem 12 (MVM structured kernel interpolation (SKI)) *A matrix vector multiplication (MVM) with $\tilde{K}_{X,X}$ as defined before costs $O(n + m^2)$ computations and $O(n + m^2)$ storage for sparse W.*

Proof: We only prove it for the SoR/DTC/VFE, the FIC/FITC is completely analogue. The multiplication of W^t with a vector takes $O(n)$ computation time. The multiplication of $WK_{Z,Z}$ with a vector is of order $O(m^2)$. Hence MVM with $\tilde{K}_{X,X}$ takes $O(m^2 + n)$ computation time. We need to store a $m \times m$ matrix, a sparse $n \times m$ matrix and a $n \times 1$ vector resulting in $O(m^2 + n)$ storage. □

However, for inference, one needs to calculate the inverses and log determinants. The inverse can easily be calculated by the linear conjugate gradients algorithm (see section 11.19) in the appendix) which only requires MVMs. Each iteration in this algorithm requires a single MVM with the matrix $\tilde{K}_{X,X} + \sigma^2 I_n$. Let us denote with $\mu(K_{X,X})$ the time complexity of this MVM. If we need to do p iterations, the conjugate gradient algorithm has time complexity $O(p\mu(K_{X,X}))$. If the matrix is well conditioned, we only need $p \ll n$ iterations in order to obtain a good accuracy. One can take fast MVMs in standard eigenvalue solvers to efficiently compute the log determinant exactly or to use an approximate. We do not discuss these methods since a more clever way will be given in section 7.3. Notice that we do not use a Cholesky product as before.

Using this method we can predict in a very fast way. Suppose we learn $K_{Z,Z}W^t(\tilde{K}_{X,X} + \sigma^2 I)^{-1}\boldsymbol{y}$ during training. The prediction value is given by

$$\hat{f}(X^*) = W^* K_{Z,Z}W^t(\tilde{K}_{X,X} + \sigma^2 I)^{-1}\boldsymbol{y}. \quad (7.6)$$

Hence, we only need to compose a sparse matrix W^* and do a MVM which is $O(1)$.

In the one dimensional case, we can exploit the Toeplitz structure and we only require $O(n + m\log(m))$ computations and $O(n + m)$ storage for a MVM with $\tilde{K}_{X,X}$ by working

on a regular grid. For few dimensions, but bigger than one, we can exploit the Kronecker structure for the inducing points. We only have $O\left(d \cdot m^{\frac{d+1}{d}}\right)$ in runtime and $O(n + dm^{\frac{2}{d}})$ in storage which enabling fast MVMs. Using the structure of the inducing values, this method is called KISS-GP (kernel interpolation for scalable structured Gaussian processes).

For large dimensions, (In paper (Wilson and Nickisch, 2015), this means a dimension greater than 5), we have a lot of problems. One can use the Kronecker structure for the inducing points. However, the curse of dimensionality and the dependence of dimension in storage and runtime makes fast matrix vector products impossible. Notice that, in order to use a grid, the number of inducing points grows exponentially with dimension. One has to use multidimensional interpolation. In other words, we can not make a profit of the placements of the inducing points. For cubic interpolation (rectangular grid!) W has 4^d non-zero entries per row. This can easily be seen by generalising equation (11.38) in the appendix for d-dimensional inputs. In other words, the sparseness is threatened.

Hence, for higher dimensions, one is obliged to use a k-means based method for the inducing points and to use local inverse weighting interpolation. By not using the structure of the inducing points, this method does not result in substantial performance gains.

In (Wilson et al., 2015) one proposes to use projections for dimension reduction. However, we discuss other methods to handle this problem. For the product kernel structure (see equation (3.7)), the interpolation can be done separately for each dimension, which we will exploit in the next section.

7.2 Product Kernel Interpolation For Scalable Gaussian Processes

In this section, we follow (Gardner et al., 2018b). In order to compute the log marginal likelihood of the data, we need to find $(\tilde{K}_{X,X}+\sigma^2 I)^{-1}\boldsymbol{y}$ and $\log(|\tilde{K}_{X,X}+\sigma^2 I|)$. To compute the inverse, we use the method of conjugate gradients (see appendix section 11.19). For approximating log determinants, there exists a technique called stochastic Lanzcos quadrature (see for example (Ubaru et al., 2017)). The stochastic Lanzcos quadrature can be done in $O(p\mu(K_{X,X}))$. We again do not further discuss this technique and postpone the calculation of the determinant to section 7.3 where we avoid explicitly using the Lanczos tridiagonalization algorithm which has storage and numerical stability issues. In short, the log marginal likelihood and predictions can both be done in $O(p\mu(K_{X,X}))$.

We derive a method enabling fast MVMs exploiting the product kernel structure (see equation (3.7)). In this case, the matrix $K_{X,X}$ can be decomposed as $K_{X,X} = K^{(1)}_{X,X} \circ \cdots \circ K^{(d)}_{X,X}$ where $\circ$ represents element-wise multiplication. We denote with $\boldsymbol{v}$ an arbitrary vector of dimension $n \times 1$. From now on, we work with 2 dimensions. The key limitation is that

$$\left(K^{(1)}_{X,X} \circ K^{(2)}_{X,X}\right)\boldsymbol{v} \neq \left(K^{(1)}_{X,X}\boldsymbol{v}\right) \circ \left(K^{(2)}_{X,X}\boldsymbol{v}\right). \tag{7.7}$$

Let us now denote with $D_{\boldsymbol{v}}$ the $n \times n$ diagonal matrix whose elements are $\boldsymbol{v}$. We find that

$$K_{X,X}\boldsymbol{v} = \left(K^{(1)}_{X,X} \circ K^{(2)}_{X,X}\right)\boldsymbol{v} = \text{diag}\left(K^{(1)}_{X,X} D_{\boldsymbol{v}} (K^{(2)}_{X,X})^t\right), \tag{7.8}$$

which equals the first equality in figure (7.1). To compute equation (7.8), we actually need n MVMs with $K^{(1)}_{X,X}$ and $K^{(2)}_{X,X}$. Hence, the time complexity is $O(n\mu(K^{(2)}_{X,X}) + n\mu(K^{(2)}_{X,X}))$

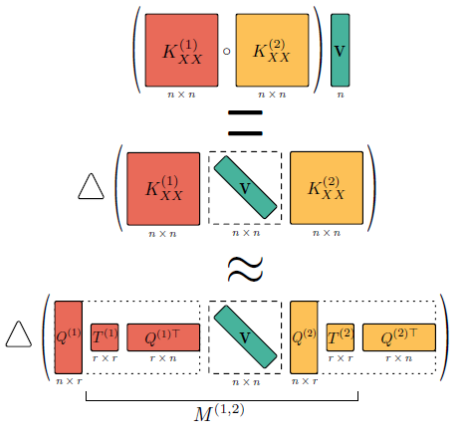


Figure 7.1: Figure from (Gardner et al., 2018b). Illustration MVMs with the product kernel $K^1_{X,X} \circ K^2_{X,X}$.

Now, we use the Lanczos decomposition (see appendix section 11.18). In short, it takes a symmetric $n \times n$ A matrix and probe vector $\boldsymbol{b}$ and returns a Q and T such that $A \approx QTQ^t$ with Q orthogonal. The algorithm is exact after n iterations and requires only 1 MVM per iteration. It has been shown that, by only computing $r < n$ columns of Q, an effective low-rank approximation $Q_{\boldsymbol{r}} T_{\boldsymbol{r}} Q^t_{\boldsymbol{r}}$ of A can be obtained. We obtain the following lemma.

Lemma 3 (Time complexity Lanczos decomposition) *Suppose that MVMs with $K^{(i)}_{X,X}$ can be computed in $O(\mu(K_{X,X}))$ time. Then, the Lanczos decomposition of rank r given by $K^{(i)}_{X,X} \approx Q^{(i)}_{\boldsymbol{r}}(T_{\boldsymbol{r}})^{(i)}(Q^{(i)}_{\boldsymbol{r}})^t$, can be computed in $O(r\mu(K^{(i)}_{X,X}))$ time.*

Next, we also obtain the following lemma.

Lemma 4 (Time complexity of equation (7.8)) *Suppose that $K^{(1)}_{X,X} = Q^{(1)}_{\boldsymbol{r}} T^{(1)}_{\boldsymbol{r}} (Q^{(1)}_{\boldsymbol{r}})^t$ and $K^{(2)}_{X,X} = Q^{(2)}_{\boldsymbol{r}} T^{(2)}_{\boldsymbol{r}} (Q^{(2)}_{\boldsymbol{r}})^t$ with $Q^{(1)}_{\boldsymbol{r}}$, $Q^{(1)}_{\boldsymbol{r}}$ both $n \times r$ matrices and $T^{(1)}_{\boldsymbol{r}}$, $T^{(2)}_{\boldsymbol{r}}$ both $r \times r$ matrices. Then, equation (7.8) can be computed in $O(nr^2)$ time.*

Proof: A visual representation of the proof can be found in the last image in figure (7.1). Both $T^{(1)}_{\boldsymbol{r}}(Q^{(1)}_{\boldsymbol{r}})^t$ and $Q^{(2)}_{\boldsymbol{r}} T^{(2)}_{\boldsymbol{r}}$ can be computed in $O(nr^2)$ time. Multiplying one of the previous two with $D_{\boldsymbol{v}}$ takes $O(nr)$ time and multiplying these two matrices together is of order $O(nr^2)$. Hence the $r \times r$ matrix $M = T^{(1)}_{\boldsymbol{r}}(Q^{(1)}_{\boldsymbol{r}})^t D_{\boldsymbol{v}} Q^{(2)}_{\boldsymbol{r}} T^{(2)}_{\boldsymbol{r}}$ can be computed in $O(nr^2)$ time. We only need the diagonal elements of $K^{(1)}_{X,X} D_{\boldsymbol{v}} (K^{(2)}_{X,X})^t$, hence the necessary outer multiplications only take $O(nr)$ time. □

Combining lemma (3) and lemma (4), we can compute an approximation of equation (7.8) in $O\left(r\mu(K^{(1)}_{X,X}) + r\mu(K^{(2)}_{X,X}) + r^2 n\right)$ time. We can easily extend the previous to product

kernels with many components. Given a kernel matrix $K_{X,X} = K^{(1)}_{X,X} \circ \cdots \circ K^{(d)}_{X,X}$, we define

$$\begin{aligned} \tilde{K}^{(1)}_{X,X} &= K^{(1)}_{X,X} \circ \cdots \circ K^{\left(\frac{d}{2}\right)}_{X,X} \\ \tilde{K}^{(2)}_{X,X} &= K^{\left(\left(\frac{d}{2}\right)+1\right)}_{X,X} \circ \cdots \circ K^{(d)}_{X,X}. \end{aligned} \tag{7.9}$$

Hence, we can write $K_{X,X} = \tilde{K}^{(1)}_{X,X} \circ \tilde{K}^{(2)}_{X,X}$. We apply this splitting recursively (we need $O(\log(d))$ merges if we do it in a parallel way) and obtain theorem (13).

Theorem 13 *Suppose that $K_{X,X} = K^{(1)}_{X,X} \circ \ldots \circ K^{(d)}_{X,X}$, and that computing a MVM with $K^{(i)}_{X,X}$ can be computed in $O(\mu(K^{(i)}_{X,X}))$ time (assume this is the same for all i). Computing a MVM with $K_{X,X}$ requires $O(dr\mu(K^{(i)}_{X,X}) + r^3 n \log(d) + r^2 n)$, with r the rank of the Lanczos decomposition as before.*

Proof: We first need the Lanczos decomposition of the d component matrices which takes $O\left(dr\mu(K^{(i)}_{X,X})\right)$ time. Next, we perform $O(\log(d))$ merges which each takes $O(nr^3)$ time which is the same as lemma (4) but noticing that we now do a matrix matrix multiplication (MMM). Finally MVMs with $K_{X,X}$ takes $O(nr^2)$ time. We find that a single MVM with $K_{X,X}$ takes $O(dr\mu(K^{(i)}_{X,X}) + r^3 n \log(d) + r^2 n)$ time. □

Notice that the term $O(dr\mu(K^{(i)}_{X,X}) + r^3 n \log(d))$ represents the construction time of the Lanczos decomposition. Hence, if we store the decomposition further use, any subsequent MVMs with $K_{X,X}$ takes $O(nr^2)$ time. Now, we use the constructions of section 7.1 about structured kernel interpolation. We apply the approximation

$$K^{(i)}_{X,X} \approx W K_{Z,Z} W^t \tag{7.10}$$

for each component.

Theorem 14 *Using the structured kernel interpolation approximation for the components, the time complexity of constructing a rank r Lanczos decomposition and computing a MVM with $K_{X,X}$ takes $O(dr(n + m\log(m)) + r^3 n \log(d) + r^2 n)$.*

Proof: It follows from the fact that MVMs with $K^{(i)}_{X,X}$ can be computed in $O(n + m\log(m))$ exploiting the Toeplitz structure. □

We now have that the time complexity is linear in dimension, without severely punishing the number of inducing points, which is a huge improvement. We call this method SKIP (structured kernel interpolation for products). However, we can encounter stability issues since we make extensive use of the Lanczos decomposition. Nevertheless, this section was only a foretaste of what is possible. In the next section, we discuss a whole framework which enables a speed-up of almost all the methods we have seen before.

7.3 Blackbox Matrix-Matrix Gaussian Process

In this section, we follow (Gardner et al., 2018a). We discuss a framework called blackbox matrix matrix (BBMM) inference that can be used as a black-box which performs matrix-matrix multiplications with a kernel matrix and its derivative. Denoting with $\hat{K}_{X,X} = \tilde{K}_{X,X} + \sigma^2 I_n$ (also including the exact case), the evidence is given by

$$l(\boldsymbol{\theta}|X, \boldsymbol{y}) = \log p(\boldsymbol{y}|X) = -\frac{1}{2}\boldsymbol{y}^t(\hat{K}_{X,X})^{-1}\boldsymbol{y} - \frac{1}{2}\log|\hat{K}_{X,X}| - \frac{n}{2}\log(2\pi). \tag{7.11}$$

Using lemma (11.63) in the appendix, one can easily find that its derivative towards a hyperparameter θ is given by

$$\frac{dl}{d\theta} = \boldsymbol{y}^t(\hat{K}_{X,X})^{-1}\frac{d\hat{K}_{X,X}}{d\theta}(\hat{K}_{X,X})^{-1}\boldsymbol{y} + \operatorname{Tr}\left((\hat{K}_{X,X})^{-1}\frac{d\hat{K}_{X,X}}{d\theta}\right). \tag{7.12}$$

The derivative is for example necessary in order to use the Adam algorithm (see appendix section 11.15). Hence, in order to train our model, we need fast ways to calculate $(\hat{K}_{X,X})^{-1}\boldsymbol{y}$, $\operatorname{Tr}\left((\hat{K}_{X,X})^{-1}\frac{d\hat{K}_{X,X}}{d\theta}\right)$ and $\log|\hat{K}_{X,X}|$ in an efficient way. Before we are able to continue with our discussion, we need to say a word about preconditioning on the system $(\hat{K}_{X,X})^{-1}\boldsymbol{y}$ (see appendix section 11.20 for more information). Now, the preconditioner needs to satisfy two requirements. First, we need to compute the determinant of this preconditioner matrix P since

$$\log|\hat{K}_{X,X}| = \log|P^{-1}\hat{K}_{X,X}| + \log|P|. \tag{7.13}$$

One can calculate $\log|P^{-1}\hat{K}_{X,X}|$ in a fast way using the CG method which we explain later. Secondly, the calculation of this preconditioner may not dominate the running time, it should only afford a linear time in solves and space. We use a preconditioner called the pivoted Cholesky decomposition. We find the preconditioner $\hat{P}_k = (L_k L_k^t + \sigma^2 I_n)$ with $L_k L_k^t \approx K_{X,X}$. Without going in detail, linear solves of the preconditioner and its log determinant can be computed in $O(nk^2)$ time. The convergence of CG improves exponentially with the rank of the pivoted Cholesky decompostion for RBF-kernels and the number of CG iterations to accurately estimate $\log|\hat{P}^{-1}\hat{K}_{X,X}|$ decreases quickly as k increases.

One uses the Modified Batched Conjugate Gradient (MBCG) Algorithm which performs multiple solves $A^{-1}B = [A^{-1}\boldsymbol{b}_1, \ldots, A^{-1}\boldsymbol{b}_t]$ simultaneously using MMMs. This algorithm also returns the Lanczos tridiagonalisation matrices associated with each of the solves $\tilde{T}_1, \ldots, \tilde{T}_t$ (see appendix section 11.20) which enables fast estimating of the derivative and determinant, which we discuss later. While studying the algorithm, it may be clarifying to compare it with the preconditioned CG algorithm which can be found in the appendix section 11.20. We denote with

$$\frac{(R_{(i)} \circ Z_{(i)})^t I}{(D_{(i)} \circ V_{(i)})^t I} = \left[\frac{[R_{(i)}]_1^t[Z_{(i)}]_1}{[D_{(i)}]_1^t[V_{(i)}]_1}, \ldots, \frac{[R_{(i)}]_t^t[Z_{(i)}]_t}{[D_{(i)}]_t^t[V_{(i)}]_t}\right] \tag{7.14}$$

and with

$$X_{(i)} + \text{diag}(\boldsymbol{\alpha})_{(i)} D_{(i)} = \left[[X_{(i)}]_1, \ldots, [X_{(i)}]_t\right] + \left[[\boldsymbol{\alpha}_{(i)}]_1 [X_{(i)}]_1, \ldots, [\boldsymbol{\alpha}_{(i)}]_t [X_{(i)}]_t\right]. \quad (7.15)$$

The MBCG algorithm can be found looking at algorithm 2, which replaces multiple MVMs with a single MMM allowing to perform multiple solves in parallel which enables GPU acceleration/parallel computing for example. If one is not yet familiar with stochastic trace estimation, it is recommended to read section 11.21 in the appendix.

Algorithm 2 Modified Preconditioned Conjugate Gradient Method (MBCG)

input: mmm_A(), A function for a MMM with A
  B, $n \times t$ matrix to solve against
  $\hat{P}^{-1}()$, a function for preconditioning
output: $A^{-1}B, \tilde{T}_1, \ldots, \tilde{T}_t$
$X_{(0)} = 0$
$R_{(0)} = B$ - $mmm\_A(X_{(0)})$
$D_{(0)} = Z_{(0)} = \hat{P}^{-1}(R_{(0)})$
$\tilde{T}_1, \ldots, \tilde{T}_t = \mathbf{0}$
For $i = 0, \ldots, p$ do :
  $V_{(i)} = mmm\_A(D_{(i)})$
  $\boldsymbol{\alpha}_{(i)} = \dfrac{(R_{(i)} \circ Z_{(i)})^t I}{(D_{(i)} \circ V_{(i)})^t I}$
  $X_{(i+1)} = X_{(i)} + \text{diag}(\boldsymbol{\alpha})_{(i)} D_{(i)}$
  $R_{(i+1)} = R_{(i)} - \text{diag}(\boldsymbol{\alpha})_{(i)} V_{(j)}$
  If $\forall i : ||R_{(i+1)}||_2 < tolerance$, then return $X_{(i+1)}$
  $Z_{(i+1)} = \hat{P}^{-1}(R_{(i+1)})$
  $\boldsymbol{\beta}_{(i)} = \dfrac{(R_{(i+1)} \circ Z_{(i+1)})^t I}{(R_{(i)} \circ Z_{(i)})^t I}$
  $D_{(i+1)} = Z_{(i+1)} + \text{diag}(\boldsymbol{\beta})_{(i+1)} D_{(i)}$
  $\forall j \in (1, \ldots, t) : [\tilde{T}_j]_{i+1,i+1} = \dfrac{1}{[\boldsymbol{\alpha}_{(i)}]_j} + \dfrac{[\boldsymbol{\beta}_{(i)}]_j}{[\boldsymbol{\alpha}_{(i)}]_j}$
  $\forall j \in (1, \ldots, t) : [\tilde{T}_j]_{i,i+1} = [\tilde{T}_j]_{i+1,i} = \dfrac{\sqrt{[\boldsymbol{\beta}_{(i)}]_j}}{[\boldsymbol{\alpha}_{(i)}]_j}$
Return $X_{(i+1)}$ and $\tilde{T}_1, \ldots, \tilde{T}_t$:

Now the preparations are finally done and it is time to do some learning. Let $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_t)^t$ be an array consisting of t independent samples from a random variable Z with mean $\mathbf{0}$ and variance I_n. We execute the MBCG algorithm with input the matrices $\hat{K}_{X,X}$ and $[\boldsymbol{y}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_t]$ to solve against and the preconditioner $\hat{P}_k = (L_k L_k^t + \sigma^2 I_n)$. We obtain $\hat{K}_{X,X}^{-1}\boldsymbol{y}$, $\hat{K}_{X,X}^{-1}\boldsymbol{z}_i$ and $\tilde{T}_i$ (Lanczos with p iterations and probe vector $\boldsymbol{z}_i$) for all $i \in (1, \ldots t)$ as output. Using lemma (13) in the appendix, we find an unbiased estimator of the derivative given by

$$\operatorname{Tr}\left(\hat{K}_{X,X}^{-1}\frac{d\hat{K}_{X,X}}{d\theta}\right) = \mathbb{E}\left[\boldsymbol{z}_i^t\hat{K}_{X,X}^{-1}\frac{d\hat{K}_{X,X}}{d\theta}\boldsymbol{z}_i\right] \approx \frac{1}{t}\sum_{i=1}^{t}(\boldsymbol{z}_i^t\hat{K}_{X,X}^{-1})\left(\frac{d\hat{K}_{X,X}}{d\theta}\boldsymbol{z}_i\right). \tag{7.16}$$

For the determinant, we use the decompositions $\tilde{Q}_1\tilde{T}_1\tilde{Q}_1^t, \ldots, \tilde{Q}_t\tilde{T}_t\tilde{Q}_t^t$ which are low rank approximations of $\hat{K}_{X,X}$. By using lemma (14,15) in the appendix and noticing that $\tilde{Q}_i\boldsymbol{z}_i = \boldsymbol{e}_1$ (since all but the first columns of Q_i are orthogonal to $\boldsymbol{z}_i$), we find that

$$\begin{aligned}\log(|\hat{K}_{X,X}|) &= \operatorname{Tr}(\log(\hat{K}_{X,X})) = \mathbb{E}[\boldsymbol{z}_i^t\log(\hat{K}_{X,X})\boldsymbol{z}_i] \\ &= \mathbb{E}[\boldsymbol{z}_i^t\tilde{Q}_i\log(\tilde{T}_i)\tilde{Q}_i^t\boldsymbol{z}_i] \approx \frac{1}{t}\sum_{i=1}^{t}e_1^t\log(\tilde{T}_i)e_1.\end{aligned} \tag{7.17}$$

The last logarithm can be calculated using the eigendecompostion $\tilde{T}_i = V_i\Lambda_iV_i^t$. We find thus that

$$\log(|\hat{K}_{X,X}|) \approx \sum_{i=1}^{t}e_1^tV_i\log(\Lambda)_iV_i^te_1. \tag{7.18}$$

Now, we discuss the time complexity of all our steps. The matrix $\hat{K}_{X,X}$ has a dimension equal to $n\times n$ and all the other matrices used in the MBCG algorithm $(X_i, R_i, V_i, Z_i, D_i, \hat{P})$ have dimension $n \times t$. Hence, besides the matrix $\hat{K}_{X,X}$, we can store everything in $O(nt)$. All the operations are dominated by the MMM with $\hat{K}_{X,X}$ and a $n\times t$ matrix which takes $O(n^2t)$ time.

Now, in order to calculate equation (7.16), one first needs to calculate the single MMM $\frac{d\hat{K}_{X,X}}{d\theta}[\boldsymbol{z}_1, \ldots \boldsymbol{z}_t]$. For exact GPRs and sparse approximations, the time complexity of a MMM with the derivative of the kernel is approximately equal to the MMM with the kernel. In the exact case, we have a time complexity of $O(n^2t)$. Next we need t inner products between its columns and the columns of $\hat{K}_{X,X}^{-1}[\boldsymbol{z}_1, \ldots \boldsymbol{z}_t]$ which is $O(nt)$. Hence, all of these calculations are dominated by the running time of the MBCG algorithm.

The dimension of the tridiagonal matrices $\tilde{T}_i$ is equal to $p \times p$. Their eigendecomposition can be computed in $O(p^2)$ time. The computation of equation (7.18) only requires $O(tp)$ additional work.

Now we can apply and relate this framework with all the methods that we have seen before. For a Cholesky based exact GP-inference we had a time complexity of $(O(n^3))$. However, we have shown that all the previous calculations have a lower asymptotic complexity of $(O(n^2t))$.

Now, we have that $\hat{K}_{X,X} \approx (K_{X,Z}K_{Z,Z}^{-1}K_{Z,X} + \sigma^2I_n)$ for the SoR/DTC/VFE approximation. Multiplying this kernel with a $n \times t$ matrix takes only $O(tnm + tm^3)$ time which is again asymptoticaly faster (we can use more inducing points) than the $O(nm^2 + m^3)$ using Cholesky based methods. The discussion for FITC/FIC is analogue.

For the SKI, we use theorem (12) and find that a MMM with $\hat{K}_{X,X} \approx (WK_{Z,Z}^{-1}W^t + \sigma^2 I_n)$ and a $n \times t$ matrix requires $O(tn + tm^2)$.

In KISS-GP, we exploit the Toeplitz structure of $K_{Z,Z}$ and a MMM with a $n \times t$ matrix only requires $O(tn + tm \log m)$ time. One can use this framework for training and prediction. We use the BBMM framework for training our data, but predictions are done using the Cholesky decomposition giving more accurate results.

A short extension of this discussion is given in (Wang et al., 2019). Let us partition the dataset $X = [X^{(1)}, \ldots, X^{(p)}]$. One can compute the MVM $\hat{K}_{X,X}\boldsymbol{v}$ into partitions by writing

$$\hat{K}_{X,X}\boldsymbol{v} = [\hat{K}_{X^{(1)},X}\boldsymbol{v}, \ldots, \hat{K}_{X^{(p)},X}\boldsymbol{v}], \tag{7.19}$$

enabling parallel computations. Having reduced the time complexity of exact GPs, one even stated the supremacy over approximate/sparse GPs. However, remember that sparse Gaussian processes enable very fast predictions.

Chapter 8

Dessert: Bayesian And Deep Methods

In this chapter we discuss additional techniques for training our model. We introduce Bayesian techniques sampling hyperparameters which predictions often outperforms the predictions of models optimising the inferior local optimum of the evidence. However, we often need to sacrifice speed and analytical tractability. Next, we discuss methods using neural networks combined with GPR.

8.1 Fully Bayesian GPR

In this section, we follow (Lalchand and Rasmussen, 2019). As has been said before, we come across some difficulties when maximising the marginal likelihood (evidence). The evidence may be non-convex and when there are many hyperparameters it is prone to overfitting which leads to high variability. First, we give an expression for the predictive posterior going full Bayesian. Next, we discuss some Bayesian techniques and how they can be used for Gaussian process regression.

As has been said in section 4.3, we put a prior p over the hyperparameters. We have that

$$\begin{aligned} \boldsymbol{\theta} &\sim p(\boldsymbol{\theta}) \\ \boldsymbol{f}|X, \boldsymbol{\theta} &\sim \mathcal{N}(\mathbf{0}, K_{X,X}) \\ \boldsymbol{y}|\boldsymbol{f} &\sim \mathcal{N}(\boldsymbol{f}, \sigma^2 I_n). \end{aligned} \tag{8.1}$$

We omit the extensive notation including X and $\boldsymbol{x}^*$. By conditioning on the hyperparamters, we find that the predictive distribution is given by

$$p(\boldsymbol{f}^*|\boldsymbol{y}) = \int p(\boldsymbol{f}^*|\boldsymbol{y}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\boldsymbol{y}) d\boldsymbol{\theta} \tag{8.2}$$

Like before, we condition on $\boldsymbol{f}$ and find that

$$\begin{aligned} p(\boldsymbol{f}^*|\boldsymbol{y}) &= \int\int p(\boldsymbol{f}^*|\boldsymbol{f},\boldsymbol{y},\boldsymbol{\theta})p(\boldsymbol{f},\boldsymbol{\theta}|\boldsymbol{y})d\boldsymbol{f}d\boldsymbol{\theta} \\ &= \int\int p(\boldsymbol{f}^*|\boldsymbol{f},\boldsymbol{y},\boldsymbol{\theta})p(\boldsymbol{f}|\boldsymbol{y},\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{y})d\boldsymbol{f}d\boldsymbol{\theta}. \end{aligned} \tag{8.3}$$

Notice that the inner integral

$$p(\boldsymbol{f}^*|\boldsymbol{y},\boldsymbol{\theta}) = \int p(\boldsymbol{f}^*|\boldsymbol{f},\boldsymbol{y},\boldsymbol{\theta})p(\boldsymbol{f}|\boldsymbol{y},\boldsymbol{\theta})d\boldsymbol{f}, \tag{8.4}$$

is equal to equation (6.1) or equation (4.7) where used a bit of a different notation which was more appropriate at that moment. We proved that this is equal to (4.9) and we thus find that

$$p(\boldsymbol{f}^*|\boldsymbol{y},\boldsymbol{\theta}) = \mathcal{N}(K_{X^*,\boldsymbol{x}}(K_{X,X}+\sigma^2 I_n)^{-1}\boldsymbol{y}, K_{X^*,X^*} - K_{X^*,X}(K_{X,X}+\sigma^2 I_n)^{-1}K_{X,X^*}). \tag{8.5}$$

In the chapter about approximate methods, we used some different expressions for this predictive distribution using inducing points (see equation (6.3)). For both methods, one can say that we integrate $\boldsymbol{f}$ out analytically. Thus we know the expressions for $p(\boldsymbol{f}^*|\boldsymbol{y},\boldsymbol{\theta})$ and write

$$p(\boldsymbol{f}^*|\boldsymbol{y}) = \int p(\boldsymbol{f}^*|\boldsymbol{y},\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{y})d\boldsymbol{\theta} \approx \frac{1}{M}\sum_{j=1}^{M} p(\boldsymbol{f}^*|\boldsymbol{y},\boldsymbol{\theta}_j) \quad \boldsymbol{\theta}_j \sim p(\boldsymbol{\theta}|\boldsymbol{y}), \tag{8.6}$$

since the resulting integral is intractable. However, we find the posterior predictive distribution if we can sample $\boldsymbol{\theta}_j$ with distribution $p(\boldsymbol{\theta}|\boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})$. As has been said before, we expect better predictions since optimisation of the evidence often results in an inferior local optimum and is prone to overfitting. However, we now need to solve a sum of predictive distributions over the data with different hyperparameters every time. This results in a predictive time complexity M times worse than before.

We discuss two different cases. First, one can use the Maximum a posteriori (MAP) estimator (see appendix section 11.22) for $\boldsymbol{\theta}$ and the calculation of the sum is not necessary anymore. However, one loses the information concerning the uncertainty of the estimations since the MAP estimator provides a point estimate. Next, one can sample these hyperparameters and calculate equation (8.6). Although there are a lot of sampling algorithms, we focus on Markov chain Monte Carlo (MCMC) sampling (see appendix section 11.23). Denote with $\hat{K}_{X,X} = \tilde{K}_{X,X} + \sigma^2 I_n$, where we also including the exact case. The computational cost of these sampling schemes are dominated by the calculation of $p(\boldsymbol{y}|\boldsymbol{\theta}) \sim \mathcal{N}(\boldsymbol{0}, \hat{K}_{X,X})$ (the evidence), which is equal to

$$\log(p(\boldsymbol{y}|\boldsymbol{\theta})) = -\frac{1}{2}\log|\hat{K}_{X,X}| - \frac{1}{2}\boldsymbol{y}^t(\hat{K}_{X,X})\boldsymbol{y} - \frac{n}{2}\log(2\pi). \tag{8.7}$$

Hence, we are actually dealing with the same problem as before, which was the inversion and the calculation of the determinant of the covariance matrix with has an $O(n^3)$ time complexity for exact GPR and $O(nm^2 + m^3)$ for sparse approximations. Hence, inducing point methods can come at handy reducing the time complexity. We use Metropolis Hasting and an improvement of the Hamiltonian Monte Carlo (HMC) called the no U-turn sampler (NUTS). More information can be found in the appendix section 11.23.

In a lot of papers like (Filippone et al., 2013) and (Murray and Adams, 2010), one treats the underlying function values $\boldsymbol{f}$ as latent variables and one thus also samples $\boldsymbol{f}$. We will not elaborate on this in this section although variants of MH, HMC and slice sampling have been specifically proposed for sampling $\boldsymbol{f}$ in these papers. Interested readers are encouraged to read the next section, where we discuss a method sampling from the optimal variational posterior. In our code, we merely focus on sampling $\boldsymbol{\theta}$. For the Metropolis-Hastings algorithm we can for example take a multivariate Gaussian as proposal distribution. We also use NUTS (extension of HMC) in order to sample from $p(\boldsymbol{\theta}|\boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})$.

8.2 Bayesian Variational Method

This and the next section are merely added for completeness. We discover connections with the previous methods and try to obtain more tools concerning GPR. However, all the methods we discuss from now on will not be included in our data study which means these parts can be skipped. We also omit a lot of the technical details. We begin with a Bayesian approach for variational Gaussian processes discussed in (Hensman et al., 2015).

Like in the previous section, we start with equation (6.1) and equation (6.3). However, we do not condition further and use the notation $q(\boldsymbol{u}, \boldsymbol{\theta})$. We find that

$$p(\boldsymbol{f}^*|\boldsymbol{y}) = \int\int p(\boldsymbol{f}^*|\boldsymbol{f}, \boldsymbol{y}, \boldsymbol{\theta})p(\boldsymbol{f}, \boldsymbol{\theta}|\boldsymbol{y})d\boldsymbol{\theta}d\boldsymbol{f} \tag{8.8}$$

$$q(\boldsymbol{f}^*) = \int\int p(\boldsymbol{f}^*|\boldsymbol{u}, \boldsymbol{\theta})q(\boldsymbol{u}, \boldsymbol{\theta})d\boldsymbol{\theta}d\boldsymbol{u}. \tag{8.9}$$

Notice that $\boldsymbol{f}, \boldsymbol{u}, \boldsymbol{\theta}$ are free parameters of the model. We mimic the calculations of equation (6.55) which are actually the same. In the variational framework we always want to minimise the KL-divergence between the approximate and the true posteriors which is given by

$$0 \geq \mathcal{K} = -\mathrm{KL}[q(\boldsymbol{f}, \boldsymbol{u}, \boldsymbol{\theta})||p(\boldsymbol{f}, \boldsymbol{u}, \boldsymbol{\theta}|\boldsymbol{y})] = \mathbb{E}_{q(\boldsymbol{f},\boldsymbol{u},\boldsymbol{\theta})}\left[\log\left(\frac{p(\boldsymbol{f}, \boldsymbol{u}, \boldsymbol{\theta}|\boldsymbol{y})}{q(\boldsymbol{f}, \boldsymbol{u}, \boldsymbol{\theta})}\right)\right]. \tag{8.10}$$

By simplifying the fracture and by conditioning, we can rewrite this expression as

$$\mathcal{K} = \mathbb{E}_{q(\boldsymbol{f},\boldsymbol{u},\boldsymbol{\theta})}\left[\log\left(\frac{p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{u}, \boldsymbol{\theta})p(\boldsymbol{u}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{q(\boldsymbol{u}, \boldsymbol{\theta})}\right)\right] - \log\left(p(\boldsymbol{y})\right). \tag{8.11}$$

We denote with C an intractable constant normalising the optimal variational distribution. Including this constant, writing down the expectation and some conditioning we find that

$$\mathcal{K} = -\mathrm{KL}\left[q(\boldsymbol{u},\boldsymbol{\theta})||\frac{p(\boldsymbol{u}|\boldsymbol{\theta})p(\boldsymbol{\theta})\exp\left(\mathbb{E}_{p(\boldsymbol{f}|\boldsymbol{u},\boldsymbol{\theta})}[\log\left(p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{u},\boldsymbol{\theta})\right]\right)}{C}\right] - \log\left(C\right) + \log\left(p(\boldsymbol{y})\right). \tag{8.12}$$

It may be interesting to compare this equation and the $\mathcal{L}_2$ bound (equation 6.53). By omitting the $\boldsymbol{\theta}$ parameter and the constant C, the KL term is the same expression. By minimising the KL-divergence we find that the optimal variational distribution equals

$$\log\left(\hat{q}(\boldsymbol{u},\boldsymbol{\theta})\right) = \mathbb{E}_{p(\boldsymbol{f}|\boldsymbol{u},\boldsymbol{\theta})}[\log\left(p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{u},\boldsymbol{\theta})\right] + \log\left(p(\boldsymbol{u}|\boldsymbol{\theta})\right) + \log\left(p(\boldsymbol{\theta})\right) - \log\left(C\right). \tag{8.13}$$

The computation of this expression $O(nm^2+m^3)$ but it factorises across the data enabling scalability by using batches. We make a small detour following (Murray and Adams, 2010). The variables $\boldsymbol{u}$ and $\boldsymbol{\theta}$ are strongly coupled and alternately fixing these variables will not work. We propose a reparametrisation making the parameters independent under the prior. We draw a vector $\boldsymbol{v} \sim \mathcal{N}(\boldsymbol{0}, I_m)$. Doing a Cholesky decomposition $K_{Z,Z} = RR^t$, we find that $\boldsymbol{u} = R\boldsymbol{v}$. Hence, we can rewrite the optimal variational distribution

$$\log\left(\hat{q}(\boldsymbol{u},\boldsymbol{\theta})\right) = \mathbb{E}_{p(\boldsymbol{f}|\boldsymbol{u}=R\boldsymbol{v})}[\log\left(p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{u},\boldsymbol{\theta})\right] + \log\left(p(\boldsymbol{v})\right) + \log\left(p(\boldsymbol{\theta})\right) - \log\left(C\right). \tag{8.14}$$

Now, for fixed $\boldsymbol{v}$, we can choose to update $\boldsymbol{\theta}$ which implicitly updates $\boldsymbol{u}$. In (Hensman et al., 2015), one discusses a method using HMC sampling the $\boldsymbol{v}$ and $\boldsymbol{\theta}$ jointly.

8.3 Deep Methods For Gaussian Processes

Deep learning is a family of machine learning methods based on neural networks. Readers which are not yet familiar with neural networks can read (Nielsen, 2015) which is a very basic yet understandable introduction. For completeness, we give a short outline of deep methods for GPs again omitting details and practical implementations.

In (Wilson et al., 2016a), one introduces GP models that use kernels parametrized by neural networks called deep kernel learning (DKL). Starting from a base kernel $k(\boldsymbol{x}_i,\boldsymbol{x}_j|\boldsymbol{\theta})$, we transform the input as

$$k(\boldsymbol{x}_i,\boldsymbol{x}_j|\boldsymbol{\theta}) \rightarrow k(g(\boldsymbol{x}_i,\boldsymbol{w}),g(\boldsymbol{x}_j,\boldsymbol{w})|\boldsymbol{w},\boldsymbol{\theta}). \tag{8.15}$$

Here, $g(\boldsymbol{x},\boldsymbol{w})$ is a non-linear mapping given by a deep architecture capturing hierarchical structure and non-stationary. The structure of the model for deep kernel learning can be found in figure (8.1). It is a Gaussian process with d-dimensional inputs through L

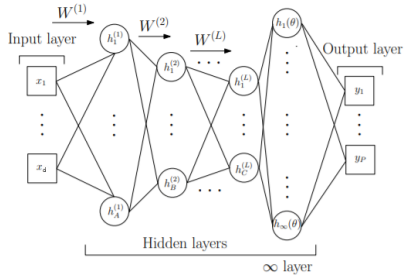


Figure 8.1: Figure from (Wilson et al., 2016a). DKL also including multidimensional outputs.

hidden layers followed by a hidden layer with infinite basis functions including the base kernel hyperparameters $\boldsymbol{\theta}$.

The parameters $\boldsymbol{\theta}$ and $\boldsymbol{w}$ can be learned by maximising the evidence (equation 8.7). Of course, inducing points methods are also possible by replacing the covariance matrix and modifying the evidence.

One year later, one proposed in (Wilson et al., 2016b) a method for DKL using a stochastic variational approach which allows stochastic training spaces but the paper focusses on classification.

In (Damianou and Lawrence, 2013), one introduces the concept of a Deep Gaussian process (DGP). Now, a DGP defines a prior recursively on $\boldsymbol{f}^1, \ldots, \boldsymbol{f}^L$. The prior is an independent GP in each dimension. The outputs at layer l are $\boldsymbol{f}^l$ and the inputs are $\boldsymbol{f}^{l-1}$ with some Gaussian noise included. In (Damianou and Lawrence, 2013), one forces the the input of each layer to be independent of the output of the previous layer. We discuss the paper of (Salimbeni and Deisenroth, 2017), not forcing independence or Gaussian form between the layers, sacrificing analytical tractability. We denote with Z^l and $\boldsymbol{u}^l$ the inducing locations and values at layer l. A DGP has joint density

$$p(\boldsymbol{y}, \{\boldsymbol{f}^l, \boldsymbol{u}^l\}_{l=1}^L) = \prod_{i=1}^{n} p(\boldsymbol{y}_i|\boldsymbol{f}_i^L) \prod_{i=1}^{L} p(\boldsymbol{f}^l|\boldsymbol{u}^l, \boldsymbol{f}^{l-1}, Z^{l-1}) p(\boldsymbol{u}^l, Z^{l-1}), \tag{8.16}$$

with $\boldsymbol{f}^0 = X$. They propose a posterior having the property to maintain the exact model conditioned on U^l, that is factorized between layers (and dimension) and that $q(U^l)$ is a Gaussian with mean $\boldsymbol{m}^l$ and variance S^l. Hence, the posterior has the following form

$$q(\{\boldsymbol{f}^l, U^l\}_{l=1}^L) = \prod_{l=1}^{L} p(\boldsymbol{f}^l|\boldsymbol{u}^l, \boldsymbol{f}^{l-1}, Z^{l-1}) q(\boldsymbol{u}^l). \tag{8.17}$$

Next, we can again find the evidence lower bound minimising this approximate and the true posterior. We have already done similar calculations two times thus we omit it here. We find that the evidence lower bound of this DGP is

$$\begin{aligned}\mathcal{L}_{DGP} &= -\mathrm{KL}[q(\{\boldsymbol{f}^l, \boldsymbol{u}^l\}_{l=1}^L)||p(\boldsymbol{y}, \{\boldsymbol{f}^l, \boldsymbol{u}^l\}_{l=1}^L)] \\ &= \sum_{i=1}^n \mathbb{E}_{q(\boldsymbol{f}_i^L}[\log(p(\boldsymbol{y}_i|\boldsymbol{f}_i^L)] - \sum_{l=1}^L \mathrm{KL}[q(\boldsymbol{u}^l)||p(\boldsymbol{u}^l, Z^{l-1}].\end{aligned} \tag{8.18}$$

We can approximate the equation using Monte Carlo sampling from $q(\hat{\boldsymbol{f}}_i^L)$ which we discuss in a second. Notice that the bound factorises over the data, enabling scalability through batching data.

Now, we use equation (6.64) where we replace $\boldsymbol{f}^*$ by $\boldsymbol{f}$ and denote with $q(\boldsymbol{f}|\boldsymbol{m}, S, X, Z) = \mathcal{N}(\boldsymbol{f}|\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$. After marginalising the inducing values of each layer, we find that

$$q(\{\boldsymbol{f}^l\}_{i=1}^L) = \prod_{i=l}^L q(\boldsymbol{f}^l|\boldsymbol{m}^l, S^l, F^{l-1}, Z^{l-1}) = \prod_{i=1}^L \mathcal{N}(\boldsymbol{f}^l|\tilde{\boldsymbol{\mu}}, \tilde{\Sigma}). \tag{8.19}$$

In (Salimbeni and Deisenroth, 2017), one uses a non-zero mean function for the inner layers. Notice that $\boldsymbol{f}_i^L$ only depends on $\boldsymbol{f}_i^{L-1}$ which only depends on $\boldsymbol{f}_i^{L-2}$ and so on. Since, $q(\{\boldsymbol{f}^l\}_{i=1}^L)$ is a fully factorised Gaussian we can apply the reparamisation trick (see (Kingma et al., 2015)) sampling only using univariate Gaussians. Hence we receive samples $\hat{\boldsymbol{f}}_i^l \sim q(\boldsymbol{f}_i^l|\boldsymbol{m}^l, S^l, \boldsymbol{f}_i^{l-1}, Z^{l-1})$

Next, to obtain the density over $\boldsymbol{f}^{(L)*}$, we use the Gaussian mixture

$$q(\boldsymbol{f}^{(L)*}) = \frac{1}{M}\sum_{j=1}^M q(\boldsymbol{f}^{(L)*}|\boldsymbol{m}^L, S^L, \boldsymbol{f}_{(j)}^{(L-1)*}, Z^{L-1}). \tag{8.20}$$

More information and further model details can be found in the source.

Chapter 9

Gaussian Process Regression in Finance

In this section, we try to train a model which is able to price options. All the simulations were done on an Asus computer with Intel core i7 processor, 8GB ram and one Nvidia Geforce 920mx graphic card. It is important to take into account possible background processes which could have slowed down some calculations.

Before continuing, some special attention is given to (Abadi et al., 2016), (Salvatier et al., 2016) and (Gardner et al., 2018a), which represent the packages TensorFlow, PyMC3 and GPyTorch in Python. When one of these packages is used, we use the abbreviations 'tf', 'pym' and 'gpy'. Using GPyTorch, we included the possibility for GPU-accelerated computing using cuda in the code. Since we do not have multiple GPUs, we were not able to test additional parallelism distributing the kernel matrix across multiple GPUs. The code and datasets which are used for the data study can be found on github: https://github.com/2290Veusseleir/Thesis-Machine-Learning-In-Finance.

We apply the parameter space of the models which was used in (De Spiegeleer et al., 2018) for vanilla and barrier options. For American options we discuss a bigger parameter space. We use two measures quantifying the quality of pricing/prediction. The first quantifies the maximum absolute error and the second the average absolute error

$$\text{MAE} = \max\left[|\boldsymbol{y}_i - \boldsymbol{f}_i^*|, i \in (1, \ldots, n)\right] \tag{9.1}$$

$$\text{AAE} = \frac{1}{n}\sum\nolimits_{i=1}^{n} |\boldsymbol{y}_i - \boldsymbol{f}_i^*|. \tag{9.2}$$

We train the models on 1000, 4000 and (if feasible) 10000 points. Next, we predict 1000 new data points and calculate their MAE and AAE. We always take the mean of 10 different prediction time measurements in order to diminish the effects of background processes. We take the maximum of zero and the predicted value since the price of an option can-not be negative. We always mention the number of inducing points we used. A clear scheme of all the models and their abbreviations we discussed can be found in chapter (10). We programmed a lot of different regression and financial models and which

can price a lot of options. Comparing all these regression models using different options and financial models will create a mess. Hence, we made a selection.

Models we will not discuss are FITC and VFE with training the locations of the inducing points by optimising the evidence, SKIP, SVG using gpytorch and exact GPR and VFE going full Bayesian. We did not include the SKIP method since it suffers from tuning and stability issues using big ill-conditioned matrices due to the extensive use of the Lanczos decomposition in training and predicting. For small datasets, we do not encounter problems using this method. We also do not include the full Bayesian methods in our competitive data study since they are too slow. However using these methods, we obtain information concerning the uncertainties of the hyperparameters. Some interesting plots using the full Bayesian methods for vanilla calls in the Heston model are given at the end of this chapter.

We omit the pricing of options using the Variance gamma model and the MAE and AAE of the training set. Suitable parameter ranges for the VG model can be found in (De Spiegeleer et al., 2018). Interested readers can run and study the code for these cases themselves. Notice that we need to make a lot of assumptions, i.e. the number of iterations and the learning rate for numerical/stochastic methods. This means that results can differ a little when you run the code yourself. All the parameter ranges can be found in appendix section 11.24.

9.1 Pricing Vanilla Options Heston Model

We start with pricing vanilla call options in the Heston model. The parameter ranges used for training and validation can be found in table 11.1. All the parameters are sampled uniformly except v_0, which is sampled from a bounded exponential distribution to incorporate more small volatilities in the training set. For testing, notice that we shrink the parameter space a little in order to avoid the edges.

First, we calculate the options using the fast Fourier transform FFT algorithm (which we did not parallelize). Then we train a polynomial regression (PR) model which can be seen as our point of comparison. Notice that the function $(S_T - K)^+$ (see equation (2.1)), has a hockey stick shape if we plot it against K. Since the strike is a very important parameter, a quadratic model can give decent results. Next, we train the standard GPR model without any modifications and a fixed noise variance σ^2. Subsequently, we train the FITC and VFE using k-means for the inducing points and a fixed noise variance σ^2, which appears to be very good approximation models. In these implementations, we do not further train the inducing points and the noise variance. Thereafter, we train the SVG-model which initialises the inducing points with k-means and trains them further. We decided to fix the noise variance. We continue with training the standard GPR and VFE model using BBMMs and training basic GPR and VFE with MAP determining the place of the inducing points using k-means and training the noise variance. For the MAP methods, we include a prior on the hyperparameters based on our past experiences pushing the optimisation in the right direction. The outcomes can be found in table 9.1.

When looking at the results in a few seconds, pay attention to the increase in computation time for the different methods when increasing the number of data/inducing points and

compare them with the theoretical time complexities. Next, notice that we can almost exactly calculate the values of vanillas in the Heston (and Variance gamma model). The only approximation we make is the Simpson's rule to evaluate an integral (see appendix section 11.3). Hence, we predict using noise-free observations, which means that we need to take the Gaussian noise variance σ^2 as close as possible to zero. Doing this, we can encounter problems concerning the inversion of the covariance matrix K of the GPR. For algorithms training the noise variance, we sometimes need to include some jitter on the diagonal of this matrix in order to preserve positive definiteness. We also calculate the MAE and the AAE for all models, quantifying their predictive ability. Our main goal was to calculate the price of an option faster. Hence, the prediction time is also very important. A visualisation of the results of most of the models can be found in figures 9.3, 9.4 and 9.5. This can come in handy when comparing the the models which each other and their corresponding theoretical time complexities. In figures 9.1 and 9.2, one can also compare the different error distributions of exact GPR and GPR using the BBMM framework using 1000 data-points for both training the model and predictions.

We first discuss the exact GPR models. For these models, we obtain very accurate predictions. However, the time complexity of $O(n^3)$ gives us problems using large datasets. Using the standard GPR model using BBMM, there is a trade-off between the computation time $O(n^2)$ and some accuracy. However, the BBMM enables the use of more data points, and very accurate predictions can be achieved. The prediction time for the exact methods is $O(n)$. The very fast prediction time of $O(m)$ is the main advantage of using inducing point methods. Since the time complexity of standard inducing point methods is still $O(nm^2+m^3)$, it may be appropriate to use stochastic techniques or to use the BBMM framework decreasing the computation time dramatically for a presumably small increase in prediction error. The methods using MAP give somewhat the same results as the corresponding exact GPR and sparse methods. The differences are due to the use of a prior for the hyperparameters, the training of the noise variance and the different implementation.

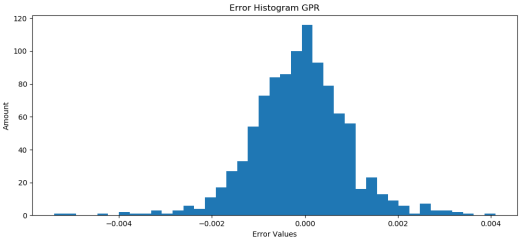


Figure 9.1: Error histogram GPR ($n = 1000$).

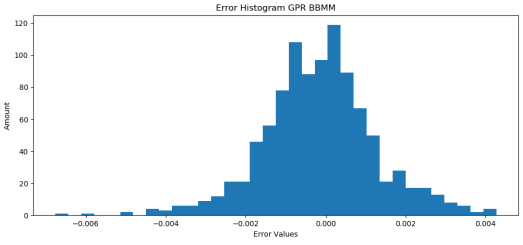


Figure 9.2: Error histogram GPR with BBMM ($n = 1000$).

	Training time	Prediction time	MAE	AAE
FFT(n=1000/4000/10000)	/	5.1165	0	0
PR(n=1000)	0.0076	0.0017	0.0272	0.00575
PR(n=4000)	0.0187	0.0017	0.0284	0.00563
PR(n=10000)	0.0293	0.0018	0.0273	0.00570
—	—	—	—	—
GPR(n=1000)	7.9999	0.0542	0.0054	0.00077
GPR(n=4000)	507.78	0.1803	0.0030	0.00040
FITC(n=1000)(m=200)	10.722	0.0082	0.0136	0.00267
FITC(n=4000)(m=200)	53.628	0.0083	0.0112	0.00208
FITC(n=4000)(m=400)	107.62	0.0159	0.0098	0.00155
VFE(n=1000)(m=200)	10.002	0.0084	0.0090	0.00181
VFE(n=4000)(m=200)	68.471	0.0082	0.0089	0.00162
VFE(n=4000)(m=400)	155.48	0.0171	0.0068	0.00112
SVG(n=1000)(m=200)(tf)	5.7177	0.0084	0.0109	0.00203
SVG(n=4000)(m=200)(tf)	10.981	0.0085	0.0115	0.00196
SVG(n=10000)(m=400)(tf)	49.520	0.0173	0.0100	0.00160
GPR(n=1000)(BBMM)(gpy)	2.1939	0.0404	0.0068	0.00108
GPR(n=4000)(BBMM)(gpy)	21.433	0.1680	0.0061	0.00082
GPR(n=10000)(BBMM)(gpy)	138.54	0.4430	0.0051	0.00063
VFE(n=1000)(m=200)(BBMM)(gpy)	5.3466	0.0085	0.0115	0.00214
VFE(n=4000)(m=200)(BBMM)(gpy)	24.525	0.0082	0.0109	0.00183
VFE(n=10000)(m=400)(BBMM)(gpy)	104.27	0.0168	0.0139	0.00173
GPR(n=1000)(MAP)(pym)	25.421	0.0448	0.0053	0.00078
GPR(n=4000)(MAP)(pym)	1005.2	0.1711	0.0021	0.00027
VFE(n=1000)(m=200)(MAP)(pym)	14.959	0.0083	0.0131	0.00246
VFE(n=4000)(m=200)(MAP)(pym)	23.426	0.0081	0.0126	0.00233

Table 9.1: Training and predicting vanilla calls using the Heston model.

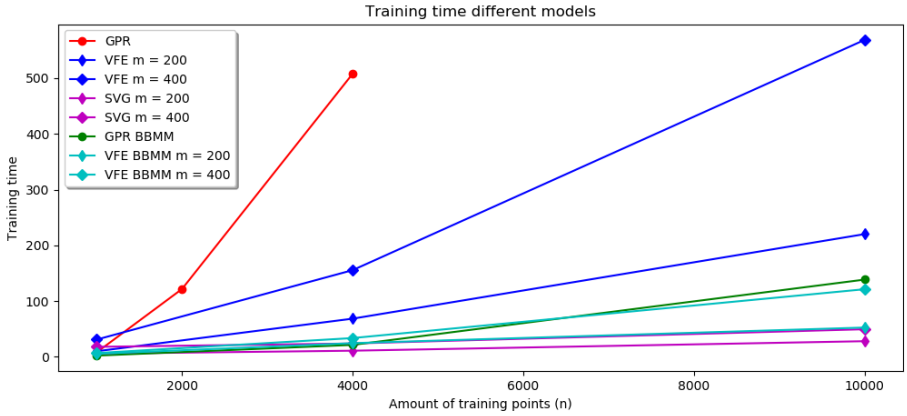


Figure 9.3: Training time vanilla calls for different models.

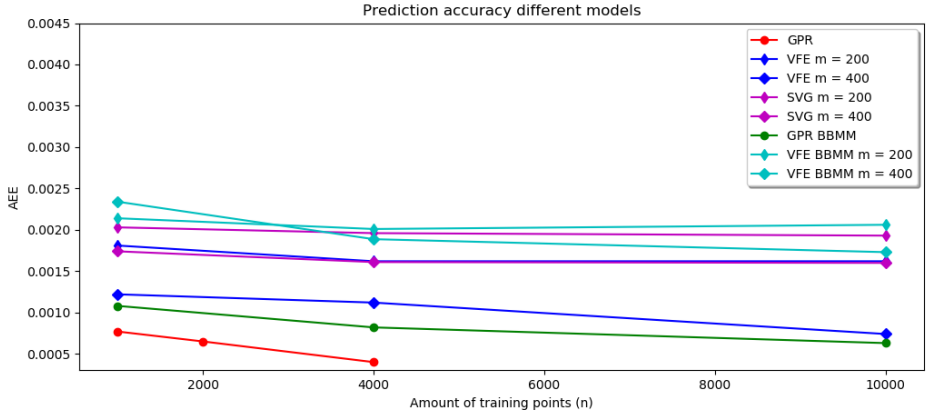


Figure 9.4: Prediction accuracy vanilla calls for different models.

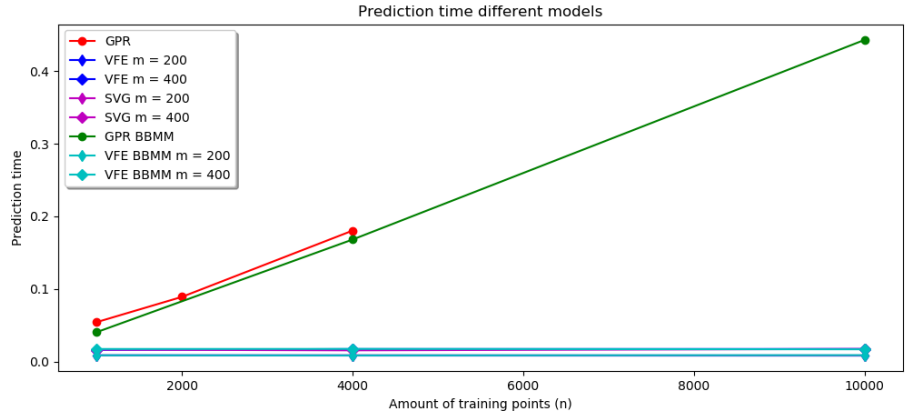


Figure 9.5: Prediction time vanilla calls for different models. The prediction times of inducing point methods almost coincide.

9.2 Pricing Barrier Options Heston Model

In the next data study, we train models which are able to predict DOBP options using the Heston model. The corresponding parameter ranges can be found in table 11.2. Often, algorithms pricing these derivatives use Monte-Carlo simulation and are very slow. We take the opportunity to test how our models cope with noisy data and outliers. In our training set, we only simulate 1000(!) different paths with time steps of 1 day. Hence, the value of the DOBP corresponding to this simulation will be quite noisy. Next, we test our model using 1000 accurate test points which we priced simulating 100000 different paths with time steps of 1 day. The results can be found in table 9.2.

We already said that pricing using Monte-Carlo simulations takes a very long time. Hence, using GPR for pricing difficult/exotic options can be very advantageous. For example, predicting 1000 points using VFE with the BBMM framework is $93393/0.0072 = 12971250$ times faster than pricing the option with Monte-Carlo simulations using 100000 different paths, where we again omitted any form of parallelisation. This simulation tests how our models can handle noisy data. Although the difference in computation time between exact and approximate models stays the same, the difference between their predictive ability is less severe as before. For implementations for (sparse) GPR without training the noise variance, one needs to choose the noise variance. It may be appropriate to choose an higher value as before since we work with noisy observations. For a discussion comparing FITC and VFE, their shortfalls and understanding the value of σ^2 in these models, see the end of section 6.2. Using SVG and VFE with BBMM, we obtain fast and accurate predictions with fast training. A lot of the derivatives with high barrier have zero price in our dataset or are very close to zero.

Using less noisy data, all the models obtain better prediction values. This becomes clear in our next data-study where we try to price DIBP options using 100000 Monte-Carlo simulations for training and prediction which both give accurate results. We use 1000 test observations and the same parameter space as for DOBP options which can be found in table 11.2. The results are displayed in table 9.3 and are better than the results for DOBP options.

	Training time	Prediction time	MAE	AAE
MC(n=1000/4000/10000)	/	93390	0	0
PR(n=1000)	2.3607	0.0016	0.0951	0.0107
PR(n=4000)	0.0175	0.0016	0.1123	0.0107
PR(n=10000)	0.4411	0.0017	0.1158	0.0110
—	—	—	—	—
GPR(n=1000)	6.3609	0.0415	0.0659	0.0045
GPR(n=4000)	298.93	0.1663	0.0531	0.0032
FITC(n=1000)(m=200)	9.876	0.0077	0.0771	0.0056
FITC(n=4000)(m=200)	43.415	0.0080	0.0578	0.0047
FITC(n=4000)(m=400)	129.60	0.0151	0.0662	0.0042
VFE(n=1000)(m=200)	8.574	0.0083	0.0753	0.0055
VFE(n=4000)(m=200)	69.801	0.0082	0.0606	0.0045
VFE(n=4000)(m=400)	131.454	0.0169	0.0529	0.0041
SVG(n=1000)(m=200)(tf)	8.4615	0.0084	0.0622	0.0051
SVG(n=4000)(m=200)(tf)	11.4329	0.0079	0.0569	0.0046
SVG(n=10000)(m=400)(tf)	28.549	0.0163	0.0639	0.0048
GPR(n=1000)(BBMM)(gpy)	2.8926	0.0456	0.0599	0.0045
GPR(n=4000)(BBMM)(gpy)	35.613	0.1679	0.1081	0.0046
GPR(n=10000)(BBMM)(gpy)	286.46	0.4524	0.0910	0.0032
VFE(n=1000)(m=200)(BBMM)(gpy)	7.6467	0.0080	0.0688	0.0055
VFE(n=4000)(m=200)(BBMM)(gpy)	29.860	0.0076	0.0603	0.0048
VFE(n=10000)(m=400)(BBMM)(gpy)	114.03	0.0161	0.0626	0.0041
GPR(n=1000)(MAP)(pym)	31.070	0.0445	0.0511	0.0039
GPR(n=4000)(MAP)(pym)	1005.8	0.1812	0.0368	0.0028
VFE(n=1000)(m=200)(MAP)(pym)	24.201	0.0761	0.0734	0.0060
VFE(n=4000)(m=200)(MAP)(pym)	28.153	0.0742	0.0751	0.0060

Table 9.2: Training and predicting DOBP options using the Heston model using noisy observations.

	Training time	Prediction time	MAE	AAE
Tree(n=1000/4000/10000)	/	93390	0	0
PR(n=1000)	0.0083	0.0019	0.0577	0.01078
PR(n=4000)	0.0684	0.0019	0.0519	0.01029
PR(n=10000)	0.5833	0.0020	0.0518	0.01049
—	—	—	—	—
GPR(n=1000)	8.0849	0.0444	0.0178	0.00187
GPR(n=4000)	273.02	0.1608	0.0176	0.00105
FITC(n=1000)(m=200)	9.2305	0.0080	0.0350	0.00381
FITC(n=4000)(m=200)	50.716	0.0080	0.0372	0.00357
FITC(n=4000)(m=400)	100.85	0.0171	0.0301	0.00275
VFE(n=1000)(m=200)	8.2476	0.0082	0.0292	0.00354
VFE(n=4000)(m=200)	56.746	0.0078	0.0342	0.00322
VFE(n=4000)(m=400)	157.92	0.0162	0.0209	0.00212
SVG(n=1000)(m=200)(tf)	7.0491	0.0081	0.0330	0.00328
SVG(n=4000)(m=200)(tf)	14.827	0.0159	0.0235	0.00292
SVG(n=10000)(m=400)(tf)	59.484	0.0234	0.0153	0.00210
GPR(n=1000)(BBMM)(gpy)	2.5716	0.0433	0.0175	0.00211
GPR(n=4000)(BBMM)(gpy)	22.628	0.1700	0.0166	0.00153
GPR(n=10000)(BBMM)(gpy)	217.29	0.4166	0.0093	0.00100
VFE(n=1000)(m=200)(BBMM)(gpy)	4.7483	0.0078	0.0355	0.00418
VFE(n=4000)(m=200)(BBMM)(gpy)	17.725	0.0082	0.0307	0.00352
VFE(n=10000)(m=400)(BBMM)(gpy)	135.65	0.0169	0.0197	0.00264
GPR(n=1000)(MAP)(pym)	59.982	0.0421	0.0211	0.00183
GPR(n=4000)(MAP)(pym)	669.25	0.1759	0.0179	0.00091
VFE(n=1000)(m=200)(MAP)(pym)	20.463	0.0080	0.0377	0.00426
VFE(n=4000)(m=200)(MAP)(pym)	25.874	0.0084	0.0321	0.00412

Table 9.3: Training and predicting DIBP options.

9.3 Pricing American Options

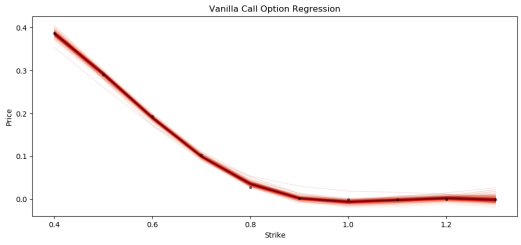
For our final data study, we try to price American put options using a binomial tree model with steps of 0.33 days for training and predicting. This means that we approximate the price by assuming that the buyer can execute the American option three times in a day which is actually very accurate but very time consuming to calculate. The parameter ranges can be found in table 11.3. We use 1000 test points. We can conclude the same as before.

	Training time	Prediction time	MAE	AAE
Tree(n=1000/4000/10000)	/	$44 \cdot 10^4$	0	0
PR(n=1000)	0.0048	0.0013	0.0603	0.01066
PR(n=4000)	0.0225	0.0014	0.0570	0.01046
PR(n=10000)	0.4473	0.0017	0.0577	0.01037
—	—	—	—	—
GPR(n=1000)	18.027	0.0410	0.0063	0.00041
GPR(n=4000)	376.68	0.1649	0.0040	0.00023
FITC(n=1000)(m=200)	13.258	0.0080	0.0106	0.00083
FITC(n=4000)(m=200)	69.662	0.0075	0.0115	0.00080
FITC(n=4000)(m=400)	151.49	0.0158	0.0084	0.00063
VFE(n=1000)(m=200)	8.8852	0.0077	0.0091	0.00084
VFE(n=4000)(m=200)	49.050	0.0081	0.0092	0.00081
VFE(n=4000)(m=400)	128.36	0.0162	0.0071	0.00057
SVG(n=1000)(m=200)(tf)	7.5439	0.0075	0.0137	0.00096
SVG(n=4000)(m=200)(tf)	9.6718	0.0084	0.0105	0.00088
SVG(n=10000)(m=400)(tf)	48.616	0.0166	0.0114	0.00086
GPR(n=1000)(BBMM)(gpy)	5.0072	0.0398	0.0095	0.00076
GPR(n=4000)(BBMM)(gpy)	21.593	0.1588	0.0062	0.00045
GPR(n=10000)(BBMM)(gpy)	129.67	0.5087	0.0057	0.00037
VFE(n=1000)(m=200)(BBMM)(gpy)	6.5637	0.0082	0.0142	0.00153
VFE(n=4000)(m=200)(BBMM)(gpy)	32.889	0.0085	0.0153	0.00121
VFE(n=10000)(m=400)(BBMM)(gpy)	77.921	0.0155	0.0120	0.00099
GPR(n=1000)(MAP)(pym)	47.559	0.0432	0.0058	0.00030
GPR(n=4000)(MAP)(pym)	1189.5	0.1748	0.0028	0.00015
VFE(n=1000)(m=200)(MAP)(pym)	16.765	0.0082	0.01317	0.00124
VFE(n=4000)(m=200)(MAP)(pym)	30.487	0.0075	0.01502	0.00124

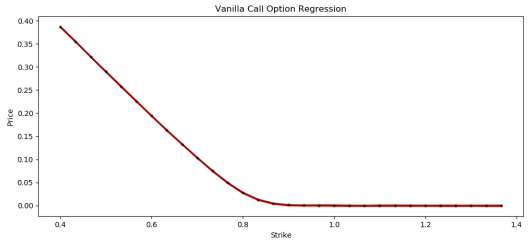
Table 9.4: Training and predicting American put options.

9.4 Visual Comparison Of Standard GPR, FITC and VFE

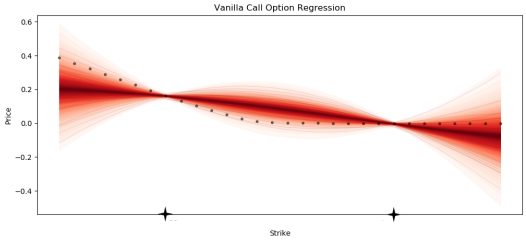
In this section we plot the stock value against the strike going full Bayesian. We use the Heston model and keep the other parameters constant. Every line in figure (9.6) equals a different sample of the posterior predictive distribution having different hyperparameters. Inducing points locations are denoted with a star. One clearly sees the hockey stick shape which is typical for a vanilla call option. Notice the superior expressiveness of the full GP, the improvement for sparse methods including more inducing points and overconfident variance for FITC at the inducing points.



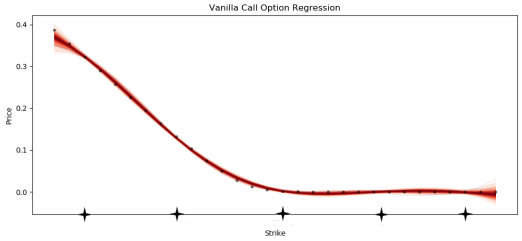
Exact GPR: $n = 10$.



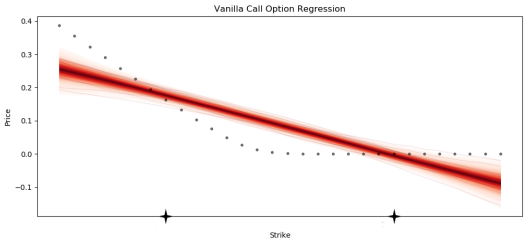
Exact GPR: $n = 30$.



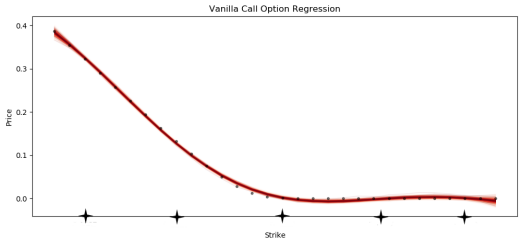
FITC: $n = 30$, $m = 2$.



FITC: $n = 30$, $m = 5$.



VFE: $n = 30$, $m = 2$.



VFE: $n = 30$, $m = 5$.

Figure 9.6: Plots price against strike vanilla calls going full Bayesian in het Heston model.

9.5 A Closing Word About Predictions

Consider the SVG and the VFE-BBMM regression models. Both are inducing methods and both are very fast. For the SVG we used stochastic techniques speeding up the calculation dramatically and VFE-BBMM uses BBMM framework speeding up de calculations which and enables parallelization. Hence, these models can handle huge amounts of data. A very striking result of inducing point methods is when keeping the number of inducing points constant, one can include more and more data-points without slowing down the prediction time. We choose to use the SVG-method and decide to really go to the extreme by fixing the number of inducing points and increasing the number of data-points. We use vanilla call options and predict 1000 prices. The accuracy can be found in figure (9.7) where one clearly sees a descending pattern. The prediction time fluctuates around a constant value which can be seen in figure (9.8). The expressiveness of the model and predictions can be further improved by including more inducing points although this implies longer training times and slower prediction times.

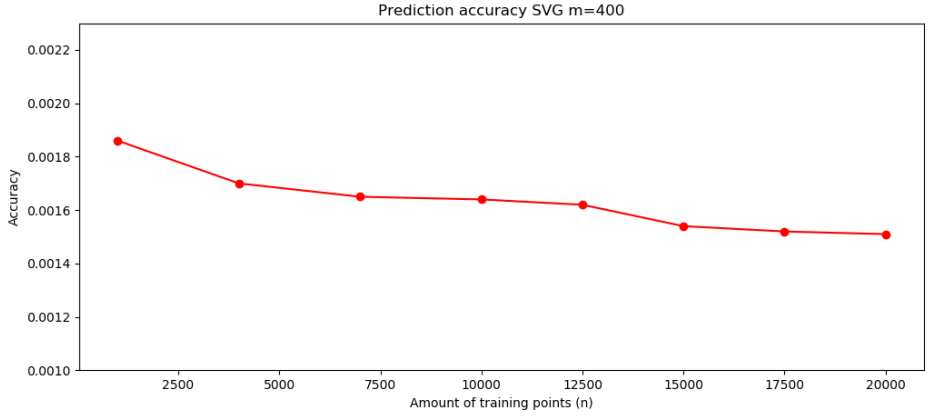


Figure 9.7: Prediction accuracy SVG $m = 400$.

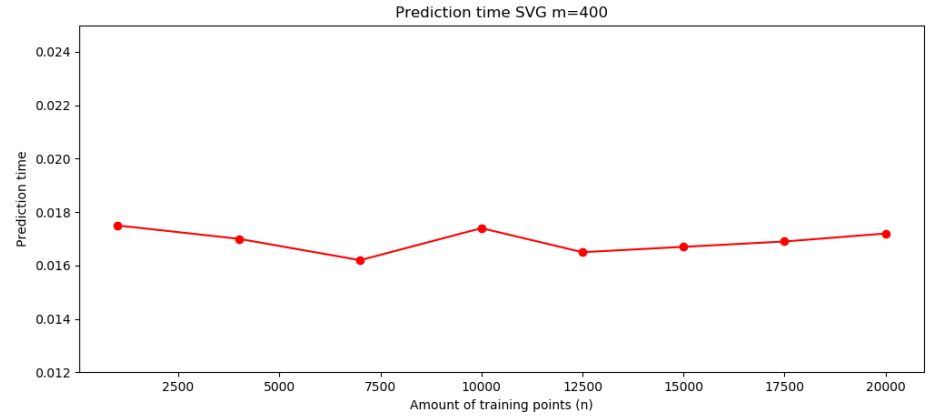


Figure 9.8: Prediction time SVG $m = 400$.

Chapter 10

Scheme And Conclusion

10.1 Scheme

A handy scheme describing all the models (and their abbreviations) we discussed can be found in figure (10.1). For BBMM GPR and the Bayesian methods, we only frame the methods we have discussed/referred to using these frameworks although other extensions may be possible like Bayesian polynomial regression. The methods we discussed but which we did not implement in Python are in dashed boxes.

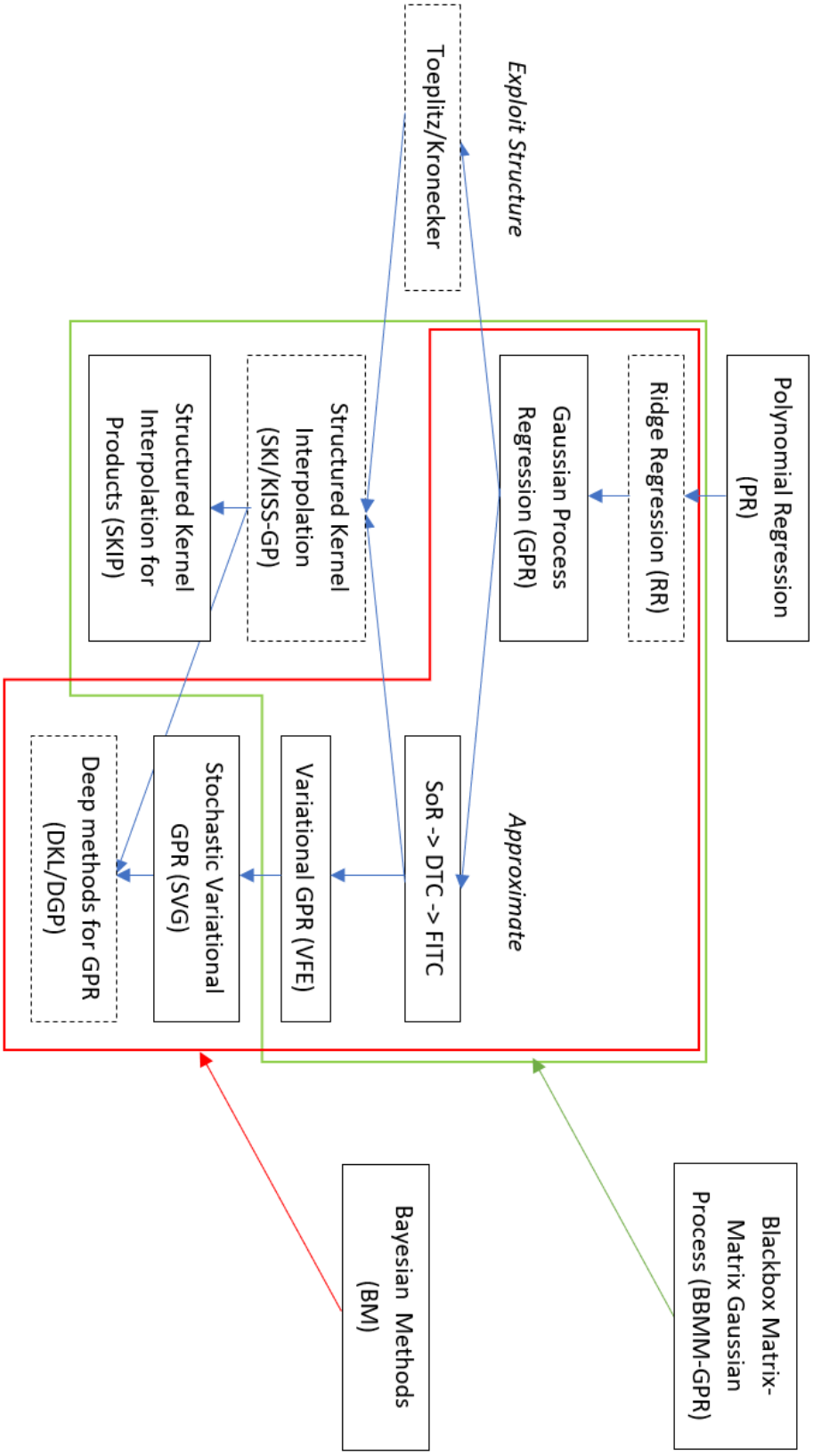


Figure 10.1: All the models.

10.2 Conclusion

The goal of this thesis is fast pricing of derivatives using machine learning techniques. We generated datasets using pricing techniques like FFT, Monte Carlo simulations and binomial trees applied on different financial models which we tried to learn using different GPR models.

The standard algorithm of Gaussian process regression achieves excellent prediction accuracy. However, for large datasets, this method becomes infeasible to train since it is $O(n^3)$ in computation time. Subsequently, predictions will be relatively slow since they are $O(n)$. Due to the fact that our problems are often high-dimensional ($d > 5$), standard methods exploiting the structure using Kronecker products and Toeplitz matrices are not suitable.

Inducing point methods approximate the posterior and by using them, we are able to achieve a considerable speed-up. For these methods, the time complexity for training is $O(nm^2 + m^3)$ and for predicting $O(m)$. Hence, using few inducing points, fast prediction can be obtained with some loss in accuracy. Since we need to train the locations of the inducing points, slow training and overfitting may occur. Hence, it is recommended to consider k-means for the inducing point locations and to use the VFE method which is protected from overfitting.

However, we are not yet satisfied due to still high training time complexity considering the number of inducing points. Using stochastic variational GPR, we can use mini batches for training which achieves a huge speed-up punishing less the number of inducing points. Furthermore, we have the same time complexity for prediction as standard inducing methods.

Next, we discussed methods which enables to train the models only using matrix vector multiplications (MVMs). Trying to exploit structure (SKI/KISS-GP), we again stumble on computational issues for high dimensional data. However, using the special properties for product kernels, we can avoid this problem (SKIP) and obtain a time complexity linear with the number of data which does not punish the number of inducing points in a severe way. Nevertheless, we encountered stability issues while training, since the method frequently uses the numerical unstable Lanczos decomposition.

Using approximations, the model loses some of its predictive ability. Thus we introduced the blackbox matrix matrix (BBMM) framework, enabling training standard GPR in $O(n^2)$ time and which also achieves considerable speed-ups for other methods. Since this framework only uses MVMs, it is possible to enable parallel computations.

Subsequently, we also discussed Bayesian and deep methods. However, they are more complex and one often needs to sample the predictive posterior which makes prediction slow.

Finally, we included a data study comparing the models trying to price derivatives and obtaining a considerable speed-up relative to the standard pricing techniques with little loss in accuracy.

Appendix

11.1 Brownian Motion

We follow (Schoutens, 2008). Let us denote with Ω the sample space, with $\mathcal{F}$ the event space which is a flow of information i.e. a filtration and with $\mathcal{P}$ the probability function.

Definition 14 (Brownian motion) *A stochastic process $X = \{X_t, t \geq 0\}$ is a standard Brownian motion on some probability space $(\Omega, \mathcal{F}, \mathcal{P})$, if*

1. $X_0 = 0$ *a.s.*
2. X *has independent increments*
3. X *has stationary increments*
4. $X_{t+v} - X_t \sim \mathcal{N}(0, v)$.

11.2 Pricing Derivatives Using Binomial Trees

In this section, we explain how to use binomial trees to price options. Suppose a stock has initial price S_0 and volatility σ. In our model, there are 2 possibilities after a small time Δt. The stock could have gone up having value $S_{\Delta t} = S_0 u = S_0(1 + \sigma\sqrt{\Delta t})$ or down $S_{\Delta t} = S_0 d = S_0(1 - \sigma\sqrt{\Delta t})$. Since Δt is small, we approximate this using $u = \exp(1 + \sigma\sqrt{\Delta t})$ and $d = \exp(1 - \sigma\sqrt{\Delta t})$. Denoting with r, q the interest and the dividend rate respectively and assuming a risk-neutral world, the probability to go up is equal to

$$p = \frac{\exp((r - q)\Delta t) - d}{u - d}. \tag{11.1}$$

Hence, the probability to go down is equal to $1 - p$. European options can be priced in this way taking the average price of all outcomes of the tree. For American options, it is necessary to check at each node to see whether early exercise is preferable to holding the option for a further time period Δt. By working back through all nodes, we can obtain the value of the option at time zero. By letting $\Delta t \to 0$, one obtains the Black-Scholes model.

11.3 Pricing Derivatives Using FFT Algorithm

We now consider how we can evaluate options using the Fast Fourier Transform (FFT). See (Carr and Madan, 1999) for further details and an improvement for out of the money (OTM) options. Denote with k the log-strike of the strike price K. We write with $q_{s_T}(s)$ the risk-neutral density of the log stock price s_t of S_T at maturity T and with $C(K,T)$ the desired value of a call option with strike $\exp(k)$. We define the characteristic function of this density by

$$\phi_T(u) = \int_{-\infty}^{\infty} e^{ius} q_{s_T}(s) ds. \tag{11.2}$$

For all the models we discussed (BS, Heston and VG), a characteristic can be found. Including the discounting with expectations taken under the risk neutral Q measure, we have that

$$\begin{aligned} C(k,T) &= \exp(-rT) E_Q[(S_T - k)^+] \\ &= \exp(-rT) \int_k^{\infty} (e^s - e^k) q_{s_T}(s) ds. \end{aligned} \tag{11.3}$$

However, if k tends to $-\infty$, the call function $C(k,T)$ converges to a non-zero constant (the zero strike call price). Hence, Fourier theory would not apply since this function is not square integrable. In order to obtain a square integrable function, we study the modified call price for some $\alpha > 0$ denoted by

$$c(k,T) = \exp(\alpha k) C(k,T). \tag{11.4}$$

Notice that we now need a condition on α for the positive log axis which we discuss later. Next, we take the Fourier transform of $c(k,T)$ and find that

$$\begin{aligned} \psi(v,T) &= \int_{-\infty}^{\infty} e^{ivk} c(k,T) dk \\ &= \int_{-\infty}^{\infty} e^{ivk} \int_k^{\infty} e^{\alpha k} e^{-rT} (e^s - e^k) q_{S_T}(s) ds dk \\ &= \int_{-\infty}^{\infty} e^{-rT} q_{S_T}(s) \int_{-\infty}^{s} (e^{s+\alpha k} - e^{(1+\alpha)k} e^{ivk} dk ds \\ &= \int_{-\infty}^{\infty} e^{-rT} q_{S_T}(s) \left[\frac{e^{(\alpha+1+iv)s}}{\alpha + iv} - \frac{e^{(\alpha+1+iv)s}}{\alpha + 1 + iv} \right] ds \\ &= \frac{e^{-rT} \phi_T(v - (\alpha+1)i)}{\alpha^2 + \alpha - v^2 + i(2\alpha+1)v}. \end{aligned} \tag{11.5}$$

Now, we use the inverse Fourier Transform and find that

$$C(k,T) = \frac{e^{-\alpha k}}{2\pi} \int_{-\infty}^{\infty} e^{-ivk} \psi(v,T) dv = \frac{e^{-\alpha k}}{\pi} \int_0^{\infty} e^{-ivk} \psi(v,T) dv. \tag{11.6}$$

Notice for the second equality that $C(K,T)$ is real. This means that $\psi(v,T)$ is even in its real part and odd in its imaginary part. To come back on our problem for (square) integrability for the modified call value in the positive log strike direction, it is satisfactory that $\psi(0,T)$ is finite. Thus, from equation (11.5) this holds if $\phi_T(v-(\alpha+1)i)$ is finite. Hence, an upper bound on α can be found using the characteristic function. Notice that also the last integral of equation (11.6) can be infinite were another condition concerning α is needed, for which we refer to the original paper.

Now we are ready to understand how to price options using the FFT. This is an efficient algorithm in order to compute the transform of $\boldsymbol{\alpha}_k$ to $\boldsymbol{\beta}_k$ with $k \in (1,\ldots,N)$ given by

$$\beta_k = \sum_{j=1}^{N} \exp\left(-\frac{2i\pi(j-1)(k-1)}{N}\right)\alpha_j, \tag{11.7}$$

where N is typically a power of 2. The algorithm reduces the time complexity from $O(N^2)$ to $O(N\log(N))$. We use the FFT to return N values of k and consider the grid with spacings of size λ given by

$$k_u = -\frac{N\lambda}{2} + \lambda(u-1) \qquad \text{for} \quad u \in (1,\ldots,N). \tag{11.8}$$

This gives us log strike levels ranging from $-\frac{N\lambda}{2}$ until $\frac{N\lambda}{2}$. We use the Simpson's rule (special case of Newton-Cotes formula's) for a good approximation of the integral of equation (11.6). We denote with $\eta = \frac{v_j}{j-1}$ and with δ_n the Kronecker delta which is zero except for $n=0$ for which it equals one and find that

$$C(k,T) = \frac{e^{-\alpha k}}{\pi}\sum_{j=1}^{N} e^{-iv_j k}\psi(v_j,T)\frac{\eta(3+(-1)^j-\delta_{j-1})}{3}. \tag{11.9}$$

If we fill equation (11.8) in equation (11.9) we obtain that

$$C(k_u,T) = \frac{e^{-\alpha k_u}}{\pi}\sum_{j=1}^{N} e^{-iv_j\left(-\frac{N\lambda}{2}+\lambda(u-1)\right)}\psi(v_j,T)\frac{\eta(3+(-1)^j-\delta_{j-1})}{3}. \tag{11.10}$$

In order to apply the FFT, we need that $\lambda\eta = \frac{2\pi}{N}$. We also use that $\eta = \frac{v_j}{j-1}$ and we write $b = \frac{N\lambda}{2}$. We find that

$$C(k_u,T) = \frac{e^{-\alpha k_u}}{\pi}\sum_{j=1}^{N} e^{-i\frac{2\pi}{N}(j-1)(u-1)}e^{ibv_j}\psi(v_j,T)\frac{\eta(3+(-1)^j-\delta_{j-1})}{3}, \tag{11.11}$$

which has exactly the same form as equation (11.7) as desired. We will use this equation for pricing vanilla options using the Heston and Variance gamma model. Of course, appropriate choices for N,η and α still needs to be chosen. We use $\eta = 0.25$, $N = 4096$ and $\alpha = 1.5$ like proposed in the original paper.

11.4 Pricing Derivatives Using Monte Carlo Simulations

In this section, we follow (Palczewski, 2016). We discuss the Euler and the Milstein scheme and some of its properties. Suppose we have the following stochastic differential equation (SDE)

$$dX_t = a(X_t)dt + b(X_t)W_t, \tag{11.12}$$

with a and b satisfying some smoothness conditions. We divide a time interval $[0, T]$ into N subintervals with $\delta t = T/N$ and $t_n = n\delta t$. The Euler scheme is given by

$$X_{t_{n+1}} = X_{t_n} + a(X_{t_n})\delta t + b(X_{t_n})(W_{t_{n+1}} - W_{t_n}). \tag{11.13}$$

Denote with $X_t^{\delta t}$ a process which connects the points obtained by the numerical scheme with straight lines and with X_t^{ex} the exact solutions. Denote with K_T a parameter that depends on T and the SDE. A numerical scheme is strongly convergent with order γ if

$$\mathbb{E}\left[|X_t^{\text{ex}} - X_t^{\delta_t}|\right] \leq K_T(\delta_t)^{\gamma}. \tag{11.14}$$

Euler's scheme may not suffice for pricing path-dependent options due to its poor strong convergence to the exact solution. It only converges strongly with order 0.5, i.e. if we make the step δt 100 times smaller, the approximation only improves with factor 10. The Milstein scheme is obtained as a result of stochastic Taylor expansion or by Itô's formula and is given by

$$\begin{aligned} X_{t_{n+1}} = X_{t_n} &+ a(X_{t_n})\delta t + b(X_{t_n})(W_{t_{n+1}} - W_{t_n}) \\ &+ \frac{1}{2}b'(X_{t_n})b(X_{t_n})\left((W_{t_{n+1}} - W_{t_n})^2 - \delta t\right), \end{aligned} \tag{11.15}$$

which converges strongly with order 1. We use this scheme for pricing DOBP options in the Heston model.

11.5 Bayes Theorem

Theorem 15 (Bayes theorem) *Denote with A and B two random variables. Using the definitions of their conditional probabilities, we obtain Bayes theorem given by*

$$p(A|B) = \frac{p(A)p(B|A)}{p(B)}. \tag{11.16}$$

11.6 The Matrix Inversion Lemma

In this section we follow (Rasmussen and Williams, 2006).

Lemma 5 (The Matrix Inversion Lemma) *Denote with Z a $n \times n$ matrix , W a $m \times m$ matrix and U and V are $n \times m$ matrices. Suppose that all necessary inverses exist. The matrix inversion lemma states that*

$$(Z + UWV^t)^{-1} = Z^{-1} - Z^{-1}U(W^{-1} + V^t Z^{-1}U)^{-1}V^t Z^{-1}. \tag{11.17}$$

Notice that if $m < n$, considerable speed-up can be achieved due to inversion of matrices with fewer dimensions.

11.7 Gaussian Identities

In this section, we follow (Rasmussen and Williams, 2006) and (Kuss, 2006).

Lemma 6 (Linear form of Gaussian) *Let $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \Sigma)$ and $\boldsymbol{y} = A\boldsymbol{x} + \boldsymbol{c}$. We find that $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{y}|A\boldsymbol{\mu} + \boldsymbol{c}, A\Sigma A^t)$.*

Lemma 7 (Conditional of Gaussians) *Suppose we have that*

$$\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix} \middle| \begin{bmatrix}\boldsymbol{\mu_x}\\ \boldsymbol{\mu_y}\end{bmatrix}, \begin{bmatrix}A & C\\ C^t & B\end{bmatrix}\right). \tag{11.18}$$

Then, supposing that all necessary inverses exist, we find that

$$\boldsymbol{y}|\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{y}|\boldsymbol{\mu_y} + C^t A^{-1}(\boldsymbol{x} - \boldsymbol{\mu_x}), B - C^t A^{-1} C). \tag{11.19}$$

Lemma 8 (Product of Gaussians) *Let $X \sim \mathcal{N}(\boldsymbol{x}|\boldsymbol{a}, A)$ and $Y \sim \mathcal{N}(\boldsymbol{x}|\boldsymbol{b}, B)$ with d the dimension of $\boldsymbol{a}$ and $\boldsymbol{b}$. The product of two Gaussians is another (un-normalized) Gaussian given by*

$$\begin{aligned} XY &\sim Z^{-1}\mathcal{N}(\boldsymbol{x}|\boldsymbol{c}, C) \\ &\quad \textit{with} \quad \boldsymbol{c} = C(A^{-1}\boldsymbol{a} + B^{-1}\boldsymbol{b}), \quad C = (A^{-1} + B^{-1})^{-1} \quad \textit{and} \\ &\quad Z^{-1} = (2\pi)^{-d/2}|A + B|^{-1/2} \exp\left(-\frac{1}{2}(\boldsymbol{a} - \boldsymbol{b})^t (A + B)^{-1}(\boldsymbol{a} - \boldsymbol{b})\right). \end{aligned} \tag{11.20}$$

Lemma 9 (Gaussian Integrals) *The following property holds*

$$\int_{\mathbb{R}} \mathcal{N}(\boldsymbol{x}|\boldsymbol{a}, A)\mathcal{N}(\boldsymbol{a}|\boldsymbol{b}, B)d\boldsymbol{a} = \mathcal{N}(\boldsymbol{x}|\boldsymbol{b}, A + B). \tag{11.21}$$

11.8 L-BFGS

In this section, we briefly discuss the L-BFGS methods and follow (Liu and Nocedal, 1989). The BFGS algorithm is a Quasi-Newton method where one approximate the Hessian in each iteration using knowledge of the gradient and the previous approximation of the Hessian by a secant condition. L-BFGS is preferable for large datasets since it only uses a limited memory.

11.9 Cholesky Decomposition

In the section we follow (Krishnamoorthy and Menon, 2013).

Definition 15 (Cholesky Decompostion) *If a matrix $A \in \mathbb{C}$ is a positive definite Hermitian matrix, the Cholesky decompostion factorises the matrix into a product of a lower triangular matrix L and its conjugate transpose L^* i.e.*

$$A = LL^* \tag{11.22}$$

This factorisation is of order $O(n^3)$. We only use real matrices and we work from now with $L^* = L^t$. Inversion based on Cholesky decomposition is numerically stable for well conditioned matrices. The inverse can be found using

$$(LL^t)^{-1} = (L^t)^{-1}L^{-1}. \tag{11.23}$$

The inverse of triangular matrices can be calculated in a recursive fashion. Remember that the determinant of the product equals the product of the determinants and that the determinant of a triangular matrix equals the product of its trace. Hence, the determinant of A equals the determinant of L multiplied with the determinant of L^t which equals the product of squares of the diagonal elements of L.

11.10 The Kronecker Product

We state the most important properties of the Kronecker product found in (Saatçi, 2012) where one can find the corresponding proofs.

Definition 16 (Kronecker product) *Let A be a $m \times n$ matrix and B a $p \times q$ matrix. The Kronecker product of these two matrices is given by*

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}, \tag{11.24}$$

which is a $mp \times nq$ matrix. Let us denote with A_c an $e_c \times e_c$ matrix and with $N = \prod_{c=0}^{d} e_c$. Using Kronecker products, we say that matrix indices start at zero for simplicity. We impose that

1. $0 \leq i^{(c)}, j^{(c)} < e_c$ *(matrix indices start at zero),*

2. $0 \leq i < N$ *and* $0 \leq j < N$
3. $i = \sum_{c=1}^{d} (e_c)^c i^{(c)}$ *and* $j = \sum_{c=1}^{d} (e_c)^c j^{(c)}$.

Hence, we find for the more general equation $A = A_1 \otimes \cdots \otimes A_d = \bigotimes_{c=1}^{d} A_c$ *that*

$$A(i,j) = A_1(i^{(1)}, j^{(1)}) A_2(i^{(2)}, j^{(2)}) \cdots A_d(i^{(d)}, j^{(d)}). \tag{11.25}$$

Next, we list some important properties of the Kronecker product.

Properties 1 (Kronecker product) *Denote with* A *and* B *two square matrices with dimension* N_a *and* N_b *respectively. We have that*

- $\det(A \otimes B) = \det(A)^{N_a} \det(B)^{N_b}$

Subsequently, suppose that $K = \bigotimes_{c=1}^{d} K_c$*, we have that*

- $K^{-1} = \bigotimes_{c=1}^{d} K_c^{-1}$

Furthermore, if $K = \bigotimes_{c=1}^{d} K_c$ *and* $K = LL^t$ *and* $K_c = L_c L_c^t$*, we find that*

- $L = \bigotimes_{c=1}^{d} L_c$

A very similar result can be obtained for the eigendecompositions. If $K = \bigotimes_{c=1}^{d} K_c$ *and let* $K_c = Q_c \Lambda_c Q_c^t$ *the eigendecomposition of* K_c*. Then the eigendecomposition for* $K = Q \Lambda Q^t$ *has the following properties*

- $Q = \bigotimes_{c=1}^{d} Q_c$
- $\Lambda = \bigotimes_{c=1}^{d} \Lambda_c$

11.11 K-means

In this section, we follow (Gormley, 2017). We discuss the K-means algorithm in its most basic form (LLoyd's method) and give some possible extensions. First, choose the number of data clusters we want to obtain. In our case, this is the number of inducing points m. From now, we denote with $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m$ the cluster centres and with $C_1, \ldots, C_m$ the clusters. However, we have to initialise the center of our clusters. These points can be chosen at random which is not recommended since one probably initialises multiple points in one cluster. Hence, using a heuristic is recommended. A popular heuristic is the furthest point heuristic where one picks centres among the data points furthest away from the previously chosen centres. Notice that this method performs poorly if there are a lot of outliers. A very popular algorithm for choosing the initial values is the k-means++ initialisation (see algorithm 3), which spreads out the initial cluster centers.

The time complexity of the k-means++ initialisation is $O(nmd)$. Now LLoyd's algorithm with p iterations is given in algorithm 4.

Algorithm 3 k-means++ initialisation

input: X
output: $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m$

Pick $\boldsymbol{c}_1$ arbitrary
For $j = 2, \ldots, m$ do:
    Pick $\boldsymbol{c}_j$ among $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ according to distribution $P(\boldsymbol{c}_j = \boldsymbol{x}_i) \propto \min_{j'<j} ||\boldsymbol{x}_i - \boldsymbol{c}_{j'}||^2$.

Algorithm 4 LLoyd's algorithm

input: X and $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m$ (initial cluster centers)
output: $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m$ (updated cluster centers) and $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_m$ (the clusters)

For $i = 1, \ldots, p$ do:
    For $i = 1, \ldots, n$ do:
        $\boldsymbol{x}_i \in C_j$ if $\boldsymbol{x}_i$ closed to $\boldsymbol{c}_j$ with $j \in (1, \ldots, m)$
    For $j = 1, \ldots, m$ do:
        Set $\boldsymbol{c}_j$ mean of values in C_j

One can choose a metric to calculate the distances between the points. By running a fixed number of p iterations (or using an upper bound of p iterations), the time complexity of this algorithm is given by $O(nmdp)$. Hence the accumulation of LLoyd's algorithm with k-means++ initialisation is $O(nmdp)$. For large problems, one can use mini-batches to update centre positions.

11.12 Sylvester's Determinant Identity

We state the theorem found in (Ambikasaran et al., 2014).

Lemma 10 (Sylvester's Determinant Identity) *If $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times m}$, we find that*

$$\det(I_m + AB) = \det(I_n + BA). \tag{11.26}$$

11.13 The Kullback-Leibler Divergence And Variational Lower Bound

In this section, we follow (Blei et al., 2017) and (Yang, 2017). We denote with $\boldsymbol{y}$ the observations and $\boldsymbol{z}$ the latent variables.

Definition 17 (Kullback-Leibler (KL) divergence and variational lower bound) *The Kullback-Leibler (KL) divergence is a measure of the closeness of two distributions,*

which we denote with $q(\boldsymbol{z})$ *and* $p(\boldsymbol{z}|\boldsymbol{y})$*, given by*

$$\begin{aligned} KL(q(\boldsymbol{z})||p(\boldsymbol{z}|\boldsymbol{y})) &:= \mathbb{E}_q\left[\log \frac{q(\boldsymbol{z})}{p(\boldsymbol{z}|\boldsymbol{y})}\right] \\ &= \mathbb{E}_q[\log q(\boldsymbol{z})] - \mathbb{E}_q[\log p(\boldsymbol{z})|\boldsymbol{y}] \\ &= \mathbb{E}_q[\log q(\boldsymbol{z})] - \mathbb{E}_q[\log p(\boldsymbol{z},\boldsymbol{y})] + \log p(\boldsymbol{y}) \\ &= -\mathcal{L}(q) + \log p(\boldsymbol{y}). \end{aligned} \tag{11.27}$$

The variational lower bound (often called evidence lower bound (ELBO) is by definition equal to $\mathcal{L}(q)$*.*

Now, one can find the following important theorem.

Theorem 16 $\mathcal{L}(q)$ *is a lower bound for* $\log p(\boldsymbol{y})$*.*

Proof: It is sufficient to prove that the KL divergence of two distributions is always positive, from which the desired property holds. The proof is nothing more than Jensen's inequality for concave functions. Jensen's inequality states that for a concave function h and a r.v. X

$$\mathbb{E}[h(X)] \overset{\text{J.}}{\leq} h(\mathbb{E}[X]). \tag{11.28}$$

For convex functions, the opposite holds. Noting that the logarithm is a concave function and the expectation a linear operator, we find that

$$\begin{aligned} -\mathrm{KL}(q(\boldsymbol{z})||p(\boldsymbol{z}|\boldsymbol{y})) &= \mathbb{E}_q\left[-\log\left(\frac{q(\boldsymbol{z})}{p(\boldsymbol{z}|\boldsymbol{y})}\right)\right] \\ &= \mathbb{E}_q\left[\log\left(\frac{p(\boldsymbol{z}|\boldsymbol{y})}{q(\boldsymbol{z})}\right)\right] \\ &\overset{\text{J.}}{\leq} \log\left(\mathbb{E}_q\left[\frac{p(\boldsymbol{z}|\boldsymbol{y})}{q(\boldsymbol{z})}\right]\right) \\ &= \log(1) = 0 \end{aligned} \tag{11.29}$$

□

We state an exact expression for the KL-divergence between two multivariate Gaussians found in (Duchi, 2007)

Lemma 11 *Suppose that* $q \sim \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_1, \Sigma_1)$ *and* $p \sim \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_2, \Sigma_2)$*. We find that*

$$KL(q(\boldsymbol{z})||p(\boldsymbol{z})) = \frac{1}{2}\left(\log\left(\frac{|\Sigma_2|}{|\Sigma_1|}\right) - d + \mathrm{Tr}(\Sigma_2^{-1}\Sigma_1) + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^t\Sigma_2^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)\right). \tag{11.30}$$

11.14 Stochastic Variational Inference and Stochastic Gradient Methods

In this section, we follow (Bottou, 2010) and (Blei et al., 2017). We use the same notation as in the previous section. We need that the variational objective (f.e. the ELBO) decomposes into a sum of terms, one for each data point in the analysis, so we can use stochastic optimisation. In order to do so, we strive for a structure like in figure (11.2). The distribution of each observation y_i only corresponds on its latent/local variable z_i and the global variables $\boldsymbol{g}$.

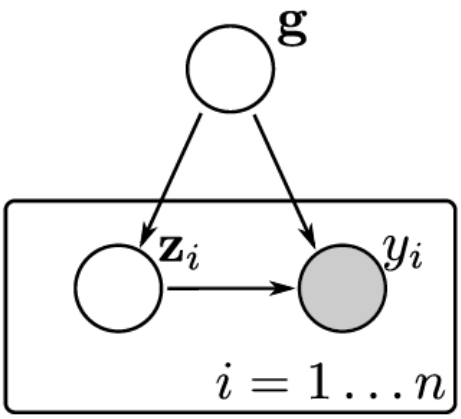


Figure 11.2: Figure from (Hensman et al., 2013). Requirements for SVI.

Let us denote with $\boldsymbol{\theta}$ the family of parameters we want to optimize by minimising the loss $\mathcal{L}(\boldsymbol{z}, \boldsymbol{y}, \boldsymbol{\theta})$. Suppose that this loss can be decomposed in a sum such that

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{y}, \boldsymbol{\theta}) = \sum_{i=1}^{n} l(z_i, y_i, \boldsymbol{\theta}). \tag{11.31}$$

Using the ordinary gradient descent, we can update the parameters by taking steps (with a gain/learning rate γ) in the direction of the gradient. Notice also the strong similarities with the famous Newton-based optimisation algorithms and remember that the gradient is the direction of the steepest ascent. We find that

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \bigtriangledown l(z_i, y_i, \boldsymbol{\theta}). \tag{11.32}$$

One can achieve linear convergence when, under sufficient regularity conditions, the initial value $\boldsymbol{\theta}_0$ is close enough to the optimum and the gain is sufficiently small. A second order gradient descent can be found by by replacing the scalar gain γ by a positive definite matrix that approaches the inverse of the Hessian of the cost at the optimum. Now, stochastic gradient descent calculates the gradient of one arbitrary point (of a batch B containing arbitrary points with a specified size) and one does

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \frac{1}{n} \sum_{b \in B} \bigtriangledown l(z_b, y_b, \boldsymbol{\theta}). \tag{11.33}$$

This is of course less computationally demanding and results often only in a slightly lower convergence rate. Convergence results usually require decreasing gains satisfying the conditions $\sum_t \gamma_t^2 < \infty$ and $\sum_t \gamma_t = \infty$ and some mild conditions. The best convergence speed is achieved using gains $\gamma_t \sim t^{-1}$. This method works when equation (11.33) behaves like its expectation equation (11.32) despite the noise introduced by this simplified procedure. However, a bad choice of the learning rate can be problematic. Hence, a more advanced optimise, for example the Adam optimiser (see section 11.15), is recommended.

11.15 Adam Optimiser

In this section, we follow (Kingma and Ba, 2014). Adam only requires first-order gradients for efficient stochastic optimization with little memory requirement. Using estimates for the first and second moment of the gradient, the method computes individual adaptive learning rates. The algorithm and more information can be found in the source.

11.16 The Natural Gradient

In this section, we give a short outline about gradients and follow (Hoffman et al., 2013). For the classical gradient method, one can optimise a function $f(\boldsymbol{\lambda})$ by taking steps (with a step-size) in the direction of the gradient. Remember that the gradient is the direction of the steepest ascent. Thus the direction of the gradient $\bigtriangledown_{\boldsymbol{\lambda}} f(\boldsymbol{\lambda})$ points in the same direction as the solution of the problem

$$\arg\max_{d\boldsymbol{\lambda}}(f(\boldsymbol{\lambda}) + \boldsymbol{d\lambda}) \qquad \text{subject to} \quad ||d\boldsymbol{\lambda}||^2 < \varepsilon, \tag{11.34}$$

for sufficient small ε. Notice that the direction of the gradient depends on the Euclidian distance metric which is often a poor measure of the dissimilarity of distributions $p(g|\lambda)$ and $p(g|\lambda')$. For example, suppose that $p(g)$ is the univariate normal with the parameter $\boldsymbol{\lambda}$ equal to the mean μ and standard deviation σ. The distributions $\mathcal{N}(0, 10000)$ and $\mathcal{N}(10, 10000)$ are almost indistinguishable and the Euclidean distance equals 10. However, the distributions $\mathcal{N}(0, 0.01)$ and $\mathcal{N}(0.1, 0.01)$ barely overlap and their Euclidean distance equals 0.01.

Hence, we need the introduce the concept of natural gradient. This is based on the symmetrized KL divergence given by

$$D_{KL}^{sym}(\boldsymbol{\lambda}, \boldsymbol{\lambda'}) = \mathbb{E}_{\boldsymbol{\lambda}} \left[\log \frac{p(g|\boldsymbol{\lambda})}{p(g|\boldsymbol{\lambda'})} \right] + \mathbb{E}_{\boldsymbol{\lambda'}} \left[\log \frac{p(g|\boldsymbol{\lambda'})}{p(g|\boldsymbol{\lambda})} \right] \tag{11.35}$$

The natural gradient $\hat{\bigtriangledown}_{\boldsymbol{\lambda}} f(\boldsymbol{\lambda})$points in the same direction as the solution of the problem

$$\arg\max_{d\boldsymbol{\lambda}} f(\boldsymbol{\lambda}) + \boldsymbol{d\lambda}) \qquad \text{subject to} \quad D_{KL}^{sym}(\boldsymbol{\lambda}, \boldsymbol{\lambda} + d\boldsymbol{\lambda}) < \varepsilon, \tag{11.36}$$

with ε small enough. Denoting with $I(\boldsymbol{\lambda})$ the fisher information matrix of $p(\boldsymbol{\lambda})$, one has shown that the natural gradient equals

$$\hat{\bigtriangledown}_{\boldsymbol{\lambda}} f(\boldsymbol{\lambda}) = I(\boldsymbol{\lambda})^{-1} \bigtriangledown_{\boldsymbol{\lambda}} f(\boldsymbol{\lambda}). \tag{11.37}$$

We will not go in detail about exponential families. However, the natural gradient is easily computable in this case (even easier as the normal gradient).

11.17 A Short Notion About Interpolation

In the first part of this section, we follow (Keys, 1981).

In order to have clear notation, we only describe the 2-dimensional case. Denote with h_x and h_y the sample increments, (x_k, y_l) the interpolation nodes, u the interpolation kernel and f the sampled function. For equally spaced (in each dimension) data, any interpolation function (2D) can be written in the form

$$g(x, y) = \sum_k \sum_l c_{k,l} u\left(\frac{x - x_k}{h_x}\right) u\left(\frac{y - y_l}{h_y}\right), \tag{11.38}$$

since we assume that $u(0) = 1$. Since we interpolate f, we want $g(x_k, y_l) = f(x_k, y_l)$. Hence, we replace all the $c_{k,l}$ by the sampled data i.e. $c_{k,l} = f(x_k, y_l)$. The linear interpolation kernel is given by

$$u(s) = \begin{cases} -s + 1, & \text{if } 0 < s < 1 \\ s + 1, & \text{if } -1 < s \leq 0 \\ 0 & \text{if } |s| \geq 1. \end{cases} \tag{11.39}$$

The cubic convolution interpolation kernel is given by

$$u(s) = \begin{cases} \frac{3}{2}|s|^3 - \frac{5}{2}|s|^2 + 1 & \text{if } 0 \leq |s| < 1 \\ -\frac{1}{2}|s|^3 + \frac{5}{2}|s|^2 - 4|s| + 2 & \text{if } 1 \leq |s| < 2 \\ 0 & \text{if } |s| \geq 2. \end{cases} \cdot \tag{11.40}$$

In this case, the interpolation error goes to zero uniformly at a rate proportional to $O(h^3)$. However, equispaced grids are not always applicable. In this case, one can use local inverse distance weighting interpolation, which can be applied to irregular grids. From now, we follow (Babak and Deutsch, 2009). It is defined as a spatially weighted average of the sample values within a search neighbourhood U_x. Suppose d is the Euclidian metric, although other possibilities are possible.

Let us denote

$$\lambda_i(\boldsymbol{x}) = \frac{\frac{1}{d(\boldsymbol{x}_i,\boldsymbol{x})}}{\sum\limits_{j:\boldsymbol{x}_j\in U_{\boldsymbol{x}}} \frac{1}{d(\boldsymbol{x}_j,\boldsymbol{x})}}. \tag{11.41}$$

Then local inverse distance weighting interpolation is calculated as

$$g(\boldsymbol{x}) = \sum_{i:\boldsymbol{x}_i\in U_{\boldsymbol{x}}} \lambda_i(\boldsymbol{x}) f(\boldsymbol{x}_i). \tag{11.42}$$

11.18 The Lanczos Decomposition

In this section, we follow (Gardner et al., 2018a) and chapter 10 of (Arbenz et al., 2012) which is clearly written. The Lanczos decomposition executes an orthogonal transformation from a Hermitian matrix to a tridiagonal form. It is a simplification of the Arnoldi algorithm using the Hermitian property. Let A be a $n \times n$ an Hermitian matrix and $\boldsymbol{x}$ a generating vector. The Lanczos decomposition computes an orthonormal basis of the Krylov space $\mathcal{K}^j(\boldsymbol{x}, A) = \text{span}\{\boldsymbol{x}, A\boldsymbol{x}, \ldots, A^{j-1}\boldsymbol{x}\}$. As one could suspect, one uses the Gram–Schmidt orthogonalization process and exploits the Hermitian form. Now, let Q_k be a $n \times k$ matrix representing the orthonormal basis for $\mathcal{K}^k(\boldsymbol{x}, A)$ and T_k a tridiagonal $k \times k$ matrix. The Lanczos relation is given by

$$AQ_k = Q_k \underbrace{\begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{k-1} \\ & & & \beta_{k-1} & \alpha_k \end{bmatrix}}_{T_k} + \beta_k [\underbrace{\mathbf{0}, \ldots, \mathbf{0}}_{k-1}, \boldsymbol{q}_{k+1}]^t. \tag{11.43}$$

Remember that the inverse of an orthogonal matrix equals the transpose. Hence, this procedure yields a low-rank approximate tridiagonalisation

$$A \approx Q_k T_k Q_k^t. \tag{11.44}$$

Now let m be the smallest index such that $\mathcal{K}^m\boldsymbol{x} = \mathcal{K}^{m+1}\boldsymbol{x}$. Notice also that the set $\{1, A, A^2, \ldots, A^n\}$ is linearly dependent for any matrix A. Hence, for n iterations this procedure produces always an exact iteration. Then $\beta_k = 0$ and we obtain

$$AQ_m = Q_m T_m \tag{11.45}$$

with $Q_m = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m]$ ($\boldsymbol{q}_1$ is $\boldsymbol{x}$) and we find $A = Q_m T_m Q_m^t$. The algorithm can be found in (Arbenz et al., 2012). The main results are that in a single iteration step, we only have to execute a matrix-vector multiplication, $7n$ further floating point operations and that

the eigenvalues of T_m equals the eigenvalues of A. However, the Lanczos tridiagonalization algorithm has storage and numerical stability issues.

11.19 Conjugate Gradient Method

In the previous sections of the appendix about numerical techniques (Cholesky decomposition, L-BFGS, Adam, Lanczos decomposition, ...) we omitted the pseudo-code and its derivations which can be found in the corresponding sources. However, this algorithm will be modified and used to retain the Lanczos tridiagonal matrix T, thus a more expanded discussion is appropriate. Nevertheless, one can also directly go to algorithm 5. First, we follow (Shewchuk et al., 1994) where the derivations can be found and which is a piece of art. The conjugate gradient (CG) tries to find the solution of a numerical system of linear equations and is faster than the steepest decent (due to zigzagging etc.). The quadratic form

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^t A\boldsymbol{x} - \boldsymbol{b}^t\boldsymbol{x} + c \tag{11.46}$$

is minimised by the solution of $A\boldsymbol{x} = b$ if A is symmetric. Hence, we can try to find a method minimising equation (11.46). We denote with $\boldsymbol{e}_{(i)} = \boldsymbol{x}_{(i)} - \boldsymbol{x}$ the error, with $\boldsymbol{r}_{(i)} = \boldsymbol{b} - A\boldsymbol{x}_i$ the residual and with $\boldsymbol{d}_{(i)}$ the search directions. It is easy to see that $\boldsymbol{r}_{(i)} = -A\boldsymbol{e}_{(i)}$. Notice that $\boldsymbol{r}_{(i)} = -f(\boldsymbol{x}_{(i)})$, which implies that the residual is the direction of steepest descent. For each step we choose a point

$$\boldsymbol{x}_{(i+1)} = \boldsymbol{x}_{(i)} + \alpha_{(i)}\boldsymbol{d}_{(i)}. \tag{11.47}$$

The main question is, how big must the step α be? Two vectors $\boldsymbol{d}_{(i)}$ and $\boldsymbol{d}_{(j)}$ are A-orthogonal or conjugate if $\boldsymbol{d}^t_{(i)}A\boldsymbol{d}_{(j)} = 0$. We require that $\boldsymbol{e}_{(i+1)}$ is A-orthogonal to $\boldsymbol{d}_{(i)}$. This is equivalent to finding the minimum point along the search direction since

$$\frac{d}{d\alpha}f(\boldsymbol{x}_{(i+1)}) = 0 \Leftrightarrow f'(\boldsymbol{x}_{(i+1)})^t\frac{d}{d\alpha}\boldsymbol{x}_{(i+1)} = 0 \Leftrightarrow -(\boldsymbol{r}_{(i+1)})^t\boldsymbol{d}_{(i)} = 0 \Leftrightarrow \boldsymbol{d}^t_{(i)}A\boldsymbol{e}_{(i+1)} = 0. \tag{11.48}$$

This implies that

$$\boldsymbol{d}^t_{(i)}A\boldsymbol{e}_{(i+1)} = 0 \Leftrightarrow \boldsymbol{d}^t_{(i)}A(\boldsymbol{e}_{(i)} + \alpha_{(i)}\boldsymbol{d}_{(i)}) = 0 \Leftrightarrow \alpha_{(i)} = \frac{\boldsymbol{d}_{(i)}\boldsymbol{r}_{(i)}}{\boldsymbol{d}^t_{(i)}A\boldsymbol{d}_{(i)}}. \tag{11.49}$$

Now we try to find A-orthogonal search directions. We generate them using the conjugate Gram-Schmidt process. Suppose we have a set $\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{n-1}$, which are linearly independent vectors. Set $\boldsymbol{d}_0 = \boldsymbol{u}_0$ and for $i > 0$ set

$$\boldsymbol{d}_{(i)} = \boldsymbol{u}_i + \sum_{k=0}^{i-1}\beta_{ik}\boldsymbol{d}_{(k)} \tag{11.50}$$

with

$$\beta_{(i,j)} = -\frac{\boldsymbol{u}_i^t A \boldsymbol{d}_{(j)}}{\boldsymbol{d}_{(j)}^t A \boldsymbol{d}_{(j)}}. \tag{11.51}$$

Hence, we subtract out any components that are not A-orthogonal to the previous $\boldsymbol{d}$ vectors. By taking the product of equation (11.50) with $\boldsymbol{r}_{(i)}$, one finds that

$$\boldsymbol{d}_{(i)}^t \boldsymbol{r}_{(i)} = \boldsymbol{u}_{(i)}^t \boldsymbol{r}_{(i)}. \tag{11.52}$$

Now, the error term is A-orthogonal to all old search directions. Because $\boldsymbol{r}_{(i)} = -A\boldsymbol{e}_{(i)}$ each residual is orthogonal to the previous search directions. This also implies that each residual is orthogonal to the previous residuals. Now seeing that

$$\boldsymbol{r}_{(i+1)} = -A\boldsymbol{e}_{(i+1)} = -A(\boldsymbol{e}_{(i)} + \alpha_{(i)}\boldsymbol{d}_{(i)}) = \boldsymbol{r}_{(i)} - \alpha_{(i)} A \boldsymbol{d}_{(i)}, \tag{11.53}$$

we find that each residual is linear combination of $A\boldsymbol{d}_{(i)}$ and the previous residual. Hence, the linear space of residuals $\text{span}\{\boldsymbol{r}_{(0)}, \ldots, \boldsymbol{r}_{(i)}\}$ equals the Krylov space $\mathcal{K}(\boldsymbol{r}_{(0)}, A)$. Now, the Gram-Schmidt constants are given by

$$\beta_{(i,j)} = -\frac{\boldsymbol{r}_{(j)} A \boldsymbol{d}_{(j)}}{\boldsymbol{d}_{(j)} A \boldsymbol{d}_{(j)}}. \tag{11.54}$$

Taking the inner product from equation (11.53), we find that

$$\begin{aligned} \boldsymbol{r}_{(i)}^t \boldsymbol{r}_{(j+1)} &= \boldsymbol{r}_{(i)}^t \boldsymbol{r}_{(j)} - \alpha_{(j)} \boldsymbol{r}_{(i)}^t A \boldsymbol{d}_{(j)} \\ \alpha_{(j)} \boldsymbol{r}_{(i)}^t A \boldsymbol{d}_{(j)} &= \boldsymbol{r}_{(i)}^t \boldsymbol{r}_{(j)} - \boldsymbol{r}_{(i)}^t \boldsymbol{r}_{(j+1)}. \end{aligned} \tag{11.55}$$

Now, since each residual is orthogonal to the previous residuals, we find that

$$\boldsymbol{r}_{(i)}^t A \boldsymbol{d}_{(j)} = \begin{cases} \frac{1}{\alpha_{(i)}} \boldsymbol{r}_{(i)}^t \boldsymbol{r}_{(i)} & \text{if } i = j \\ \frac{1}{\alpha_{(i-1)}} \boldsymbol{r}_{(i)}^t \boldsymbol{r}_{(i)} & \text{if } i = j+1 \\ 0 & \text{otherwise.} \end{cases} \tag{11.56}$$

Hence, we find that

$$\beta_{(i,j)} = \begin{cases} \frac{1}{\alpha_{(i-1)}} \dfrac{\boldsymbol{r}_{(i)}^t \boldsymbol{r}_{(i)}}{\boldsymbol{d}_{(i-1)}^t A \boldsymbol{d}_{(i-1)}} & \text{if } i = j+1 \\ 0 & \text{if } i > j+1, \end{cases} \tag{11.57}$$

which means that most of the $\beta_{(i,j)}$ terms disappear making the calculations much faster. Now using the abbreviation $\beta_{(i)} = \beta_{(i,i+1)}$ and equation (11.49), (11.52), we find that

$$\begin{aligned}\beta_{(i)} &= \frac{\boldsymbol{r}_{(i)}^t \boldsymbol{r}_{(i)}}{\boldsymbol{d}_{(i-1)}^t \boldsymbol{r}_{i-1}} \\ &= \frac{\boldsymbol{r}_{(i)}^t \boldsymbol{r}_{(i)}}{\boldsymbol{r}_{(i-1)}^t \boldsymbol{r}_{(i-1)}}.\end{aligned} \tag{11.58}$$

Putting everything together, we find algorithm 5.

Algorithm 5 Conjugate Gradient Method

input: $A, \boldsymbol{b}$
output: $A^{-1}\boldsymbol{b}$
$\boldsymbol{x}_{(0)} = 0$
Compute $\boldsymbol{d}_{(0)} = \boldsymbol{r}_{(0)} = \boldsymbol{b} - A\boldsymbol{x}_{(0)}$
For $i = 0, \ldots, p$ do :
    $\alpha_{(i)} = \dfrac{\boldsymbol{d}_{(i)}^t \boldsymbol{r}_{(i)}}{\boldsymbol{d}_{(i)}^t A \boldsymbol{d}_{(i)}}$
    $\boldsymbol{x}_{(i+1)} = \boldsymbol{x}_{(i)} + \alpha_{(i)} \boldsymbol{d}_{(i)}$
    $\boldsymbol{r}_{(i+1)} = \boldsymbol{r}_{(i)} - \alpha_{(i)} A \boldsymbol{d}_{(i)}$
    If $||\boldsymbol{r}_{(i+1)}||_2 < tolerance$, then return $\boldsymbol{x}_{(i+1)}$
    $\beta_{(i)} = \dfrac{\boldsymbol{r}_{(i+1)}^t \boldsymbol{r}_{(i+1)}}{\boldsymbol{r}_{(i)}^t \boldsymbol{r}_{(i)}}$
    $\boldsymbol{d}_{(i+1)} = \boldsymbol{r}_{(i+1)} + \boldsymbol{\beta}_{(i+1)} \boldsymbol{d}_{(i)}$
Return $\boldsymbol{x}_{(i+1)}$

Notice that the time complexity of this algorithm is $O(p\xi(A))$ with p the (maximal) number of iterations and $\xi(A)$ the cost of one MVM with A. The error can be bounded in terms of an exponential decay using the condition number. One already could have realized that the conjugate gradient method is an orthogonal projection technique onto the Krylov subspace $\mathcal{K}(\boldsymbol{r}_{(0)}, A)$. Hence, it is maybe not so suprising that the conjugate gradient algorithm can be derived from the Lanczos algorithm. We omitted an excessive discussion of the Lanczos decomposition and its pseudocode since it is possible to retain the Lanczos tridiagonal matrix T by running the conjugate gradient method. We do not go in detail and just state the result.

Lemma 12 (Retrieving Lanczos tridiagonal matrices from standard CG) *Let $\alpha_{(0)}, \ldots, \alpha_{(p-1)}$ and $\beta_{(0)}, \ldots, \beta_{(p-1)}$ the scalar coefficients of algorithm 5. The p-dimensional Lanczos tridiagonal matrix in terms of conjugate gradient coefficients is given by*

$$T_p = \begin{bmatrix} \frac{1}{\alpha_{(0)}} & \frac{\sqrt{\beta_{(0)}}}{\alpha_{(0)}} & & & & \\ \frac{\sqrt{\beta_{(0)}}}{\alpha_{(0)}} & \frac{1}{\alpha_{(1)}} + \frac{\beta_{(0)}}{\alpha_{(0)}} & \frac{\sqrt{\beta_{(1)}}}{\alpha_{(1)}} & & & \\ & \frac{\sqrt{\beta_{(1)}}}{\alpha_{(1)}} & \frac{1}{\alpha_{(2)}} + \frac{\beta_{(1)}}{\alpha_{(1)}} & \ddots & & \\ & & \ddots & \ddots & \frac{\sqrt{\beta_{(p-2)}}}{\alpha_{(1)}} & \\ & & & \frac{\sqrt{\beta_{(p-2)}}}{\alpha_{(p-2)}} & \frac{1}{\alpha_{(p-1)}} + \frac{\sqrt{\beta_{(p-2)}}}{\alpha_{(p-2)}} & \end{bmatrix} \tag{11.59}$$

11.20 Preconditioning

In this section we follow (Gardner et al., 2018a) and (Shewchuk et al., 1994). Let us introduce a matrix P and try to solve the linear system

$$P^{-1}A\boldsymbol{v} = P^{-1}\boldsymbol{y}. \tag{11.60}$$

Now, we notice that this system and $A^{-1}\boldsymbol{y}$ has the same solution. As has been said before, the convergence of the system $A^{-1}\boldsymbol{y}$ using CG depends on the conditioning of A. Now, using the preconditioned linear system, the convergence depends on the conditioning of $P^{-1}A$. However, there is a catch. Even if P and A are symmetric and positive definite, this is not necessarily the case for $P^{-1}A$. With some mathematical trickery one can avoid this problem and derive the Untransformed Preconditioned Conjugate Gradient Method (derivations see (Shewchuk et al., 1994)) given by

Algorithm 6 Preconditioned Conjugate Gradient Method

input: $A, \boldsymbol{b}, P$
output: $A^{-1}\boldsymbol{b}$
$\boldsymbol{x}_{(0)} = 0$
$\boldsymbol{r}_{(0)} = \boldsymbol{b} - A\boldsymbol{x}_{(0)}$
$\boldsymbol{d}_{(0)} = P^{-1}\boldsymbol{r}_{(0)}$
For $i = 0, \ldots, p$ do :
    $\alpha_{(i)} = \dfrac{\boldsymbol{r}^t_{(i)}P^{-1}\boldsymbol{r}_{(i)}}{\boldsymbol{d}^t_{(i)}A\boldsymbol{d}_{(i)}}$
    $\boldsymbol{x}_{(i+1)} = \boldsymbol{x}_{(i)} + \alpha_{(i)}\boldsymbol{d}_{(i)}$
    $\boldsymbol{r}_{(i+1)} = \boldsymbol{r}_{(i)} - \alpha_{(i)}A\boldsymbol{d}_{(i)}$
    If $||\boldsymbol{r}_{(i+1)}||_2 < tolerance$, then return $\boldsymbol{x}_{(i+1)}$
    $\beta_{(i)} = \dfrac{\boldsymbol{r}^t_{(i+1)}P^{-1}\boldsymbol{r}_{(i+1)}}{\boldsymbol{r}^t_{(i)}P^{-1}\boldsymbol{r}_{(i)}}$
    $\boldsymbol{d}_{(i+1)} = P^{-1}\boldsymbol{r}_{(i+1)} + \boldsymbol{\beta}_{(i+1)}\boldsymbol{d}_{(i)}$
Return $\boldsymbol{x}_{(i+1)}$:

It may be important to notice that the standard CG algorithm is a special case of the preconditioned CG algorithm by replacing P with I. Lemma (12) also holds for the preconditioned algorithm. Without going in detail, we propose the preconditioner given in definition (18).

Definition 18 (The Pivoted Cholesky Decomposition) *The pivoted Cholesky algorithm allows the computation of a low-rank approximation of a positive definite matrix $A \approx L_k L_k^t$ with k the number of steps.*

Hence, assuming that the pivoted Cholesky decomposition admits a good approximation, we find that $(L_k L_k^t)^{-1} A \approx I$ which is well conditioned. Hence, it makes sense to use the preconditioner $P_k = (L_k L_k^t)$.

11.21 Stochastic Trace Estimation

In this section, we follow (Hutchinson, 1990). One uses the following Monte-Carlo method for estimating the trace.

Lemma 13 (Stochastic Trace Estimation) *Let A be a $n \times n$ symmetric matrix with $\mathrm{Tr}(A) \neq 0$. Let $\boldsymbol{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n)^t$ be a vector consisting of n independent samples from a random variable Z with mean 0 and variance σ^2. Then $\boldsymbol{z}^t A \boldsymbol{z}$ is an unbiased estimator of $\sigma^2 \mathrm{Tr}(A)$ i.e.*

$$\mathbb{E}(\boldsymbol{z}^t A \boldsymbol{z}) = \sigma^2 \mathrm{Tr}(A). \tag{11.61}$$

Stochastic trace estimation has been proven very useful for determining the determinant of a matrix.

Lemma 14 (Trace Determinant Identity) *Denoting with $\log(A)$ the matrix logarithm of the square matrix A, we find that*

$$\log(|A|) = \mathrm{Tr}(\log(A)). \tag{11.62}$$

Now, using the Lanczos decomposition, we know that the eigenvalues of T equals the eigenvalues of A (see appendix section 11.18). Since the determinant of a matrix equals the product of its the eigenvalues, we find that the determinant of T equals the determinant of A. Subsequently, by combining lemma (13) with random variable Z with variance equal to one and lemma (14), we find that $\boldsymbol{z}^t \mathbf{log}\,(\boldsymbol{A}) \boldsymbol{z}$ is an unbaised estimator for $\log(|A|)$ i.e.

$$\log(|A|) = \log(|T|) = \mathrm{Tr}(\log(T)) = \mathbb{E}(\boldsymbol{z}^t \log(T) \boldsymbol{z}). \tag{11.63}$$

In order to calculate $\log(T)$, we need the lemma (15).

Lemma 15 (Matrix Exponentials Property) *Let S be a non-singular matrix. We have that*

$$\log(SAS^{-1}) = S \log(A) S^{-1} \tag{11.64}$$

or equivalently

$$(Se^A S^{-1}) = \exp(SAS^{-1}). \tag{11.65}$$

Proof: *We prove the last equation by noting that*

$$(Se^A S^{-1}) = S\left(I + \sum_{i=1}^{\infty} \frac{A^i}{i!}\right) S^{-1} = I + \sum_{i=1}^{\infty} \frac{(SAS^{-1})^i}{i!} = \exp(SAS^{-1}). \tag{11.66}$$

□

Now, tridiagonal matrices of dimension n can be eigendecomposed in $O(n^2)$ time. Using the eigendecomposition $T = V\Lambda V^t$, we can calculate the logarithm by using $\log(T) = V\log(\Lambda)V^t$.

11.22 Maximum A Posteriori (MAP) Estimation

In this section, we use some ideas from (Andrieu et al., 2003). The MAP estimator $\hat{\theta}_{\text{map}}$ is given by

$$\hat{\theta}_{\text{map}}(\boldsymbol{y}) = \max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\boldsymbol{y}) = \max_{\theta} \frac{p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int p(\boldsymbol{y}|\boldsymbol{\theta})} p(\boldsymbol{\theta})d\boldsymbol{\theta} = \max_{\theta} p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta}). \tag{11.67}$$

Notice that the MAP is a point estimator and does not give any information concerning the uncertainty of the estimations. This estimator can give a local optimum of the parameters. It is possible that (often with high dimensional posteriors) there exist areas of high density but with a low total probability (i.e. a very small peaks). We are again doing an optimisation problem, but we just include a prior on the hyperparameters and can for example be solved using the (L)-BFGS methods (see appendix section 11.8). Other popular methods are simulated annealing and Monte Carlo EM which are based on sampling (see appendix section 11.23) and can be found in (Andrieu et al., 2003).

11.23 Markov Chain Monte Carlo Methods

In this section we follow (Andrieu et al., 2003), (Betancourt, 2017), (Fichtner et al., 2018) and (Hoffman and Gelman, 2014). The first gives an overview of different Bayesian techniques, the second a conceptual introduction to Hamiltonian Monte Carlo (HMC), the third is a more dense introduction and the forth is the paper proposing the no-U-turn sampler (NUTS).

Suppose that a set $\boldsymbol{\theta}_i \in \Theta$ with $i \in (1, \ldots, n)$ is sampled from a target density $p(\boldsymbol{\theta})$. These n samples can be used to approximate the target density (f.e. with smoothing techniques) which is the Monte Carlo principle. This also implies that for a function f

$$\frac{1}{n}\sum_{i=1}^{n} f(\boldsymbol{\theta}_i) \xrightarrow{\text{a.s.}} \int_{\Theta} f(\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}. \tag{11.68}$$

MCMC algorithms explore the space Θ using a Markov chain mechanism which is constructed such that the chain mimics sampling from $p(\boldsymbol{\theta})$. It is obvious that it has to be possible that this chain can visit all the states and that the chain should not get trapped in cycles.

Now we are ready to discuss the Metropolis-Hastings (MH) algorithm which is a very popular MCMC method. Denote with $q(\boldsymbol{\theta}^*|\boldsymbol{\theta})$ a proposal distribution from which it is easy to sample with $\boldsymbol{\theta}^*$ a sampling candidate and $\boldsymbol{\theta}$ the current value of the chain. We denote with $\boldsymbol{\theta}_0$ the initial value. The MH algorithm is given by algorithm 7.

Algorithm 7 Metropolis-Hastings (MH)

input: $q, \boldsymbol{\theta}_0$
output: $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$
For $i = 1, \ldots, n$ do :
  Sample $u \sim \mathcal{Z}_{(0,1)}$
  Sample $\boldsymbol{\theta}^* \sim q(\boldsymbol{\theta}^* | \boldsymbol{\theta}_i)$
  If $u < \min\left\{1, \frac{p(\boldsymbol{\theta}^*) q(\boldsymbol{\theta}_i | \boldsymbol{\theta}^*)}{p(\boldsymbol{\theta}_i) q(\boldsymbol{\theta}^* | \boldsymbol{\theta}_i)}\right\}$
    $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}^*$
  Else
    $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i$

If the proposal distribution is a symmetric distribution (f.e. normal), we simplify the algorithm by deleting the factors concerning q in the fraction. This is called Random Walk Metropolis. However, one has to pay attention since different proposal distributions leads to different results. For example, if the proposal distribution is too narrow, only one mode of $p(\boldsymbol{\theta})$ might be visited. However, if it is too wide, the rejection rate can be very high leading to slow mixing and high correlations. Next, it is recommended to use a burn-in. This means that we delete the first m samples since the initial value was arbitrarily chosen. Finally, tuning the chains to make them mix well is hard for high dimensions. The rejection rate can become dramatic in high dimensions and the chain often remains on the same position for a long time ruining the mixing. This phenomenon is a consequence of the curse of dimensionality. The 'volume' of the parameter space ($d\boldsymbol{\theta}$) behaves very differently from the density ($p(\boldsymbol{\theta})$). For example, we can consider the extreme case where the density is largest around the mode but there is not a lot of volume. If it is possible to simulate from the conditional distributions, then Gibbs sampling is a popular method to turn a high-dimensional problem in several one-dimensional problems.

We discuss hybrid/Hamiltonian Monte Carlo (HMC) which uses gradient information of the target distribution to improve mixing in high dimensions. In (Betancourt, 2017), one uses the concept of typical set, were $p(\boldsymbol{\theta})d\boldsymbol{\theta}$ is large. Random walk Metropolis leaves the typical set and the acceptance probability becomes negligible. One solution is to shrink the size of the proposal, although this makes the Markov chain move very slowly. In Hybrid Monte Carlo, we do not draw data randomly but we use a proposal influenced by the data. We try to follow a Hamiltonian trajectory (see figure (11.3)).

Next, we introduce the momentum parameter $\boldsymbol{p}$ which we will use in our calculations but which we marginalise out to recover our target distribution. We lift the target distribution $p(\boldsymbol{\theta})$ onto a joint probability distribution $p_c(\boldsymbol{\theta}, \boldsymbol{p})$ on the phase space $(\boldsymbol{\theta}, \boldsymbol{p})$ which we call the canonical distribution. Denoting with $H(\boldsymbol{\theta}, \boldsymbol{p})$ the invariant Hamiltonian function, the canonical distribution is defined as

$$p_c(\boldsymbol{\theta}, \boldsymbol{p}) = \exp(-H(\boldsymbol{\theta}, \boldsymbol{p})). \tag{11.69}$$

Hence, we find that

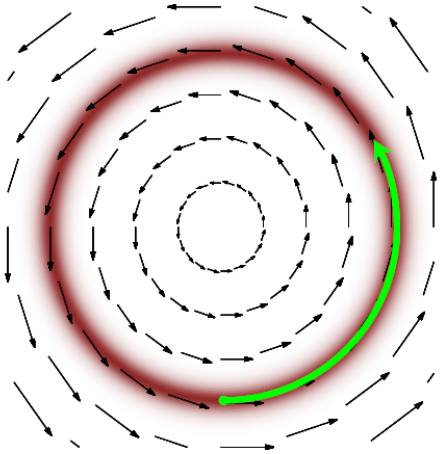


Figure 11.3: Plot from (Betancourt, 2017). The red band is the typical set. The arrows describe directions aligned with the typical set which guids the sampling algorithm.

$$H(\boldsymbol{\theta}, \boldsymbol{p}) = -\log(p_c(\boldsymbol{\theta}, \boldsymbol{p})) = -\log(p(\boldsymbol{p}|\boldsymbol{\theta})) - \log(p(\boldsymbol{\theta})) = K(\boldsymbol{p}, \boldsymbol{\theta}) + V(\boldsymbol{\theta}), \tag{11.70}$$

where we use the physical analogy with $K(\boldsymbol{p}, \boldsymbol{\theta}) = -\log(p(\boldsymbol{p}|\boldsymbol{\theta}))$ the kinetic energy and $V(\boldsymbol{\theta}) = -\log(p(\boldsymbol{\theta}))$ the potential energy. Hence, we can use the equations of Hamilton given by

$$\begin{aligned} \frac{d\boldsymbol{\theta}}{dt} &= \frac{\partial H}{\partial \boldsymbol{p}} \\ \frac{d\boldsymbol{p}}{dt} &= -\frac{\partial H}{\partial \boldsymbol{\theta}}. \end{aligned} \tag{11.71}$$

Hamiltonian systems have some interesting properties like time reversibility and volume preservation in the phase space. In order to solve this system, we use symplectic integrators, since they preserve phase space volume. We use the leapfrog integrator which we directly include in the algorithm of HMC. We denote with ε the step size and with L the length of the Leapfrog integrator. The new state $(\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{p}})$, which is the output of the leapfrog, is accepted with probability

$$\min\left(1, \frac{p_c(\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{p}})}{p_c(\boldsymbol{\theta}, \boldsymbol{p})}\right) = \min\left(1, \frac{\exp(-V(\tilde{\boldsymbol{\theta}}) - K(\tilde{\boldsymbol{p}}, \tilde{\boldsymbol{\theta}}))}{\exp(-V(\boldsymbol{\theta}) - K(\boldsymbol{p}, \boldsymbol{\theta}))}\right). \tag{11.72}$$

One also needs to choose a kinetic energy function in order to obtain a sample to start our trajectory with. We keep it easy and take

$$K(\boldsymbol{p}, \boldsymbol{\theta}) = -\log(p(\boldsymbol{p}|\boldsymbol{\theta})) = \mathcal{N}(\boldsymbol{p}|\mathbf{0}, I) = \frac{1}{2}\boldsymbol{p}^t\boldsymbol{p} + \text{cte}, \tag{11.73}$$

which we use M times. More possibilities can be found in (Betancourt, 2017).

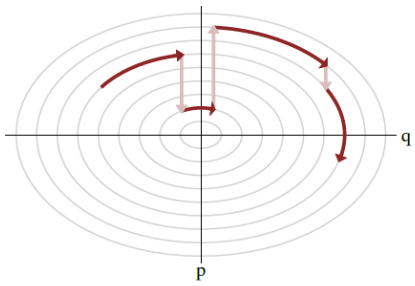


Figure 11.4: Plot from (Betancourt, 2017). The pink arrows represent the momentum resampling step and the red arrows the trajectories accros these levels with leapfrog.

Before reading algorithm of HMC, figure (11.4) may be clarifying. Now, the algorithm for HMC, following (Hoffman and Gelman, 2014), can be found in algorithm 8.

Algorithm 8 Hamiltonian Monte Carlo (HMC)

input: $\boldsymbol{\theta}_0, \varepsilon, L, M$
output: $\boldsymbol{\theta}, \boldsymbol{p}$
For $m = 1, \ldots, M$ do :
  Sample $\boldsymbol{p}_0 \sim \mathcal{N}(\boldsymbol{p}|\mathbf{0}, I)$
  $\boldsymbol{\theta}_m = \boldsymbol{\theta}_{m-1}, \tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}_{m-1}, \tilde{\boldsymbol{p}} = \boldsymbol{p}_0$
  For $i = 1, \ldots, L$ do:
    $\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{p}} = \mathit{Leapfrog}(\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{p}}, \varepsilon)$
  With probability $\alpha = \left(1, \dfrac{\exp(\log(p(\tilde{\boldsymbol{\theta}}) - \frac{1}{2}\tilde{\boldsymbol{p}}^t\tilde{\boldsymbol{p}})}{\exp(\log(p(\boldsymbol{\theta}_{m-1}) - \frac{1}{2}\boldsymbol{p}_0^t\boldsymbol{p}_0)}\right)$, set $\boldsymbol{\theta}_m = \tilde{\boldsymbol{\theta}}$, $\boldsymbol{p}_m = -\tilde{\boldsymbol{p}}$

function: Leapfrog$(\boldsymbol{\theta}, \boldsymbol{p}, \varepsilon)$
  $\tilde{\boldsymbol{p}} = \boldsymbol{p} + \dfrac{\varepsilon}{2}\nabla_{\boldsymbol{\theta}} \log(p(\boldsymbol{\theta})$
  $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} + \varepsilon\tilde{\boldsymbol{p}}$
  $\tilde{\boldsymbol{p}} = \tilde{\boldsymbol{p}} + \dfrac{\varepsilon}{2}\nabla_{\boldsymbol{\theta}} \log(p(\tilde{\boldsymbol{\theta}})$
  return: $\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{p}}$

In the proposal, we negated $\tilde{\boldsymbol{p}}$ which is necessary to have time reversibility. Notice that this step can be skipped since we are only interested in sampling from $p(\boldsymbol{\theta})$.

Choosing suitable values for L and ε is very important in order to obtain a good performance. If L is too small, we obtain random walk behaviour and slow mixing. In the opposite case, when L is too large, we generate trajectories that loop back and reiterate the steps. Next, if ε is too small, we need to do a lot of computations since we need to calculate a new value of the gradient every loop. If ε is too large, the simulation will be inaccurate and we encounter low acceptance rates. Hence, we use an extension of HMC called the no-U-turn sampler (NUTS).

Interested reader can find the derivation of NUTS in (Hoffman and Gelman, 2014)) and some extra plots and explanations in (Betancourt, 2017). Without going to technical, NUTS traces out a path forward and backwards in a fictitious time. First, it runs one step

backwards or forwards, then two steps backwards or forwards, then four steps backwards or forwards etc. We stop the doubling when the leftmost or rightmost nodes of any balanced subtree of the overall binary tree starts to double back on itself. A tree is balanced if the subtrees are balanced and the height of the subtrees differ by at most one. If the trajectory begins to loop back, it is likely that it retraces its steps and the simulation is wasteful. More rigorously, the doubling process stops if for one of the subtrees the states $\boldsymbol{\theta}^-, \boldsymbol{p}^-$ (leftmost leave) and $\boldsymbol{\theta}^+, \boldsymbol{p}^+$ (rightmost leave) we have that

$$(\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-)\boldsymbol{p}^- < 0 \quad \text{or} \quad (\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-)\boldsymbol{p}^+ < 0. \tag{11.74}$$

In other words, if an infinitesimal amount forward or backward in time reduces the distance between $\boldsymbol{\theta}^-$ and $\boldsymbol{\theta}^+$, we stop. Much more information and extra modifications can be found in the sources.

11.24 Parameter Space Financial Datasets

In this section, one can find the parameter space of the data study using machine learning techniques in order to price different types of options.

	Product/Market	Heston
Training set		
	$K : 40\% \to 160\%$	$\kappa : 1.4 \to 2.6$
	$T : 11M \to 1Y$	$\rho : -0.85 \to -0.55$
	$r : 1.5\% \to 2.5\%$	$\theta : 0.45 \to 0.75$
	$q : 0\% \to 5\%$	$\eta : 0.01 \to 0.1$
		$v_0 : 0.01 \to 0.1$
Test set		
	$K : 50\% \to 150\%$	$\kappa : 1.5 \to 2.5$
	$T : 11M \to 1Y$	$\rho : -0.8 \to -0.6$
	$r : 1.5\% \to 2.5\%$	$\theta : 0.5 \to 0.7$
	$q : 0\% \to 5\%$	$\eta : 0.02 \to 0.1$
		$v_0 : 0.02 \to 0.1$

Table 11.1: Parameter ranges training and validation vanilla options.

	Product/Market	Heston
Training set		
	$K : 40\% \to 160\%$	$\kappa : 1.4 \to 2.6$
	$T : 11M \to 1Y$	$\rho : -0.85 \to -0.55$
	$r : 1.5\% \to 2.5\%$	$\theta : 0.35 \to 0.75$
	$q : 0\% \to 5\%$	$\eta : 0.01 \to 0.16$
	$H : 0.55 \to 0.99$	$v_0 : 0.01 \to 0.16$
Test set		
	$K : 50\% \to 150\%$	$\kappa : 1.5 \to 2.5$
	$T : 11M \to 1Y$	$\rho : -0.8 \to -0.6$
	$r : 1.5\% \to 2.5\%$	$\theta : 0.4 \to 0.7$
	$q : 0\% \to 5\%$	$\eta : 0.02 \to 0.16$
		$v_0 : 0.02 \to 0.16$

Table 11.2: Parameter ranges training and validation for DOBP/DIBP options .

	Product/Market
Training set	
	$K : 40\% \rightarrow 160\%$
	$T : 1.5M \rightarrow 12.5M$
	r: $2\% \rightarrow 3\%$
	q: $0\% \rightarrow 5\%$
	$\sigma : 0.05 \rightarrow 1.05$
Test set	
	$K : 50\% \rightarrow 150\%$
	$T : 2M \rightarrow 12M$
	$r : 2\% \rightarrow 3\%$
	$q : 0\% \rightarrow 5\%$
	$\sigma : 0.1 \rightarrow 1$

Table 11.3: Parameter ranges training and validation for American options.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th Usinex Symposium on Operating Systems Design and Implementation)*, pages 265–283.

Ambikasaran, S., Foreman-Mackey, D., Greengard, L., Hogg, D. W., and O'Neil, M. (2014). Fast direct methods for Gaussian processes. *arXiv preprint arXiv:1403.6015*.

Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. (2003). An introduction to MCMC for machine learning. *Machine learning*, 50(1-2):5–43.

Arbenz, P., Kressner, D., and Zürich, D. (2012). Lecture notes on solving large scale eigenvalue problems. *D-MATH, EHT Zurich*, 2.

Babak, O. and Deutsch, C. V. (2009). Statistical approach to inverse distance interpolation. *Stochastic Environmental Research and Risk Assessment*, 23(5):543–553.

Bauer, M., van der Wilk, M., and Rasmussen, C. E. (2016). Understanding probabilistic sparse Gaussian process approximations. In *Advances in neural information processing systems*, pages 1533–1541.

Betancourt, M. (2017). A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

Camilla, T. (2018). *Computational aspects of Gaussian process regression*. PhD thesis, University of Amsterdam.

Carr, P. and Madan, D. (1999). Option valuation using the fast Fourier transform. *Journal of computational finance*, 2(4):61–73.

Chen, Z. and Wang, B. (2018). How priors of initial hyperparameters affect Gaussian process regression models. *Neurocomputing*, 275:1702–1710.

Damianou, A. and Lawrence, N. (2013). Deep gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215.

De Spiegeleer, J., Madan, D. B., Reyners, S., and Schoutens, W. (2018). Machine learning for quantitative finance: fast derivative pricing, hedging and fitting. *Quantitative Finance*, 18(10):1635–1643.

Dong, K., Eriksson, D., Nickisch, H., Bindel, D., and Wilson, A. G. (2017). Scalable log determinants for Gaussian process kernel learning. In *Advances in Neural Information Processing Systems*, pages 6327–6337.

Duchi, J. (2007). Derivations for linear algebra and optimization. *Berkeley, California*, 3:2325–5870.

Fichtner, A., Zunino, A., and Gebraad, L. (2018). A tutorial introduction to the Hamiltonian Monte Carlo solution of weakly nonlinear inverse problems.

Filippone, M., Zhong, M., and Girolami, M. (2013). A comparative evaluation of stochastic-based inference methods for Gaussian process models. *Machine Learning*, 93(1):93–114.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. (2018a). GPytorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems*, pages 7576–7586.

Gardner, J. R., Pleiss, G., Wu, R., Weinberger, K. Q., and Wilson, A. G. (2018b). Product kernel interpolation for scalable Gaussian processes. *arXiv preprint arXiv:1802.08903*.

Gormley, M. (2017). Clustering (k-means). `https://www.cs.cmu.edu/~mgormley/courses/10601-s17/slides/lecture15-cluster.pdf`.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*.

Hensman, J., Matthews, A. G., Filippone, M., and Ghahramani, Z. (2015). MCMC for variationally sparse Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 1648–1656.

Heston, S. L. (1993). A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The review of financial studies*, 6(2):327–343.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347.

Hoffman, M. D. and Gelman, A. (2014). The no-u-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1):1593–1623.

Hutchinson, M. F. (1990). A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450.

Keys, R. (1981). Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, 29(6):1153–1160.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational dropout and the local reparameterization trick. In *Advances in neural information processing systems*, pages 2575–2583.

Krishnamoorthy, A. and Menon, D. (2013). Matrix inversion using Cholesky decomposition. In *Signal processing: algorithms, architectures, arrangements, and applications (SPA)*, pages 70–72. IEEE.

Kuss, M. (2006). *Gaussian process models for robust regression, Classification, and Reinforcement Learning*. PhD thesis, Technische Universität Darmstadt.

Lalchand, V. and Rasmussen, C. E. (2019). Approximate inference for fully Bayesian Gaussian process regression. *arXiv preprint arXiv:1912.13440*.

Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528.

Madan, D. B., Carr, P. P., and Chang, E. C. (1998). The variance gamma process and option pricing. *Review of Finance*, 2(1):79–105.

Murray, I. and Adams, R. P. (2010). Slice sampling covariance hyperparameters of latent Gaussian models. In *Advances in neural information processing systems*, pages 1732–1740.

Nielsen, M. A. (2015). *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, USA:.

Palczewski, J. (2016). Numerical schemes for SDEs. *The University of Leeds-School of Mathematics. Lecture notes MATH5350 (Computations in Finance)*.

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). An unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for machine learning*. the MIT Press: (Cambridge, MA).

Riley, K. F., Hobson, M. P., and Bence, S. J. (2006). *Mathematical methods for physics and engineering: a comprehensive guide*. Cambridge university press.

Saatçi, Y. (2012). *Scalable inference for structured Gaussian process models*. PhD thesis, Citeseer.

Salimbeni, H. and Deisenroth, M. (2017). Doubly stochastic variational inference for deep Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 4588–4599.

Salvatier, J., Wiecki, T. V., and Fonnesbeck, C. (2016). Probabilistic programming in Python using pymc3. *PeerJ Computer Science*, 2:e55.

Schölkopf, B., Herbrich, R., and Smola, A. J. (2001). A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer.

Schoutens, W. (2008). The world of VG. In *M. Vanmaele, D. Deelstra, A. De Schepper, J. Dhaene, en P. Van Goethem, editors, Handelingen Contactforum Actuarial and Financial Mathematics Conference, pagina's*, pages 3–54.

Sejdinovic, D. and Gretton, A. (2012). What is an RKHS?

Shewchuk, J. R. et al. (1994). An introduction to the conjugate gradient method without the agonizing pain.

Snelson, E. and Ghahramani, Z. (2006). Sparse Gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264.

Titsias, M. K. (2009). Variational model selection for sparse Gaussian process regression. *Report, University of Manchester, UK*.

Ubaru, S., Chen, J., and Saad, Y. (2017). Fast estimation of tr(f(a)) via stochastic lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1075–1099.

Wang, K., Pleiss, G., Gardner, J., Tyree, S., Weinberger, K. Q., and Wilson, A. G. (2019). Exact Gaussian processes on a million data points. In *Advances in Neural Information Processing Systems*, pages 14622–14632.

Welling, M. (2013). Kernel ridge regression. *Max Welling's Classnotes in Machine Learning*, pages 1–3.

Wilson, A. and Nickisch, H. (2015). Kernel interpolation for scalable structured Gaussian processes (kiss-gp). In *International Conference on Machine Learning*, pages 1775–1784.

Wilson, A. G. (2014). *Covariance kernels for fast automatic pattern discovery and extrapolation with Gaussian processes*. PhD thesis, University of Cambridge.

Wilson, A. G., Dann, C., and Nickisch, H. (2015). Thoughts on massively scalable Gaussian processes. *arXiv preprint arXiv:1511.01870*.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. (2016a). Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378.

Wilson, A. G., Hu, Z., Salakhutdinov, R. R., and Xing, E. P. (2016b). Stochastic variational deep kernel learning. In *Advances in Neural Information Processing Systems*, pages 2586–2594.

Yang, X. (2017). Understanding the variational lower bound.

DEPARTMENT MATHEMATICS
Celestijnenlaan 200b - bus 2400
3001 LEUVEN, BELGIË
tel. + 32 16 32 70 15
fax + 32 16 32 79 98
www.kuleuven.be