Aalto University School of Science and Technology
Department of Biomedical Engineering and Computational Science (BECS)

Espoo, Finland July 17, 2010

Gaussian processes in Bayesian modeling: Manual for Matlab toolbox GPstuff, Version 2.0

Jarno Vanhatalo, Jaakko Riihimäki,
Jouni Hatikainen, and Aki Vehtari

(This is an early version of the manual, which is still subject to some modifications. The text contains still errors in some details but great picture is correctly described.)

contact: jarno.vanhatalo@tkk.fi

Contents

Preface

This is a manual for software package GPstuff, which is a collection of Matlab functions to build and analyze Bayesian models build over Gaussian processes. The purpose of the manual is to help people to use the software in their own work and possibly modify and extend the features. The manual consist of two short introductory sections about Bayesian inference and Gaussian processes, which introduce the topic and the notation used throughout the book. The theory is not extensively covered and readers not familiar with it are suggested to see the references for a more complete discussion.

After the introductory part a whole chapter (2) is devoted for inference techniques. Gaussian process models lead to analytically unsolvable models for which reason efficient approximative methods are essential. The techniques implemented in GPstuff are introduced in general level and references are given to direct the reader for more detailed discussion on the methods.

The rest of the manual discusses the basic models implemented in the package and demonstrates their usage. This discussion begins from the chapter 3 which considers simple Gaussian process regression and classification problems. These examples serve also as examples how to use the basic functions in the GPstuff and all the rest of the examples build over the considerations in this chapter. The rest of the chapters concentrate on more special model constructions, such as sparse Gaussian processes, additive models, and various observations models. Also, functions for model assessment and comparison are discussed. All these considerations are more or less individual examples that can be read if needed. The essential parts that everyone should know are covered by the end of the chapter 3. The appendix collects technical details and lists of prior and function definitions.

This is an early version of the manual which is subject to minor modifications and addition of new material.

Chapter 1

Introduction

1.1 Bayesian modeling

Building a Bayesian model, denoted by $\mathcal{M}$, starts with an *observation model* which contains the description of the data generating process, or its approximation. The observation model is denoted by $p(\mathcal{D}|\theta, \mathcal{M})$, where θ stands for the parameters and $\mathcal{D}$ the observations. The observation model quantifies a conditional probability for data given the parameters, and when looked as a function of parameters it is called *likelihood*. If the parameter values were known, the observation model would contain all the knowledge of the phenomenon and could be used as such. If the observations contain randomness, sometimes called *noise*, one would still be uncertain of the future observations, but could not reduce this uncertainty since everything that can be known exactly would be encoded in the observation model. Usually, the parameter values are not known exactly but there is only limited knowledge on their possible values. This prior information is formulated mathematically by the *prior probability* $p(\theta|\mathcal{M})$, which reflects our beliefs and knowledge about the parameter values before observing data. Opposed to the aleatory uncertainty encoded in the observation model the epistemic uncertainty present in the prior can be reduced by gathering more information on the phenomenon (for a more illustrative discussion on the differences between these two sources of uncertainty see (O'Hagan, 2004)). Bayesian inference is the process of updating our prior knowledge based on new observations – in other words it is the process for reducing the epistemic uncertainty.

The cornerstone of Bayesian inference is the Bayes' theorem which defines the conditional probability of the parameters after observing the data

$$p(\theta|\mathcal{D}, \mathcal{M}) = \frac{p(\mathcal{D}|\theta, \mathcal{M})p(\theta|\mathcal{M})}{p(\mathcal{D}|\mathcal{M})} \tag{1.1}$$

This is called the *posterior distribution* and it contains all information about parameter θ conveyed from the data $\mathcal{D}$ by the model. The normalization constant

$$p(\mathcal{D}|\mathcal{M}) = \int p(\mathcal{D}|\theta, \mathcal{M})p(\theta|\mathcal{M})d\theta \tag{1.2}$$

is equal to the conditional probability that the data came from the model $\mathcal{M}$ given our model assumptions. It is also called the marginal likelihood for the model. The model, $\mathcal{M}$, stands for all the hypotheses and assumptions that are made about the phenomenon. It embodies the functional forms of the observation model and the prior,

which are always tied together, as well as our subjective assumptions used to define these mathematical abstractions. Because everything is conditioned on $\mathcal{M}$, it is a redundant symbol and as such omitted from this on. Usually we are not able to define 'correct' model and most of the time we have only limited ability to encode our prior beliefs in the mathematical formulation. Still, many models turn out to be useful in practice.

The true power of Bayesian approach comes from the possibility to construct and analyze hierarchical models. In hierarchical models, prior probabilities are appointed also for the parameters of the prior. Let us write the prior as $p(\theta|\eta)$, where η denotes the parameters of the prior distribution, *hyperparameters*. By setting a hyperprior, $p(\eta)$, for the hyperparameters we obtain a hierarchical model structure where the fixed parameter values move further away from the data. This allows more flexible models and leads to vaguer prior, which is beneficial if the modeller is unsure of the specific form of the prior. In theory the hierarchy could be extended as far as one desires but there are practical limits how many levels of hierarchy are reasonable (Goel and Degroot, 1981).

The models implemented in GPstuff are hierarchical models where the parameter θ is replaced by a latent function $f(\mathbf{x})$. The observation model is build such that an individual observation depends on a function value at a certain input location $\mathbf{x}$. The latent function $f(\mathbf{x})$ is given a Gaussian process (GP) prior (Rasmussen and Williams, 2006), whose properties are defined by a mean and covariance function, and the hyperparameters related to them. The hierarchy is continued to third level by giving a hyperprior for the covariance function parameters. The assumption is that there is a functional description for the studied phenomenon, which we are not aware of, and the observations are noisy realizations of this underlying function. The power of this construction lies in the flexibility and non-parametric form of the GP prior. We can use simple parametric observation models that describe the assumed observation noise. The assumptions about the functional form of the phenomenon are encoded in the GP prior. Since GP is a non-parametric model we do not need to fix the functional form of the latent function, but we can give implicit statements of it. These statements are encoded in the mean and covariance function, which determine, for example, the smoothness and variability of the function.

1.2 Gaussian process models

The general form of the models in GPstuff can be written as follows:

$$\text{observation model:} \qquad \mathbf{y} \sim \prod_{i=1}^{n} p(y_i|f_i,\gamma) \tag{1.3}$$

$$\text{GP prior:} \qquad f(\mathbf{x})|\theta \sim \mathcal{GP}\left(m(\mathbf{x}), k(\mathbf{x},\mathbf{x}'\,|\theta)\right) \tag{1.4}$$

$$\text{hyperprior:} \qquad \theta,\gamma \sim p(\theta)p(\gamma). \tag{1.5}$$

Here $\mathbf{y} = [y_1,...,y_n]^{\mathrm{T}}$ is a vector of observations (target values) at (input) locations $\mathbf{X} = \{\mathbf{x}_i = [x_{i,1},...,x_{i,d}]^{\mathrm{T}}\}_{i=1}^{n}$. $f(\mathbf{x})$ is a latent function with value $f_i = f(\mathbf{x}_i)$ at input location $\mathbf{x}_i$. The boldface notation will denote a set of latent variables in a vector $\mathbf{f} = [f_1,...,f_n]^{\mathrm{T}}$. Here, the inputs are real valued vectors $\mathbf{x} \in \Re^d$ but in general other inputs, such as strings or graphs, are possible as well. θ collects the hyperparameters in the covariance function $k(\mathbf{x},\mathbf{x}'\,|\theta)$, and γ collects the hyperparameters in the observation model $p(y_i|f_i,\gamma)$. The notation will be slightly abused since $p(y_i|\cdot)$ is used also for

the likelihood, which is the same equation as the observation model but a function of parameters instead of y_i. The mean function is considered zero, $m(\mathbf{x}) \equiv 0$, throughout the work since this simplifies the notation and the current implementation of GPstuff assumes that prior mean is set explicitly to zero. This constraint will be relaxed in the future though.

A Gaussian process is a type of a continuous stochastic process, which defines a probability distribution over functions. The advantage of using GPs is that we can conduct the inference directly in the function space $f(\mathbf{x}) : \Re^d \rightarrow \Re$. Formal definition for the process is given as (e.g. Rasmussen and Williams, 2006):

A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.

By definition, any set of function values $\mathbf{f}$, indexed by the input co-ordinates $\mathbf{X}$, have a multivariate Gaussian prior distribution

$$p(\mathbf{f}\,|\mathbf{X}, \theta) = \mathrm{N}(\mathbf{f}\,|\mathbf{0}, \mathbf{K}_{\mathrm{f,f}}), \tag{1.6}$$

where $\mathbf{K}_{\mathrm{f,f}}$ is the covariance matrix. Notice, that the prior over functions will be denoted by $\mathcal{GP}(\cdot,\cdot)$, whereas the prior over finite set of latent variables will be denoted by $\mathrm{N}(\cdot,\cdot)$. The covariance matrix is constructed from a covariance function, $[\mathbf{K}_{\mathrm{f,f}}]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j|\theta)$, which characterizes the correlations between different points in the process $\mathrm{E}[f(\mathbf{x}_i), f(\mathbf{x}_j)] = k(\mathbf{x}_i, \mathbf{x}_j|\theta)$ (remember that the prior mean is explicitly set zero in this work). Covariance function encodes the prior assumptions of the latent function, such as the smoothness and scale of the variation, and can be chosen freely as long as the covariance matrices produced are symmetric and positive semi-definite ($\mathbf{v}^{\mathrm{T}}\,\mathbf{K}_{\mathrm{f,f}}\,\mathbf{v} \geq 0, \forall \mathbf{v} \in \Re^n$). An example of a stationary covariance function is the squared exponential

$$k_{\mathrm{se}}(\mathbf{x}_i, \mathbf{x}_j\,|\theta) = \sigma^2_{\mathrm{se}} \exp\left(-\sum_{k=1}^{d}(x_{i,k} - x_{j,k})^2/l_k^2\right), \tag{1.7}$$

where $\theta = \{\sigma^2_{\mathrm{se}}, l_1, ..., l_d\}$. Here, σ^2_{se} is the scaling parameter, and l_k is the length-scale, which governs how fast the correlation decreases as the distance increases in the direction k. See Figure 1.1 for illustration. The squared exponential is only one example of possible covariance functions and other functions will be considered along the way through the text. All the covariance functions implemented in GPstuff are summarized in the Appendix .1. Discussion on common covariance functions and their properties is given by, for example, in (Diggle and Ribeiro, 2007; Finkenstädt et al., 2007; Rasmussen and Williams, 2006).

By definition, the marginal distribution of any subset of latent variables, the function values at fixed input points, can be constructed by simply taking the appropriate submatrix of the covariance and subvector of the mean. Imagine, that we want to predict the values $\tilde{\mathbf{f}}$ at new input locations $\tilde{\mathbf{X}}$. The joint prior for latent variables at observation $\mathbf{X}$ and prediction locations $\tilde{\mathbf{X}}$ is

$$\begin{bmatrix}\mathbf{f} \\ \tilde{\mathbf{f}}\end{bmatrix} |\theta \sim \mathrm{N}\left(\mathbf{0}, \begin{bmatrix}\mathbf{K}_{\mathrm{f,f}} & \mathbf{K}_{\mathrm{f},\tilde{\mathrm{f}}} \\ \mathbf{K}_{\tilde{\mathrm{f}},\mathrm{f}} & \mathbf{K}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}}\end{bmatrix}\right), \tag{1.8}$$

where $\mathbf{K}_{\mathrm{f,f}} = k(\mathbf{X}, \mathbf{X}|\theta)$, $\mathbf{K}_{\mathrm{f},\tilde{\mathrm{f}}} = k(\mathbf{X}, \tilde{\mathbf{X}}|\theta)$ and $\mathbf{K}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}} = k(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}|\theta)$. Here, the covariance function $k(\cdot,\cdot)$ denotes also vector and matrix valued functions $k(\mathbf{x}, \mathbf{X})$:

$\Re^d \times \Re^{d\times n} \rightarrow \Re^{1\times n}$, and $k(\mathbf{X},\mathbf{X}) : \Re^{d\times n} \times \Re^{d\times n} \rightarrow \Re^{n\times n}$. The marginal distribution of $\tilde{\mathbf{f}}$ is $p(\tilde{\mathbf{f}}|\tilde{\mathbf{X}},\theta) = \mathrm{N}(\tilde{\mathbf{f}}|\mathbf{0},\mathbf{K}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}})$ like the marginal distribution of $\mathbf{f}$ given in (1.6). This marginal is also called a prior predictive distribution since it is not conditioned to any observations. The conditional distribution of a set of latent variables given other set of latent variables is Gaussian as well. For example, the distribution of $\tilde{\mathbf{f}}$ given $\mathbf{f}$ is

$$\tilde{\mathbf{f}}|\,\mathbf{f},\theta \sim \mathrm{N}(\mathbf{K}_{\tilde{\mathrm{f}},\mathrm{f}}\,\mathbf{K}_{\mathrm{f,f}}^{-1}\,\mathbf{f}, \mathbf{K}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}} - \mathbf{K}_{\tilde{\mathrm{f}},\mathrm{f}}\,\mathbf{K}_{\mathrm{f,f}}^{-1}\,\mathbf{K}_{\mathrm{f},\tilde{\mathrm{f}}}), \tag{1.9}$$

which can be interpreted as the posterior predictive distribution for $\tilde{\mathbf{f}}$ after observing the function values at locations $\mathbf{X}$. The mean and covariance of the conditional distribution are functions of input vector $\tilde{\mathbf{x}}$ and $\mathbf{X}$ plays the role of fixed parameters. Thus, the above distribution generalizes to a Gaussian process with mean function $m_{\mathrm{p}}(\tilde{\mathbf{x}}) = k(\tilde{\mathbf{x}},\mathbf{X})\,\mathbf{K}_{\mathrm{f,f}}^{-1}\,\mathbf{f}$ and covariance $k_{\mathrm{p}}(\tilde{\mathbf{x}},\tilde{\mathbf{x}}') = k(\tilde{\mathbf{x}},\tilde{\mathbf{x}}') - k(\tilde{\mathbf{x}},\mathbf{X})\,\mathbf{K}_{\mathrm{f,f}}^{-1}\,k(\mathbf{X},\tilde{\mathbf{x}}')$, which define the posterior distribution of the latent function $f(\tilde{\mathbf{x}})$. The posterior GP is illustrated in Figure 1.2.

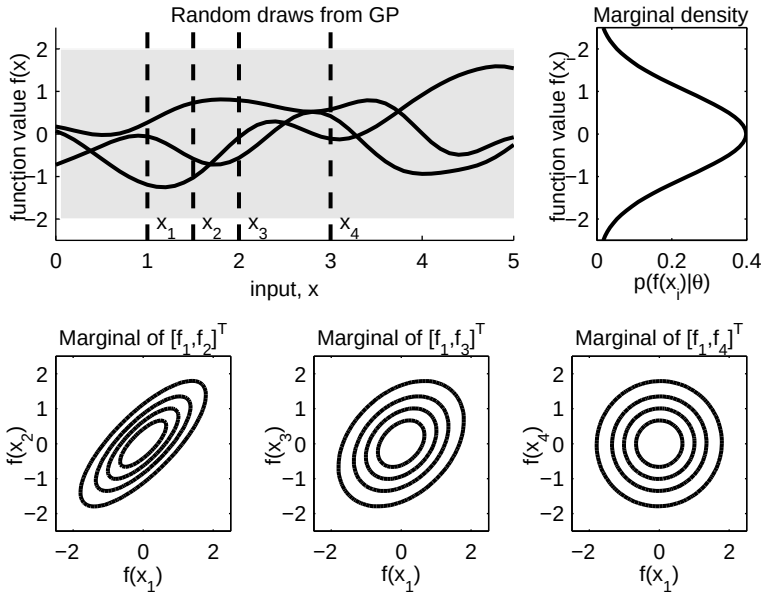


Figure 1.1: An illustration of a Gaussian process. The upper left figure presents three functions drawn randomly from a zero mean GP with squared exponential covariance function. The hyperparameters are $l = 1$ and $\sigma^2 = 1$ and the grey shading represents central 95% probability interval. The upper right subfigure presents the marginal distribution for a single function value. The lower subfigures present three marginal distributions between two function values at distinct input locations shown in the upper left subfigure by dashed line. It can be seen that the correlation between function values $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$ is the greater the closer $\mathbf{x}_i$ and $\mathbf{x}_j$ are to each others.

As will be seen, the class of models described by the equations (1.3)-(1.5) is rather rich. Even though the observation model is assumed to be factorizable given the latent variables $f(\mathbf{x}_1),...,f(\mathbf{x}_n)$, the correlations between the observations are incorporated into the model via the GP prior, and the marginalized observation model

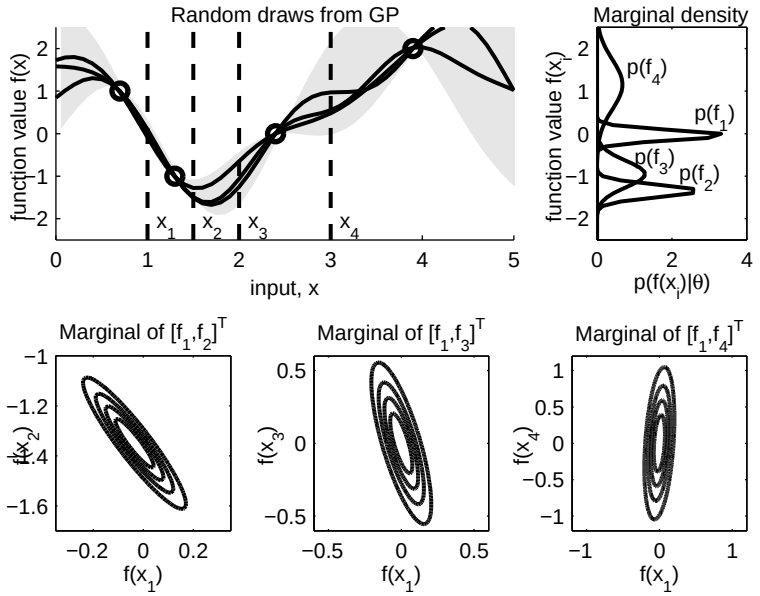


Figure 1.2: A conditional (posterior) GP $p(\tilde{f}|\,\mathbf{f},\theta)$. The observations $\mathbf{f} = [f(0.7) = 1, f(1.3) = -1, f(2.4) = 0, f(3.9) = 2]^{\mathrm{T}}$ are plotted with circles in the upper left subfigure and the prior GP is illustrated in the figure 1.1. When comparing the subfigures to the equivalent ones in Figure 1.1 we can see clear distinction between the marginal and the conditional GP. Here, all the function samples travel through the observations, the mean is no longer zero and the covariance is non-stationary.

$p(\mathbf{y}\,|\gamma,\theta) = \int d\,\mathbf{f}\,p(\mathbf{f}\,|\theta)\prod_{i=1}^{n} p(y_i|f_i,\gamma)$ is no longer factorizable. The models considered here are also rather old since utilizing Gaussian processes is certainly not a new invention. Early examples of their usage can be found, for example, in time series analysis and filtering (Wiener, 1949), and geostatistics (e.g. Matheron, 1973). GPs are still widely and actively used in these fields and usefull overviews are provided by Cressie (1993), Grewal and Andrews (2001), Diggle and Ribeiro (2007), and Gelfand et al. (2010). O'Hagan (1978) was one of the firsts to consider Gaussian processes in a general probabilistic modeling context. He provided a general theory of Gaussian process prediction and utilized it for a number of regression problems. This general regression framework was later rediscovered as an alternative for neural network models (Williams and Rasmussen, 1996; Rasmussen, 1996), and extended for other problems than regression (Neal, 1997; Williams and Barber, 1998). This machine learning perspective is comprehensively summarized by Rasmussen and Williams (2006).

Chapter 2

Inference and prediction

When conducting inference, the interest is in the posterior distributions of the hyperparameters and the latent function, as well as the predictive distribution of new observations (possibly at new input locations). In an ideal situation, all the desired distributions could be solved analytically, but unfortunately this is not possible. This chapter illustrates how the posterior distributions can be approximated. First are discussed methods for evaluating (or approximating) the conditional posterior of latent variables,

$$p(\mathbf{f}\,|\mathcal{D},\theta,\gamma) = \frac{p(\mathbf{y}\,|\,\mathbf{f},\gamma)p(\mathbf{f}\,|\theta)}{\int p(\mathbf{y}\,|\,\mathbf{f},\gamma)p(\mathbf{f}\,|\theta)d\,\mathbf{f}}, \tag{2.1}$$

where the problem is the integral over $\mathbf{f}$. Above, the training data is denoted by $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$. Section 2.2 treats the problem of marginalizing over the hyperparameters to obtain the marginal posterior distribution for the latent variables

$$p(\mathbf{f}\,|\mathcal{D}) = \int p(\mathbf{f}\,|\mathcal{D},\theta,\gamma)p(\theta,\gamma|\mathcal{D})d\theta d\gamma. \tag{2.2}$$

The question how to approximate the marginal posterior of the hyperparameters $p(\theta,\gamma|\mathcal{D})$ is left for less attention even though the topic is touched shortly in the section 2.2.

The above considerations generalize straightforwardly to the evaluation of the posterior predictive distribution of latent function, for which we may evaluate first the conditional posterior $p(\tilde{f}|\mathcal{D},\theta,\gamma,\tilde{\mathbf{x}})$ and then marginalize over the hyperparameters to obtain $p(\tilde{f}|\mathcal{D},\tilde{\mathbf{x}})$. The predictive distribution for new observations can then be evaluated for each $\tilde{y}_i$ separately since the observation model is assumed to be factorizable. Thus we need to be able to evaluate rather low dimensional integrals

$$p(\tilde{y}_i|\tilde{\mathbf{x}}_i,\mathcal{D}) = \int p(\tilde{y}_i|\tilde{f}_i,\gamma)p(\tilde{f}_i|\tilde{\mathbf{x}}_i,\mathcal{D})p(\gamma|\mathcal{D})d\tilde{f}_i d\gamma. \tag{2.3}$$

For many observation models, which do not have free parameters γ, this integral reduces to marginalization over $\tilde{f}_i$ only.

2.1 Conditional posterior of the latent function

2.1.1 The posterior mean and covariance

If the hyperparameters are considered fixed, GP's marginalization and conditionalization properties can be exploited, for example, in prediction. Assume that we have found

the conditional posterior distribution $p(\mathbf{f}\,|\mathcal{D},\theta)$, which, in general, is not Gaussian. We can then evaluate the posterior predictive mean simply by using the expression of the conditional mean $\mathrm{E}_{\tilde{\mathbf{f}}|\,\mathbf{f},\theta}[f(\tilde{\mathbf{x}})] = k(\tilde{\mathbf{x}},\mathbf{X})\,\mathbf{K}_{\mathrm{f,f}}^{-1}\,\mathbf{f}$ (see equation (1.9) and the text below it). Since this holds for any set of latent variables $\tilde{\mathbf{f}}$, we obtain a parametric posterior mean function

$$m_p[\tilde{\mathbf{x}}] = \int \mathrm{E}_{\tilde{\mathbf{f}}|\,\mathbf{f},\theta}[f(\tilde{\mathbf{x}})]p(\mathbf{f}\,|\mathcal{D},\theta)d\,\mathbf{f} = k(\tilde{\mathbf{x}},\mathbf{X})\,\mathbf{K}_{\mathrm{f,f}}^{-1}\,\mathrm{E}_{\mathbf{f}\,|\mathcal{D},\theta}[\mathbf{f}]. \tag{2.4}$$

The posterior predictive covariance between any set of latent variables, $\tilde{\mathbf{f}}$, can be evaluated from the relation (see, for example, Gelman et al., 2004, page 23 for justification)

$$\mathrm{Cov}_{\tilde{\mathbf{f}}|\mathcal{D},\theta}[\tilde{\mathbf{f}}] = \mathrm{E}_{\mathbf{f}\,|\mathcal{D},\theta}\left[\mathrm{Cov}_{\tilde{\mathbf{f}}|\,\mathbf{f}}[\tilde{\mathbf{f}}]\right] + \mathrm{Cov}_{\mathbf{f}\,|\mathcal{D},\theta}\left[\mathrm{E}_{\tilde{\mathbf{f}}|\,\mathbf{f}}[\tilde{\mathbf{f}}]\right], \tag{2.5}$$

where the first term simplifies to the conditional covariance in equation (1.9) and the second term can be written as $k(\tilde{\mathbf{x}},\mathbf{X})\,\mathbf{K}_{\mathrm{f,f}}^{-1}\,\mathrm{Cov}_{\mathbf{f}\,|\mathcal{D},\theta}[\mathbf{f}]\,\mathbf{K}_{\mathrm{f,f}}^{-1}\,k(\mathbf{X},\tilde{\mathbf{x}}')$. Plugging these into the equation and simplifying gives us the posterior covariance function

$$k_p(\tilde{\mathbf{x}},\tilde{\mathbf{x}}') = k(\tilde{\mathbf{x}},\tilde{\mathbf{x}}') - k(\tilde{\mathbf{x}},\mathbf{X})\left(\mathbf{K}_{\mathrm{f,f}}^{-1} - \mathbf{K}_{\mathrm{f,f}}^{-1}\,\mathrm{Cov}_{\mathbf{f}\,|\mathcal{D},\theta}[\mathbf{f}]\,\mathbf{K}_{\mathrm{f,f}}^{-1}\right)k(\mathbf{X},\tilde{\mathbf{x}}'). \tag{2.6}$$

Even if the exact posterior process, or in other words the posterior distribution $p(\tilde{f}|\mathcal{D},\theta)$, was not analytically solvable we can still evaluate its posterior mean and covariance functions easily, as long as we can solve the mean $\mathrm{E}_{\mathbf{f}\,|\mathcal{D},\theta}$ and covariance $\mathrm{Cov}_{\mathbf{f}\,|\mathcal{D},\theta}[\mathbf{f}]$. Following, for example, Csató and Opper (2002) the conditional posterior mean can be written as

$$\mathrm{E}_{\mathbf{f}\,|\mathcal{D},\theta}[\mathbf{f}] = \mathbf{K}_{\mathrm{f,f}}\,\frac{\int d\,\mathbf{f}\,p(\mathbf{f})\partial p(\mathbf{y}\,|\,\mathbf{f})/\partial\,\mathbf{f}}{p(\mathcal{D}|\theta)}, \tag{2.7}$$

and a similar result can be obtained for the covariance. The problem with the exact formulas is that the integrals in them cannot be computed exactly. The common practice to approximate the posterior distribution $p(\mathbf{f}\,|\mathcal{D},\theta)$ is either with Markov chain Monte Carlo (MCMC) (e.g. Neal, 1997, 1998; Diggle et al., 1998; Kuss and Rasmussen, 2005; Christensen et al., 2006) or by giving an analytic approximation to it (e.g. Williams and Barber, 1998; Gibbs and Mackay, 2000; Minka, 2001; Csató and Opper, 2002; Rue et al., 2009). The analytic approximations considered here assume a Gaussian form in which case it is natural to approximate the predictive distribution with Gaussian as well. In this case the equations (2.4) and (2.6) give its mean and covariance. The Gaussian approximation can be justified if the conditional posterior is unimodal, which it is if the likelihood is log concave (this can easily be seen by evaluating the Hessian of the posterior $p(\mathbf{f}\,|\mathcal{D},\theta)$), and there is enough data so that the posterior will be close to Gaussian. A pragmatic justification for using Gaussian approximation is that many times it suffices to approximate well the mean and variance of the latent variables. These on the other hand fully define Gaussian distribution and one can approximate the integrals over $\mathbf{f}$ by using the Gaussian form for their conditional posterior.

2.1.2 Gaussian observation model

A special case of an observation model, for which the conditional posterior of the latent variables can be evaluated analytically, is the Gaussian distribution, $y_i \sim \mathrm{N}(f_i,\sigma^2)$, where the parameter γ is replaced by the noise variance σ^2. In this case, both the likelihood and the prior are Gaussian functions of the latent variable, and we are able

to analytically integrate over $\mathbf{f}$ to obtain the marginal likelihood of the hyperparameters

$$p(\mathbf{y}\,|\theta,\sigma^2) = \mathrm{N}(\mathbf{y}\,|\mathbf{0},\mathbf{K}_{\mathrm{f,f}}+\sigma^2\mathbf{I}). \tag{2.8}$$

Setting this in the denominator of the equation (2.1), re-arranging the terms and simplifying the expression gives a Gaussian distribution also for the conditional posterior of the latent variables

$$\mathbf{f}\,|\mathcal{D},\theta,\sigma^2 \sim \mathrm{N}(\mathbf{K}_{\mathrm{f,f}}(\mathbf{K}_{\mathrm{f,f}}+\sigma^2\mathbf{I})^{-1}\,\mathbf{y}, \mathbf{K}_{\mathrm{f,f}}-\mathbf{K}_{\mathrm{f,f}}(\mathbf{K}_{\mathrm{f,f}}+\sigma^2\mathbf{I})^{-1}\,\mathbf{K}_{\mathrm{f,f}}). \tag{2.9}$$

Since the conditional posterior of $\mathbf{f}$ is Gaussian, the posterior process, or distribution $p(\tilde{f}|\mathcal{D})$, is also Gaussian. The predictive mean and covariance can be evaluated by placing the mean and covariance from (2.10) in the equations (2.4) and (2.6), after which we obtain the predictive distribution

$$\tilde{f}|\mathcal{D},\theta,\sigma^2 \sim \mathcal{GP}\left(m_{\mathrm{p}}(\tilde{\mathbf{x}}), k_{\mathrm{p}}(\tilde{\mathbf{x}},\tilde{\mathbf{x}}')\right) \tag{2.10}$$

where the mean and covariance are $m_{\mathrm{p}}(\tilde{\mathbf{x}}) = k(\tilde{\mathbf{x}},\mathbf{X})(\mathbf{K}_{\mathrm{f,f}}+\sigma^2\mathbf{I})^{-1}\,\mathbf{y}$ and $k_{\mathrm{p}}(\tilde{\mathbf{x}},\tilde{\mathbf{x}}') = k(\tilde{\mathbf{x}},\tilde{\mathbf{x}}') - k(\tilde{\mathbf{x}},\mathbf{X})(\mathbf{K}_{\mathrm{f,f}}+\sigma^2\mathbf{I})^{-1}k(\mathbf{X},\tilde{\mathbf{x}}')$. The predictive distribution for new observations $\tilde{\mathbf{y}}$ can be obtained by integrating $p(\tilde{\mathbf{y}}|\mathcal{D},\theta,\sigma^2) = \int p(\tilde{\mathbf{y}}|\tilde{\mathbf{f}},\sigma^2)p(\tilde{\mathbf{f}}|\mathcal{D},\theta,\sigma^2)d\tilde{\mathbf{f}}$. The result is, again, Gaussian with mean $\mathrm{E}_{\tilde{\mathbf{f}}|\mathcal{D},\theta}[\tilde{\mathbf{f}}]$ and covariance $\mathrm{Cov}_{\tilde{\mathbf{f}}|\mathcal{D},\theta}[\tilde{\mathbf{f}}] + \sigma^2\mathbf{I}$.

2.1.3 Laplace approximation

In the Laplace approximation the mean is approximated by the posterior mode of $\mathbf{f}$ and the covariance by the curvature of the log posterior at the mode. The approximation is constructed from the second order Taylor expansion of $\log p(\mathbf{f}\,|\,\mathbf{y},\theta)$ around the mode $\hat{\mathbf{f}}$, which gives a Gaussian approximation to the conditional posterior

$$p(\mathbf{f}\,|\mathcal{D},\theta,\gamma) \approx q(\mathbf{f}\,|\mathcal{D},\theta,\gamma) = \mathrm{N}(\mathbf{f}\,|\hat{\mathbf{f}},\boldsymbol{\Sigma}), \tag{2.11}$$

where $\hat{\mathbf{f}} = \arg\max_{\mathbf{f}} p(\mathbf{f}\,|\mathcal{D},\theta,\gamma)$ and $\boldsymbol{\Sigma}^{-1}$ is the Hessian of the negative log conditional posterior at the mode (Gelman et al., 2004; Rasmussen and Williams, 2006):

$$\boldsymbol{\Sigma}^{-1} = -\nabla\nabla\log p(\mathbf{f}\,|\mathcal{D},\theta,\gamma)|_{\mathbf{f}=\hat{\mathbf{f}}} = \mathbf{K}_{\mathrm{f,f}}^{-1}+\mathbf{W}, \tag{2.12}$$

where $\mathbf{W}$ is a diagonal matrix, since the observation model is factorizable, with entries $\mathbf{W}_{ii} = \nabla_{f_i}\nabla_{f_i}\log p(y|f_i,\gamma)|_{f_i=\hat{f}_i}$. Here, the approximation scheme is called the Laplace method following Williams and Barber (1998), but essentially the same approximation is named Gaussian approximation by Rue et al. (2009) in their Integrated nested Laplace approximation (INLA) scheme for Gaussian Markov random field models.

The posterior mean of $f(\tilde{\mathbf{x}})$ can be approximated from the equation (2.4) by replacing the posterior mean $\mathrm{E}_{\mathbf{f}\,|\mathcal{D},\theta}[\mathbf{f}]$ by $\hat{\mathbf{f}}$. The posterior covariance is approximated similarly by using $(\mathbf{K}_{\mathrm{f,f}}^{-1}+\mathbf{W})^{-1}$ in the place of $\mathrm{Cov}_{\mathbf{f}\,|\mathcal{D},\theta}[\mathbf{f}]$. Thus, after some rearrangements and using $\mathbf{K}_{\mathrm{f,f}}^{-1}\,\hat{\mathbf{f}} = \nabla\log p(\mathbf{y}\,|\,\mathbf{f})|_{\mathbf{f}=\hat{\mathbf{f}}}$, the approximate posterior predictive distribution is

$$\tilde{f}|\mathcal{D},\theta,\sigma^2 \sim \mathcal{GP}\left(m_{\mathrm{p}}(\tilde{\mathbf{x}}), k_{\mathrm{p}}(\tilde{\mathbf{x}},\tilde{\mathbf{x}}')\right), \tag{2.13}$$

where the mean and covariance are $m_{\mathrm{p}}(\tilde{\mathbf{x}}) = k(\tilde{\mathbf{x}},\mathbf{X})\nabla\log p(\mathbf{y}\,|\,\mathbf{f})|_{\mathbf{f}=\hat{\mathbf{f}}}$ and $k_{\mathrm{p}}(\tilde{\mathbf{x}},\tilde{\mathbf{x}}') = k(\tilde{\mathbf{x}},\tilde{\mathbf{x}}') - k(\tilde{\mathbf{x}},\mathbf{X})(\mathbf{K}_{\mathrm{f,f}}+\mathbf{W})^{-1}k(\mathbf{X},\tilde{\mathbf{x}}')$ The approximate conditional predictive densities of new observations $\tilde{y}_i$ can now be evaluated, for example, with quadrature integration over each $\tilde{f}_i$ separately

$$p(\tilde{y}_i|\mathcal{D},\theta,\gamma) \approx \int p(\tilde{y}_i|\tilde{f}_i,\gamma)q(\tilde{f}_i|\mathcal{D},\theta,\gamma)d\tilde{f}_i. \tag{2.14}$$

2.1.4 Expectation propagation algorithm

The Expectation propagation (EP) algorithm is a general method for approximating integrals over functions that factor into simple terms (Minka, 2001). The Laplace method constructs a Gaussian approximation at the posterior mode and approximates the posterior covariance via the curvature of the log density at that point. EP, for its part, tries to minimize the Kullback-Leibler divergence from the true posterior to its approximation. EP approximates the conditional posterior with

$$p(\mathbf{f}\,|\mathcal{D},\theta,\gamma) \approx q(\mathbf{f}\,|\mathcal{D},\theta,\gamma) = \frac{1}{Z_{\text{EP}}} p(\mathbf{f}\,|\theta) \prod_{i=1}^{n} t_i(f_i|\tilde{Z}_i,\tilde{\mu}_i,\tilde{\sigma}_i^2), \tag{2.15}$$

where the likelihood terms have been replaced by site functions $t_i(f_i|\tilde{Z}_i,\tilde{\mu}_i,\tilde{\sigma}_i^2) = \tilde{Z}_i\,\mathrm{N}(f_i|\tilde{\mu}_i,\tilde{\sigma}_i^2)$ and the normalizing constant by Z_{EP}.

EP algorithm proceeds as follows. First, the site parameters $\tilde{Z}_i$, $\tilde{\mu}_i$ and $\tilde{\sigma}_i^2$ are initialized after which they are updated sequentially. At each iteration, first the i'th site is removed from the i'th marginal posterior to obtain a cavity distribution $q_{-i}(f_i) = q(f_i|\mathcal{D},\theta)/t_i(f_i)$. Second step is to find a Gaussian $\hat{q}(f_i)$ to which the Kullback-Leibler divergence from the cavity distribution multiplied by the exact likelihood for that site is minimized $\hat{q}(f_i) = \arg\min_q \mathrm{KL}\left(q_{-i}(f_i)p(y_i|f_i)||q(f_i)\right)$. This is equivalent to matching the first and second moment between the two distributions (Seeger, 2005). The site terms $\tilde{Z}_i$ are scaling parameters which ensure that also the zeroth moment of the approximate and real posterior match, that is $Z_{\text{EP}} = p(\mathcal{D}|\theta,\gamma)$. After the moments are solved, the parameters of the local approximation t_i are updated so that the new marginal posterior $q_{-i}(f_i)t_i(f_i)$ matches with the moments of $\hat{q}(f_i)$. For last, the parameters of the approximate posterior (2.15) are updated to give

$$p(\mathbf{f}\,|\mathcal{D},\theta,\gamma) \approx \mathrm{N}(\mathbf{f}\,|\,\mathbf{K}_{\mathrm{f,f}}(\mathbf{K}_{\mathrm{f,f}}+\tilde{\Sigma})^{-1}\tilde{\mu}, \mathbf{K}_{\mathrm{f,f}} - \mathbf{K}_{\mathrm{f,f}}(\mathbf{K}_{\mathrm{f,f}}+\tilde{\Sigma})^{-1}\,\mathbf{K}_{\mathrm{f,f}}), \tag{2.16}$$

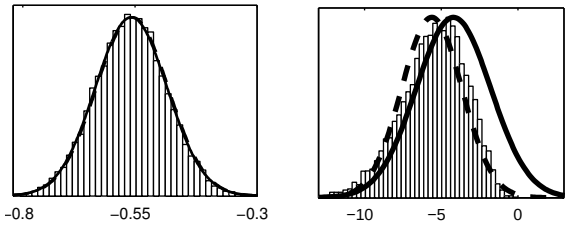
where $\tilde{\Sigma} = \mathrm{diag}[\tilde{\sigma}_1^2,...,\tilde{\sigma}_n^2]$. The predictive mean and covariance are again obtained from equations (2.4) and (2.6). The predictive distribution of new observations is derived analogically to the Laplace approximation.

From the equations (2.16), (2.13) and (2.10) it can be seen that there is great similarity between the exact solution with the Gaussian obsrvation model and the Laplace and EP approximation. The diagonal matrices $\mathbf{W}^{-1}$ and $\tilde{\mathbf{\Sigma}}$ correspond to the noise variance $\sigma^2\mathbf{I}$ in the Gaussian likelihood. Thus, the two approximations can be seen also as Gaussian approximation for the likelihood (Nickisch and Rasmussen, 2008).

2.1.5 Markov chain Monte Carlo

The accuracy of the two approximations considered this far is limited by the Gaussian form of the approximating function. Another approach, which gives exact solution in the limit of infinite computational time, is to approximate the posterior with Monte Carlo methods (Robert and Casella, 2004). These are based on sampling from $p(\mathbf{f}\,|\mathcal{D},\theta,\,\gamma)$ and using the samples to represent the posterior distribution. In this case, the posterior marginals can be visualized with histograms and posterior statistics approximated with sample means. For example, the posterior expectation of $\mathbf{f}$ is

$$\mathrm{E}_{\mathbf{f}\,|\mathcal{D},\theta,\gamma}[\mathbf{f}] \approx \frac{1}{M}\sum_{i=1}^{M}\mathbf{f}^{(i)}, \tag{2.17}$$



(a) Disease mapping (b) classification

Figure 2.1: Illustration of the Laplace approximation (solid line), EP (dashed line) and MCMC (histogram) for the conditional posterior of a latent variable $p(f_i|\mathcal{D},\theta)$ in two applications. On the left, a disease mapping problem with Poisson observation model (used in Vanhatalo et al., 2010) where the Gaussian approximation works well. On the right, a classification problem with probit likelihood (used in Vanhatalo and Vehtari, 2010) where the posterior is skewed and the Gaussian approximation is clearly a compromise. However, EP approximates the mean and variance better than the Laplace approximation in this case also.

where $\mathbf{f}^{(i)}$ is the i'th sample from the conditional posterior.

The problem with Monte Carlo methods is how to draw samples from arbitrary distributions. The challenge can be overcome with Markov chain Monte Carlo methods (Gilks et al., 1996), where one constructs a Markov chain whose stationary distribution is the posterior distribution $p(\mathbf{f}\,|\mathcal{D},\theta,\ \gamma)$ and uses the Markov chain samples to obtain Monte Carlo estimates. After having the posterior sample of latent variables, we can sample from the posterior predictive distribution of any set of latent variables $\tilde{\mathbf{f}}$ simply by sampling with each $\mathbf{f}^{(i)}$ one $\tilde{\mathbf{f}}^{(i)}$ from $p(\tilde{\mathbf{f}}|\,\mathbf{f}^{(i)},\theta,\gamma)$, which is given in the equation (1.9). Similarly, we can obtain a sample of $\tilde{\mathbf{y}}$ by drawing one $\tilde{\mathbf{y}}^{(i)}$ for each $\tilde{\mathbf{f}}^{(i)}$ from $p(\mathbf{y}\,|\tilde{\mathbf{f}},\theta,\gamma)$. A rather efficient sampling algorithm is hybrid Monte Carlo (HMC) (Duane et al., 1987; Neal, 1996), which utilizes the gradient information of the posterior distribution to direct the sampling in interesting regions. Significant improvement in mixing of the sample chain of the latent variables can be obtained by using the variable transformation discussed in (Christensen et al., 2006; Vanhatalo and Vehtari, 2007).

2.2 Marginalization over hyperparameters

2.2.1 Maximum a posterior estimate of hyperparameters

The easiest way to approximate the integral over $p(\theta,\gamma|\mathcal{D})$ is to give the hyperarameters a point estimate. A commonly used point estimate is the maximum a posterior (MAP) estimate

$$\{\hat{\theta},\hat{\gamma}\} = \arg\max_{\theta,\gamma} p(\theta,\gamma|\mathcal{D}) = \arg\min_{\theta,\gamma}\left[-\log p(\mathcal{D}|\theta,\gamma) - \log p(\theta,\gamma)\right]. \qquad (2.18)$$

In this approximation, the hyperparameter values are given a point mass one at the posterior mode, and the latent function marginal is approximated as $p(\mathbf{f}\,|\mathcal{D}) \approx p(\mathbf{f}\,|\mathcal{D},\hat{\theta},\hat{\gamma})$. Other way to interpret the hyperparameter optimization is model selection over a model family indexed by continuous parameter $[\theta^{\mathrm{T}},\gamma^{\mathrm{T}}]^{\mathrm{T}}$ (Rasmussen and Williams, 2006). In

this interpretation, GP is a fully non-parametric model, whose predictive distribution is defined entirely by the training data. This interpretation explains also why GPs are called non-parametric models. The negative log posterior cost function $-\log p(\mathcal{D}|\theta,\gamma)-\log p(\theta,\gamma)$ will be called also an *energy* in the text. This naming dates back to Markov chain Monte Carlo methods, where the log posterior cost function corresponds to its counterpart in physics problems where the cost function is really an energy.

To find the MAP estimate one needs to evaluate the log marginal likelihood. In Gaussian case this is straightforward since it has an analytic solution,

$$\log p(\mathcal{D}|\theta,\sigma) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\,\mathbf{K}_{\mathrm{f,f}} + \sigma^2\mathbf{I}| - \frac{1}{2}\,\mathbf{y}^{\mathrm{T}}(\mathbf{K}_{\mathrm{f,f}} + \sigma^2\mathbf{I})^{-1}\,\mathbf{y}, \quad (2.19)$$

The log marginal likelihood, and thus also the log posterior, is differentiable with respect to the hyperparameters, which allows a gradient based optimization.

If the observation model is not Gaussian the marginal likelihood needs to be approximated. The Laplace approximation to the marginal likelihood is constructed, for example, by writing

$$p(\mathcal{D}|\theta,\gamma) = \int p(\mathbf{y}|\,\mathbf{f},\gamma)p(\mathbf{f}\,|\theta)d\,\mathbf{f} = \int \exp(g(\mathbf{f}))d\,\mathbf{f}, \quad (2.20)$$

after which $g(\mathbf{f})$ is given a second order Taylor expansion around $\hat{\mathbf{f}}$. This gives a Gaussian integral over $\mathbf{f}$ multiplied by a constant, and results in the approximation

$$\log p(\mathcal{D}|\theta,\gamma) \approx \log q(\mathcal{D}|\theta,\gamma) \propto -\frac{1}{2}\hat{\mathbf{f}}^{\mathrm{T}}\,\mathbf{K}_{\mathrm{f,f}}^{-1}\,\hat{\mathbf{f}} + \log p(\mathbf{y}|\hat{\mathbf{f}}) - \frac{1}{2}\log|\mathbf{B}|, \quad (2.21)$$

where $|\mathbf{B}| = |I + \mathbf{W}^{1/2}\,\mathbf{K}_{\mathrm{f,f}}\,\mathbf{W}^{1/2}|$. This is the same approximation as the Gaussian approximation by Rue et al. (2009) derived from $p(\mathbf{y},\mathbf{f}\,.\theta,\gamma)/q(\mathbf{f}\,|\mathcal{D},\theta,\gamma)|_{\mathbf{f}=\hat{\mathbf{f}}}$, where the denominator is the Laplace approximation in equation (2.13) (see also Tierney and Kadane, 1986). The gradients of the approximate log marginal likelihood (2.21) can be solved analytically, which enables the use of gradient based optimization with Laplace approximation also.

EP's marginal likelihood approximation is its normalization constant

$$Z_{\mathrm{EP}} = \int p(\mathbf{f}\,|X,\theta)\prod_{i=1}^{n}\tilde{Z}_i\,\mathrm{N}(f_i|\tilde{\mu}_i,\tilde{\sigma}_i^2)df_i \quad (2.22)$$

in equation (2.15). This is a Gaussian integral multiplied by constant $\prod_{i=1}^{n}\tilde{Z}_i$, giving

$$\log Z_{\mathrm{EP}} = -\frac{1}{2}\log|K+\tilde{\Sigma}| - \frac{1}{2}\tilde{\mu}^{\mathrm{T}}\left(K+\tilde{\Sigma}\right)^{-1}\tilde{\mu} + C_{\mathrm{EP}}, \quad (2.23)$$

where C_{EP} collects terms that are not explicit functions of θ or γ (there is implicit dependence through the iterative algorithm, though). The parameters C_{EP}, $\tilde{\Sigma}$ and $\tilde{\mu}$ can be considered constants when differentiating the function with respect to the hyperparameters (Seeger, 2005), for which reason the MAP estimate can be found with gradient based optimization methods with EP also.

The advantage of MAP estimate is that it is relatively easy and fast to evaluate. Good optimization algorithms need usually at maximum tens of optimization steps to find the mode. The drawback, however, is that it may underestimate the uncertainty in hyperparameters.

2.2.2 Grid integration

Previous section treated methods to evaluate exactly (the Gaussian case) or approximately (Laplace approximation and EP) the marginal posterior $p(\theta, \gamma|\mathcal{D})$ up to the normalization. There the unnormalized posterior was used for optimizing the hyperparameters but it can also be used for exploring the posterior for purposes of numerical integration with a finite sum, such as

$$p(\mathbf{f}\,|\mathcal{D}) \approx \sum_{i=1}^{M} p(\mathbf{f}\,|\mathcal{D}, \vartheta_i) p(\vartheta_i|\mathcal{D}) \Delta_i. \tag{2.24}$$

Here $\vartheta = [\theta^{\mathrm{T}}, \gamma^{\mathrm{T}}]^{\mathrm{T}}$ and Δ_i denotes the area weight appointed to an evaluation point ϑ_i. Thus, the latent variable posterior is a mixture of Gaussians. The other marginal posteriors are approximated similarly with mixture distributions.

The first step in exploring $\log p(\vartheta|\mathcal{D})$ is to find its posterior mode as described in the previous section. After this we evaluate the negative Hessian of $\log p(\vartheta|\mathcal{D})$ at the mode, which would be the inverse covariance matrix for ϑ if the density were Gaussian. The exploration is aided using standardized variables $\mathbf{z}$. If $\mathbf{P}$ is the inverse Hessian (the approximate covariance) with eigendecomposition $\mathbf{P} = \mathbf{U}\mathbf{C}\mathbf{U}^{\mathrm{T}}$ then ϑ can be defined via $\mathbf{z}$ as

$$\vartheta(\mathbf{z}) = \hat{\vartheta} + \mathbf{U}\mathbf{C}^{1/2}\mathbf{z}. \tag{2.25}$$

If $p(\vartheta|\mathcal{D})$ were a Gaussian density, then $\mathbf{z}$ would be zero mean normal distributed. This re-parametrization corrects for scale and rotation and simplifies the integration (Rue et al., 2009). The exploration of $\log p(\vartheta|\mathcal{D})$ is started from the mode $\hat{\vartheta}$ and continued so that the bulk of the posterior mass is included in the integration. The grid points are set evenly along the directions $\mathbf{z}$, for which reason the area weights Δ_i are equal. This is illustrated in Figure 2.2(a) and discussed in (Rue et al., 2009; Vanhatalo et al., 2010).

The numerical integration using the grid search is feasible only for a small number of hyperparameters since the number of grid points grows exponentially with the dimension of the hyperparameter space d. For example, the number of the nearest neighbors of the mode increases as $O(3^d)$, which results in 728 grid points already for $d = 6$. If also the second neighbors are included, the number of grid points increases as $O(5^d)$, which results in 15624 grid points for six hyperparameters.

2.2.3 Monte Carlo integration

Monte Carlo integration works better than the grid integration in large hyperparameter spaces since its error decreases with a rate that is independent of the dimension (Robert and Casella, 2004). There are two options to find a Monte Carlo estimate for marginal posteriors, like $p(\mathbf{f}\,|\mathcal{D})$. The first option is to sample just the hyperparameters from their marginal posterior $p(\theta|\mathcal{D})$ or from its approximation given by the Laplace approximation or EP, which is illustrated in Figure 2.2(b). In this case, the posterior marginals are approximated with mixture distributions as in the grid integration but with equal weigths. The other option is to run a full MCMC for all the parameters in the model. That is, we sample both the hyperparameters and the latent variables and estimate the needed posterior statistics by sample estimates or by histograms (Neal, 1997; Diggle et al., 1998). The full MCMC is performed by alternate sampling from the conditional posteriors $p(\mathbf{f}\,|\mathcal{D}, \vartheta)$ and $p(\vartheta|\mathcal{D}, \mathbf{f})$. Sampling both, the hyperparameters and latent variables, is usually awfully slow since there is a strong correlation

between them. This slows the convergence and mixing of the Markov chain (Vanhatalo and Vehtari, 2007; Vanhatalo et al., 2010). Sampling from the (approximate) marginal, $p(\theta|\mathcal{D})$, is a much easier task since the parameter space is smaller. Tuning the sampler parameters is also the harder the more parameters are sampled.

The hyperparameters can be sampled from their marginal posterior (or its approximation) either with HMC, slice sampling (SLS) or via importance sampling. In importance sampling, we use a Normal or Student-t distribution with mean $\hat{\vartheta}$ and covariance $\mathbf{P}$ approximated with the negative Hessian of the log posterior as a proposal distribution. We sample from the proposal distribution $g(\gamma) \sim \mathrm{N}(\hat{\gamma}, \mathbf{P})$ (or $g(\gamma) \sim t_\nu(\hat{\gamma}, \mathbf{P})$) and approximate the integral with

$$p(\mathbf{f}\,|\mathcal{D}) \approx \frac{1}{\sum_{i=1}^{M} w_i} \sum_{i=1}^{M} q(\mathbf{f}\,|\mathcal{D}, \gamma_i) w_i, \tag{2.26}$$

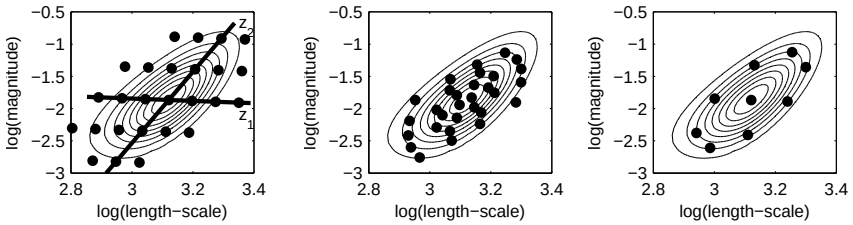
where $w_i = q(\gamma^{(i)})/g(\gamma^{(i)})$ are the importance weights. The normal proposal distribution is illustrated in the figure 2.2(b). Importance sampling is adequate only if the importance weights do not vary substantially and, thus, the goodness of the Monte Carlo integration can be monitored using the importance weights. The worst scenario occurs when the importance weights are small with high probability and with small probability get very large values (that is the tails of q are much wider than those of g). In order to detect problems it is usefull to monitor the cumulative normalized weights and the estimate of the effective sample size $M_{\text{eff}} = 1/\sum_{i=1}^{M} \hat{w}_i^2$, where $\hat{w}_i = w_i/\sum w_i$ Geweke (1989); Gelman et al. (2004); Vehtari (2001).

In some situations the naive Gaussian or Student-t proposal distribution is not adequate since the posterior distribution $q(\gamma|\mathcal{D})$ may be non-symmetric or the covariance estimate $\mathbf{P}$ is poor. In these situations, we use the scaled Student-t proposal distribution, proposed by Geweke (1989). In this approach, the scale of the proposal distribution is adjusted along each main direction defined by $\mathbf{P}$ so that the importance weights are maximized.

Although Monte Carlo integration is more efficient than grid integration, it also has its downside. For most examples, few hundred independent samples are enough for reasonable posterior summaries (Gelman et al., 2004), which seems achievable. The problem, however, is that we are not able to draw independent samples from the posterior. Even with a careful tuning of Markov chain samplers the autocorrelation is usually so large that the required sample size is in thousands, which is a clear disadvantage compared with the MAP estimate, for example.

2.2.4 Central composite design

Rue et al. (2009) suggest a central composite design (CCD) for choosing the representative points from the posterior of the hyperparameters when the dimensionality of the hyperparameters, d, is moderate or high. In this setting, the integration is considered as a quadratic design problem in a d dimensional space with the aim in finding points that allow for estimating the curvature of the posterior distribution around the mode. The design used by Rue et al. (2009) and Vanhatalo et al. (2010) is the fractional factorial design augmented with a center point and a group of $2d$ star points. In this setting, the design points are all on the surface of a d-dimensional sphere and the star points consist of $2d$ points along each axis. This is illustrated in Figure 2.2(c). The number of the design points grows very moderately and, for example, for $d = 6$ one needs only 45 points. The fractional factorial design is discussed in detail by Sanchez and Sanchez



(a) Grid based integration. (b) Monte Carlo integration (c) Central composite design

Figure 2.2: Illustration of the grid based, the Monte Carlo and the central composite design integration over the logarithm of the hyperparameters. The contour shows the posterior density $q(\log(\vartheta)|\mathcal{D})$ and the integration points are marked with dots. The left figure shows also the vectors $\mathbf{z}$ along which the points are searched in the grid integration and central composite desing. The integration is conducted over $q(\log(\vartheta)|\mathcal{D})$ rather than $q(\vartheta|\mathcal{D})$ since the former is closer to Gaussian.

(2005). The CCD integration can be summarized with the equation (2.24) where the evaluation points and weights are evaluated as follows.

The design points are searched after transforming ϑ into $\mathbf{z}$-space, which is assumed to be a standard Gaussian variable. The integration weights can then be determined from the statistics of a standard Gaussian variable $E[\mathbf{z}^{\mathrm{T}}\mathbf{z}] = d$, $E[\mathbf{z}] = \mathbf{0}$ and $E[1] = 1$. This results in integration weights

$$\Delta = \left[(n_p - 1) \exp\left(-\frac{df_0^2}{2} \right) \left(f_0^2 - 1 \right) \right]^{-1} \tag{2.27}$$

for the points on the sphere and $\Delta_0 = 1$ for the central point (see appendix Vanhatalo et al., 2010, for a more detailed derivation). The CCD integration speeds up the computations considerably compared to the grid search or Monte Carlo integration. The accuracy is between the empirical Bayes estimate and the full integration with grid search or Monte Carlo. Rue et al. (2009) and Martino (2007) report good results with this integration scheme, and it worked well also in our experiments. For more detailed treatment of the method see (Martino, 2007).

Chapter 3

Getting started with GPstuff: regression and classification

In this chapter we will discuss the use of GPstuff package with two classical examples, regression and classification. The regression task has Gaussian observation model and forms an important special case of the GP models since it is the only model where we are able to marginalize over the latent variables analytically. Thus, it serves as a good starting point for illustrating the use of the software package. The classification problem is a usuall text book example of tasks with non-Gaussian observation model where we have to utilize the approximate methods discussed in the previous chapter.

This chapter serves also as a general introduction to the GPstuff software package. In the next few sections, we will introduce and discuss many of the functionalities of the package that will be present in more advanced models as well.

3.1 Gaussian process regression: demo_regression1

The demonstration program `demo_regression1` considers the traditional regression problem with i.i.d observations with Gaussian noise, $y_i = f(\mathbf{x}_i) + \epsilon_i$, where $f(\mathbf{x})$ is given the Gaussian process prior and $\epsilon_i \sim N(0, \sigma_n^2)$. This results in the overall model

$$\mathbf{y}\,|\sigma^2 \sim \mathrm{N}(\mathbf{f}, \sigma^2\mathbf{I}), \tag{3.1}$$

$$f(\mathbf{x})|\theta \sim \mathcal{GP}\left(0, k(\mathbf{x}, \mathbf{x}'\,|\theta)\right), \tag{3.2}$$

$$\theta, \sigma^2 \sim p(\theta)p(\sigma^2). \tag{3.3}$$

Here we will show how to construct the model with squared exponential covariance function and how to conduct the inference.

3.1.1 Constructing the model

In the toolbox, the model construction requires three steps:

- create structures that define covariance function and noise function
- create structures that define priors for the covariance function parameters and set them into the covariance function structure

```
gp =
        type: 'FULL'
     likelih: 'regr'
          cf: {[1x1 struct]}
       noise: {[1x1 struct]}
infer_params: 'covariance+...'
jitterSigma2: 1.0000e-08
           p: []
```

```
gpcf1 =
        type: 'gpcf_sexp'
 lengthScale: [1.1000 1.2000]
  magnSigma2: 0.0400
           p: [1x1 struct]
      fh_pak: @gpcf_sexp/gpcf_sexp_pak
    fh_unpak: @gpcf_sexp/gpcf_sexp_unpak
        fh_e: @gpcf_sexp/gpcf_sexp_e
   fh_ghyper: @gpcf_sexp/gpcf_sexp_ghyper
   fh_ginput: @gpcf_sexp/gpcf_sexp_ginput
      fh_cov: @gpcf_sexp/gpcf_sexp_cov
    fh_trcov: @gpcf_sexp/gpcf_sexp_trcov
    fh_trvar: @gpcf_sexp/gpcf_sexp_trvar
fh_recappend: @gpcf_sexp/gpcf_sexp_recappend
```

Figure 3.1: The GP and covariance function structure in regression problem.

- create a Gaussian process structure where all the above are stored

These three steps are done with the following lines of code:

```
gpcf1 = gpcf_sexp('init', 'lengthScale', [1.1 1.2], 'magnSigma2', 0.2^2)
gpcf2 = gpcf_noise('init', 'noiseSigma2', 0.2^2);

pl = prior_t('init');
pm = prior_t('init', 's2', 0.3);
gpcf1 = gpcf_sexp('set', gpcf1, 'lengthScale_prior', pl, 'magnSigma2_prior', pm);
gpcf2 = gpcf_noise('set', gpcf2, 'noiseSigma2_prior', pm);

gp = gp_init('init', 'FULL', 'regr', {gpcf1}, {gpcf2}, 'jitterSigma2', 0.0001.^2);
```

Here functions `gpcf_sexp` and `gpcf_noise` initialize the covariance function and its parameter values. They return structures `gpcf1` and `gpcf2` that contain all the information needed in the evaluations (function handles, parameter values etc.). The next four lines create the prior structures for the parameters of the covariance and noise function, which are set into the covariance and noise function structures. The last line creates the Gaussian process structure. The prior used here is the Student-t distribution which works as a weakly informative prior. The initialization of the covariance function structure sets a uniform prior for the parameters by default. Thus, if you are satisfied with the uniform prior you do not need to run the four middle lines. The uniform prior would work well here as well, but we wanted to use the Student-t distribution to demonstrate how priors are set explicitly.

In the GPstuff toolbox, the Gaussian process structure is the fundamental unit, around which all the other blocks of the model are collected. This is illustrated in Figure 3.1. You can examine the model by going throught the `gp` structure where: `type` defines the type of the model (`FULL` means that it is not a sparse approximation, which are discussed in the chapter 4); `likelih` defines the likelihood, which is now just a string `regr`, that stands for regression model (in case of a non-Gaussian likelihood we would have here a likelihood structure, see section 3.2); `cf` and `noise` fields contain the covariance function and noise function structures; `infer_params` defines which parameters are inferred (covariance, likelihood etc.) and will be discussed in detail in the chapter 4; and `jitterSigma2` is a small constant which is added in the diagonal of the covariance matrix to make it numerically more stable. If there were more than one covariance or noise function, they would be handled additively. See section 6.2 for details. The covariance function structure is similar to the GP structure. The first four fields tell the basic information of the covariance function, such as the name and

the parameter values. After them it has a field p, which stands for prior, where the prior structures are stored. The rest of the fields in the covariance function structure are function handles to functions specific to that covariance function.

Now our GP structure is ready. We can, for example, evaluate basic summaries such as covariance matrices, make predictions with the present parameter values etc. For example the covariance matrix can be evaluated as follows:

```
>> example_x = [-1 -1 ; 0 0 ; 1 1];
>> [K, C] = gp_trcov(gp, example_x)
K =
    0.0400    0.0054    0.0000
    0.0054    0.0400    0.0054
    0.0000    0.0054    0.0400
C =
    0.0800    0.0054    0.0000
    0.0054    0.0800    0.0054
    0.0000    0.0054    0.0800
```

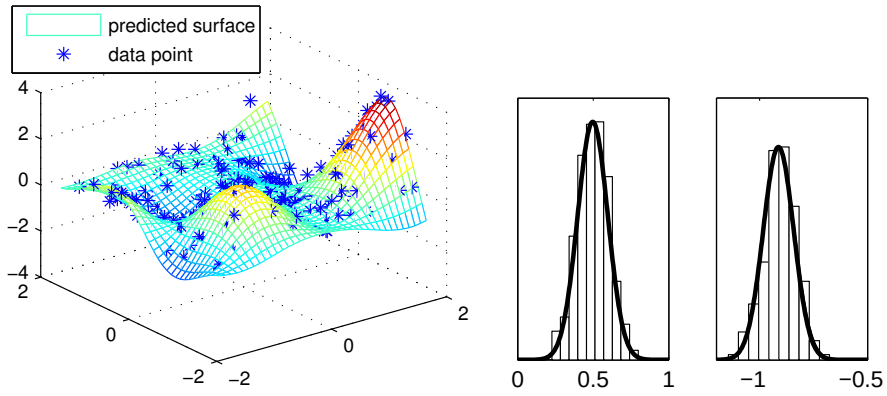
Here K is $\mathbf{K}_{\mathrm{f,f}}$ and C is $\mathbf{K}_{\mathrm{f,f}} + \sigma^2_{\mathrm{noise}}\mathbf{I}$.

3.1.2 MAP estimate for the hyperparameters

We can use what ever gradient based optimization method to find the posterior mode of the hyperparameters. For example, Matlab's optimization toolbox has fminunc routine which can choose from several different optimization algorithms. GPstuff toolbox has the scaled conjugate gradient algorithm implemented in the function scg2. Common to most of the ready available optimization methods is that user has to provide them the starting point as a vector, function handles to a function to minimize and to its gradient. For this reason GPstuff package contains gp_pak, gp_e and gp_g functions, whose usage is demonstrated below:

```
>> w = gp_pak(gp)
w =
   -3.2189    0.0953    0.1823   -3.2189
>> [e, edata, eprior] = gp_e(w, gp, x, y)
e =
   51.1294
edata =
   41.6645
eprior =
    9.4648
>> g = gp_g(w, gp, x, y)
g =
   -0.0236   -0.0307   -0.0339    0.0061
```

The function gp_pak packs all the hyperparameter values from all the covariance function structures into one vector. The reason why the values above do not mach with the values set into the covariance funtion is that the inference for the covariance functions parameters is conducted in a log space, $w = \log\theta$. Thus, the packing function of a covariance function structure, which is called by gp_pak, takes the logarithm of the hyperparameters. The elements of the packed vector can be set back into the covariance function with un_pak function. edata stands for *energy data*, which is the negative log marginal likelihood, $-\log p(\mathbf{y}\,|\theta)$. eprior contains the negative log prior and the log Jacobian resulting from the log transformation $-\log p(\theta) - \log|J|$. The first output argument from gp_e summarizes these two, e = edata + eprior and thus it is the negative log posterior cost function. The function gp_g returns the gradients of the energy function gp_e with respect to the elements in w. The gradient function returns also the gdata and gprior, but there is no direct relation between edata



(a) The prediction surface and training data of demo_regression1

(b) The posterior predictive distribution $p(f_i|\mathcal{D})$ for two latent variable marginals. Histogram shows the MCMC solution and the grid integration solution is drawn with a line.

Figure 3.2: The predictive surface, training data, and the marginal posterior for two latent variables in demo_regression1.

and `eprior`. The data gradient is actully $-\theta\nabla_\theta \log(p(\mathbf{y}\,|\theta))$ and the prior gradient is $-\theta\nabla_\theta \log(p(\theta)) - 1$. The multiplication by θ and the subtraction of 1 result from the Jacobin that is needed in the variable transformation. See appendix A for more detailed explanation of the log transformation.

With the above functions we can find the mode of the posterior as follows:

```
w=gp_pak(gp);
fe=str2fun('gp_e');
fg=str2fun('gp_g');

% set the options for scg2
opt = scg2_opt;
opt.tolfun = 1e-3;
opt.tolx = 1e-3;
opt.display = 1;

% do the optimization
w=scg2(fe, w, opt, fg, gp, x, y);

% Set the optimized hyperparameter values back to the gp structure
gp=gp_unpak(gp,w);
```

Now, the parameters of the covariance function and noise function in the `gp` structure are set to their MAP estimate, which are $\sigma^2_{\text{se}} = 1.7$, $l_{\text{se}} = [1.11.2]^{\text{T}}$ and $\sigma^2_{\text{noise}} = 0.04$.

To make predictions for new locations we can use the `gp_pred` function, which returns the posterior mean and variance for each $f(\tilde{\mathbf{x}})$ (see equation (2.10)). This is illustrated with the below lines of code where we first create a regular grid where we want to make predictions and then find the posterior mean and variance. The posterior mean μ_p together with the training data points is shown in the figure 3.2.

```
[p1,p2]=meshgrid(-1.8:0.1:1.8,-1.8:0.1:1.8);
p=[p1(:) p2(:)];
[Ef_full, Varf_full] = gp_pred(gp, x, y, p);
```

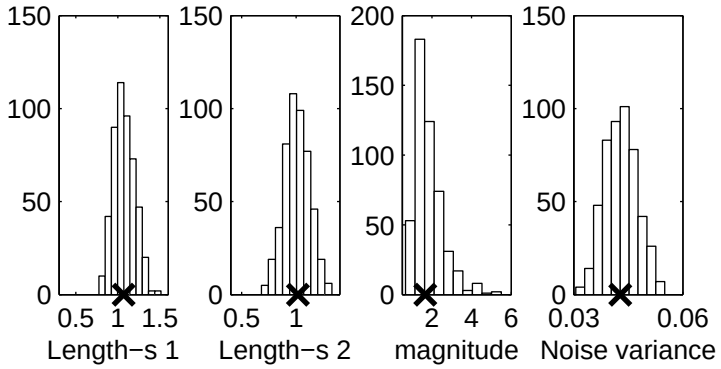


Figure 3.3: The posterior distribution of the hyperparameters together with MAP estimate (X). The results are from `demo_regression1`.

3.1.3 Marginalization over hyperparameters with grid integration

To integrate over the hyperparameters we can use which ever method described in the section 2.2. Here, we will demonstrate the grid integration and after that show how MCMC is performed. The CCD and importance sampling methods are demonstrated in the sections **??** and **??**. The integration is performed with the following line:

```
[gp_array, P_TH, th, Ef_ia, Varf_ia, fx_ia, x_ia] = gp_ia(gp, x, y, p, 'int_method', 'grid');
```

What happens inside `gp_ia` is the following. First the hyperparameters are optimized to their posterior mode, the Hessian $\mathbf{P}^{-1}$ is evaluated with finite differences, and the grid search is conducted starting for the mode. Since we give the prediction inputs `p` (representing $\tilde{\mathbf{X}}$) the integration method returns also the predictive distributions for those locations. Otherwise `gp_ia` would return only the first three arguments, which contain the array of GP's (`gp_array`) for hyperparameter values `th` ($[\vartheta_i]_{i=1}^M$) with weights `P_TH` ($[p(\vartheta_i|\mathcal{D})\Delta_i]_{i=1}^M$). Since we use the grid method the weights are proportional to the marginal posterior and $\Delta_i \equiv 1 \forall i$ (see section 2.2.2). `Ef_ia` and `Varf_ia` contain the predictive mean and variance at the prediction locations `p`. The last two output arguments can be used to plot the predictive distribution $p(\tilde{f}_i|\mathcal{D})$ at input location $\tilde{\mathbf{x}}_i$ (which is the i'th row of `p`) as demonstrated in the figure 3.2. `x_ia` contains a regular grid of values $\tilde{f}_i$ and `fx_ia` contains $p(\tilde{f}_i|\mathcal{D})$ at those values.

3.1.4 Marginalization over hyperparameters with MCMC

The main function to conduct Markov chain sampling is `gp_mc`, which loops through all the possible samplers in turn and saves the sampled parameters into a record structure. In later sections, we will discuss models where also latent variables are sampled but now we concentrate on the covariance function parameters, which are sampled as follows:

```
hmc_opt = hmc2_opt;
hmc_opt.steps=4;
hmc_opt.stepadj=0.05;
hmc_opt.persistence=0;
hmc_opt.decay=0.6;
hmc_opt.nsamples=1;
```

```
hmc2('state', sum(100*clock));

[rfull,g,opt] = gp_mc(gp, x, y, 'nsamples', 400, 'repeat', 5, 'hmc_opt', hmc_opt);
rfull = thin(rfull, 10, 2);
[Ef_mc, Varf_mc] = mc_pred(rfull, x, y, p);
```

The `gp_mc` function makes `nsamples` iterations (400 here) and stores every `repeat`'th (5) sample. At each iteration it runs the actuall samplers. For example, giving the option `'hmc_opt'` tells that `gp_mc` should run the hybrid Monte Carlo sampler with sampling options stored in the structure `hmc_opt`. The default sampling options for HMC are set by `hmc2_opt` function, after which some of them are modified for better mixing. The function `thin` removes the burn-in from the sample chain (here 50) and thins the chain with user defined parameter (2 here). This way we can decrease the autocorralation between the remaining samples. The Markov chain should always be examined closely to ensure that the sample chain has actually converged (Gelman et al., 2004; Robert and Casella, 2004). Diagnostic tools for Markov chains can be found from the diag folder in GPstuff package. Its contents can be examined, for example, with `help diag` command.

The function `mc_pred` returns the conditional predictive mean and variance for each sampled hyperparameter value. These are $\mathrm{E}_{p(\mathbf{f}|\mathbf{X},\mathcal{D},\vartheta^{(s)})}[\tilde{\mathbf{f}}], s = 1, ..., M$ and $\mathrm{Var}_{p(\mathbf{f}|\mathbf{X},\mathcal{D},\vartheta^{(s)})}[\tilde{\mathbf{f}}], s = 1, ..., M$. Note, that the values returned by `Ef_mc` are not samples from $p(\mathbf{f}\,|\mathcal{D})$ since we sampled only the hyperparameters and the latent variables were marginalized out analytically. Thus, in order to obtain the marginal mean and variance we have to marginalize over the hyperparameters using and (see, Gelman et al., 2004)

$$\mathrm{E}[\tilde{\mathbf{f}}|\mathcal{D}] = \mathrm{E}_{p(\theta|\mathcal{D})}[\mathrm{E}[\tilde{\mathbf{f}}|\theta,\mathcal{D}]] \tag{3.4}$$

$$\mathrm{Var}_{\tilde{\mathbf{f}}|\mathcal{D}}[\tilde{\mathbf{f}}] = \mathrm{E}_{\theta|\mathcal{D}}\left[\mathrm{Var}_{\tilde{\mathbf{f}}|\theta,\mathcal{D}}[\tilde{\mathbf{f}}]\right] + \mathrm{Var}_{\theta|\mathcal{D}}\left[\mathrm{E}_{\tilde{\mathbf{f}}|\theta,\mathcal{D}}[\tilde{\mathbf{f}}]\right], \tag{3.5}$$

In Matlab notation these calculations are done as follows:

```
mean(Ef_mc, 2)
mean(Varf_mc,2) + var(Ef_mc,0,2)
```

In the section 3.2, we discuss a model where latent variables are sampled alongside the hyperparameters and where `mc_pred` returns samples from $p(\mathbf{f}\,|\mathcal{D})$.

We could also use the HMC sampler alone by calling `hmc2` but then we would get only a matrix of hyperparameter values. The reason we use the `gp_mc` is that it stores all the samples in a structure that can be passed to `thin` and `mc_pred` function. Above we used the posterior mode as a starting point for the sampler. In order to conduct a detailed analysis with MCMC methods we should use several starting points and monitor the convergence carefully. However, here we just presented the basic procedure for sampling.

3.2 Gaussian process classification: demo_classific

We will now consider a binary Gaussian process classification. In the regression problem considered earlier the target of the inference was to learn a function that maps the inputs into real valued function. In binary classification, the aim is to learn a rule that assignes inputs into two separate classes. The Gaussian process serves as a prior for latent function that is transformd through a sigmoid function to give the propability for

a specific class at that input location. A detailed discussion on the GP classification is given by Rasmussen and Williams (2006).

We will consider a classification problem with binary observations, $y_i \in \{-1, 1\}, i = 1, ..., n$, appointed to inputs $\mathbf{X} = \{\mathbf{x}\}_{i=1}^{n}$. The observations are considered to be drawn from a Bernoulli distribution with a success probability $p(y_i = 1|\mathbf{x}_i)$. The probability is related to a latent function $f(\mathbf{x}) : \Re^d \rightarrow \Re$ that is mapped to a unit interval by a sigmoid transformation. Possible transformations in the GPstuff package are the probit and logit, which lead to observation models

$$p_{\text{probit}}(y_i|f(\mathbf{x}_i)) = \Phi(y_i f(\mathbf{x}_i)) = \int_{-\infty}^{y_i f(\mathbf{x}_i)} N(z|0, 1)\, \mathrm{d}\, z \tag{3.6}$$

$$p_{\text{logit}}(y_i|f(\mathbf{x}_i)) = \frac{1}{1 + \exp(-y_i f(\mathbf{x}_i))}. \tag{3.7}$$

The latent function is given a Gaussian process prior. Since the likelihood is not Gaussian we need to use the approximate methods, discussed in the section 2.1, for the inference. We will conduct the inference with Laplace approximation, EP and full MCMC. With the two analytic approximations we optimize the hyperparameters to their approximate MAP estimate and use that point estimate for prediction. In MCMC approach we sample both the hyperparameters and the latent variables. For a detailed discussion on the differences between MCMC, Laplace and EP approaches in classification setting see (Kuss and Rasmussen, 2005; Nickisch and Rasmussen, 2008).

3.2.1 Constructing the model

The model construction for the classification follows closely the steps presented in the regression model case. The model is constructed as follows:

```
gpcf1 = gpcf_sexp('init', 'lengthScale', [0.9 0.9], 'magnSigma2', 10);

pl = prior_logunif('init');
gpcf1 = gpcf_sexp('set', gpcf1, 'lengthScale_prior', pl,'magnSigma2_prior', pl); %

likelih = likelih_probit('init');
%likelih = likelih_logit('init');

gp = gp_init('init', 'FULL', likelih, {gpcf1}, [],'jitterSigma2', 0.01);
```

The first three lines initialize the covariance function and set uniform prior for the logarithm of its parameters. Fourth line initializes the likelihood structure and the last line initializes the GP stucture. The differences between GP initialization here and in the regression case are that now we give the likelihood structure as the fourth input for the `gp_init` whereas in regression case the argument is a string `'regr'`. Also here we do not give noise covariance function and thus the sixth argument for `gp_init` is just an empty matrix.

Here we will use the probit likelihood but the model construction and the inference with logit likelihood are done exactly the same way as for the probit likelihood. We only need to replace the construction of the probit likelihood structure with the construction of logit likelihood structure, which is shown in the above code with a commented line `likelih = likelih_logit('init');`. The two likelihoods are compared in `demo_modelassesment2`.

3.2.2 Inference with Laplace approximation

The MAP estimate for the hyperparameters is searched with the following lines:

```
gp = gp_init('set', gp, 'latent_method', {'Laplace', x, y});

fe=str2fun('gpla_e');
fg=str2fun('gpla_g');
n=length(y);
opt = scg2_opt;
opt.tolfun = 1e-3;
opt.tolx = 1e-3;
opt.display = 1;
opt.maxiter = 20;

w=gp_pak(gp);
w=scg2(fe, w, opt, fg, gp, x, y);
gp=gp_unpak(gp,w);
```

The first line defines which inference method is used for the latent variables. It initializes the Laplace algorithm and sets needed fields into the GP structure. The next two lines define function handles to the the energy function and its gradient, which are needed for the scaled conjugate gradient algorithm (`scg2`). Notice that here the energy function is `gpla_e` and its gradient is `gpla_g` which work just as `gp_e` and `gp_g` discussed in the section 3.1.2. The difference is that in the regression setting the returned energy is exact whereas here the energy is an approximation since the marginal likelihood is an approximation given in the equation (2.20). The next lines set the options for the optimization algorithm, pack the hyperparameters into wector, conducts the optimization and unpacks the optimized parameters into the GP structure.

The Laplace method has to be initialized since the algorithm uses Matlabs nested functions to store the results of the algorithm. Every time `gpla_e` is run the Laplace approximation for the conditional posterior, $q(\mathbf{f}\,|\mathcal{D},\theta)$, and the marginal likelihood $q(\mathcal{D}|\theta)$, are stored. Only one result is stored at a time and the stored result is changed every time hyperparameters are modified. If `gpla_e` is called multiple times with exactly the same hyperparameter values θ the function returns the approximations evaluated at the first time. This kind of functionality is implemented for speeding up the computations. The Laplace approximation is needed in the gradient evaluations and prediction and `gpla_e` is called from both `gpla_g` and `la_pred` functions. If the Laplace approximation was run from scracth every time `gpla_g` is called the optimization would take twice the time compared to current implementation. The next few lines of code show a snapshot of the function `gpla_e`:

```
function [e, edata, eprior, f, L, a, La2, p] = gpla_e(w, gp, x, y, varargin)

    if ischar(w) && strcmp(w, 'init')
        w0 = rand(size(gp_pak(gp)));
        e0=[];
        ...
        p0 = [];

        laplace_algorithm(gp_pak(gp), gp, x, y, z);

        gp.fh_e = @laplace_algorithm;
        e = gp;
    else
        [e, edata, eprior, f, L, a, La2, p] = feval(gp.fh_e, w, gp, x, y, z);
    end

    % Begin the nested function
    function [e, edata, eprior, f, L, a, La2, p] = laplace_algorithm(w, gp, x, y, z)

        if abs(w-w0) < 1e-8
```

```
            % The covariance function parameters haven't changed so just
            % return the Energy and the site parameters that are saved
            e = e0;
            ...
            p = p0;
        else
            % We end up here if the hyperparameters have changed since
            % the last call for gpla_e. In this case we need to
            % re-evaluate the Laplace approximation, which is done
            % below
            ...

            % Store the parameters
            w0 = w;
            e0 = e;
            ...
            p0 = p;
        end
    end  % End the nested function

end % End the main function
```

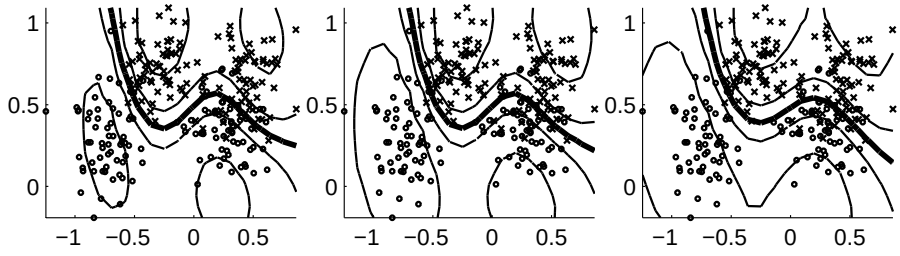
gpla_e is called with option 'init' from gp_init when the Laplace approximation is initialized as discussed above. In the initialization, first essential parameters, w0, e0,...p0, are defined. These are parameters that are needed when evaluating the Laplace approximations and which will be stored. After that the Laplace algorithm is run (laplace_algorithm(...)) during which the earlier defined parameters are filled with values representing the Laplace approximations with the current hyperparameter values (e0=e...). Finally a function handle to the Laplace algorithm is created and set into the GP structure. Whenever we call gpla_e after this initilization we end up in the Laplace algorithm directly through the function handle in the GP structure. There the function first checks our hyperparameter values (if abs(w-w0) < 1e-8) and if they have not changed since the last call for gpla_e the function will immediately return the values stored in the parameters e0, edata0, ... If the hyperparameters have changed the Laplace approximation for $p(\mathbf{f}\,|\mathcal{D},\theta)$, and $p(\mathcal{D}|\theta)$ is re-evaluated.

The key structure in nested functions is to end the main fucntion and the subfunctions with end statement and position the subfunctions inside the main function. Then all the subfunctions will see the parameter space of the main function. Whenever we construct a function handle to the subfunction the parameters initilized in the main function will be stored in the memory space of that function handle. This memory space will persist as long as the function handle and the parameters in there can be modified. This property is utilized in GPstuff in many places where we want to store some results for future use. For a detailed discussion on Matlabs nested functions see the Matlab documentation.

To return to our demo we will consider the prediction next. We can use the GP with optimized hyperparameters to evaluate the posterior predictive statistics. The function la_pred returns the mean and variance for the latent variables and new observations together with the predictive probability for test observation:

```
[Ef_la, Varf_la, Ey_la, Vary_la, p1_la] = la_pred(gp, x, y, xt, 'yt', ones(size(xt,1),1) );
```

The first four input arguments are the same as for the gp_pred function (see section 3.1.2), the fifth and sixth arguments are a parameter-value pair. Parameter yt tells that we give test outputs related to xt as optional inputs for la_pred. Here we want to evaluate the propabability to observe class 1 and thus we give a vector of ones as test observations. The probability densities for test outputs are returned in the fifth



(a) Laplace approximation. (b) Expectation propagation. (c) Markov chain Monte Carlo.

Figure 3.4: The class probability contours for Laplace approximation, EP, and MCMC solution. The strongest line in the middle is the 50% probability line (the decision boundary). The thinner lines are 2.5%, 25%, 75% and 97.5% probability contours. It can be seen that the decision boundary is approximated similarly with all the methods but there is bigger difference in the other contours.

output argument `p1_la`. See section **??** for discussion how these propabibilities can be used for cross validation.

The training data together with predicted class probability contours is visualized in the figure 3.4. The figure 3.5 shows the marginal predictive posterior for two latent variables.

3.2.3 Inference with expectation propagation

The EP framework works just as the laplace approximation. We just need to replace the functions specific to the Laplace approximation with ones related to EP. The lines needed for EP inference are the following:

```
gp = gp_init('set', gp, 'latent_method', {'EP', x, y});

w = gp_pak(gp);
fe=str2fun('gpep_e');
fg=str2fun('gpep_g');
n=length(y);
opt = scg2_opt;
opt.tolfun = 1e-3;
opt.tolx = 1e-3;
opt.display = 1;

w=gp_pak(gp);
w=scg2(fe, w, opt, fg, gp, x, y);
gp=gp_unpak(gp,w);

[Ef_ep, Varf_ep, Ey_ep, Vary_ep, p1_ep] = ep_pred(gp, x, y, xt, 'yt', ones(size(xt,1),1) );
```

The EP algorithm is implemented using the nested functions in a similar manner as Laplace approximation, for which reason the algorithm has to be initilized first. The EP solution for the predicted class probabilities is visualized in the figure 3.4, and the figure 3.5 shows the marginal predictive posterior for two latent variables.

3.2.4 Inference with MCMC

The MCMC solution with non-Gaussian likelihood is found similarly to the regression model discussed earlier. The major difference is that now we need to sample also the

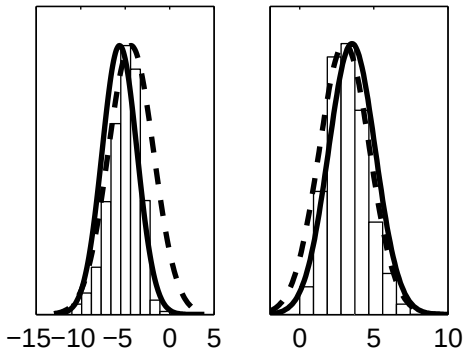


Figure 3.5: The marginal posterior for two latent variables in demo_classific1. Histogram is the MCMC solution, dashed line is the Laplace approximation and full line the EP estimate. The left figure is for latent variable at location $[-1.0, 0.2]$ and the right figure at $[0.9, 0.9]$.

latent variables appointed to the training inputs. The latent variables and the function handle to their sampler have to be stored into the GP structure. This is done by initilizing the field `latent_method` in the GP structure to be MCMC. This initilization is done with the following line:

```
gp = gp_init('set', gp, 'latent_method', {'MCMC', zeros(size(y))', @scaled_mh});
```

Here the last argument `@scaled_mh` defines the sampler which is used for sampling $p(\mathbf{f}\,|\theta)$. The scaled Metropolis Hastings algorithm is discussed by Neal (1998). Other sampler provided by the GPstuff is scaled Hybrid Monte Carlo, which is discussed in the section **??**.

Now that we have initialized the MCMC sampler for the latent variables we can proceed with the actual sampling. This is performed with the following lines:

```
% Set the parameters for MCMC...
hmc_opt.steps=10;
hmc_opt.stepadj=0.1;
hmc_opt.nsamples=1;
latent_opt.display=0;
latent_opt.repeat = 20;
latent_opt.sample_latent_scale = 0.5;
hmc2('state', sum(100*clock))

[r,g,opt]=gp_mc(gp, x, y, 'hmc_opt', hmc_opt, 'latent_opt', latent_opt, 'nsamples', 1, 'repeat', 15);

% re-set some of the sampling options
opt.nsamples=400;
opt.repeat=1;
opt.hmc_opt.steps=4;
opt.hmc_opt.stepadj=0.02;
opt.latent_opt.repeat = 5;
hmc2('state', sum(100*clock));

% Sample
[rgp,g,opt]=gp_mc(gp, x, y, opt, 'record', r);

% Make predictions
[Ef_mc, Varf_mc, Ey_mc, Vary_mc, p1_mc] = mc_pred(rgp, x, y, xt, 'yt', ones(size(xt,1),1) );
p1_mc = mean(p1_mc,2);
```

The HMC options are set into the `hmc_opt` structure in a similar manner as in the regression example. Since we are sampling also the latent variables we need to give op-

tions for their sampler as well. These options are set into the `latent_opt` structure. The options specific to `gp_mc` are given with parameter-value pairs `'nsamples', 1, 'repeat', 15`. The above lines demonstrate also how the sampling can be continued from an old sample chain. The first call for `gp_mc` returns a record structure with only one sample. This record is then given as an optional parameter for `gp_mc` in the second call for it. The sampling is continued from the previously sampled parameter values. The sampling options are also modified between the two successive sampling phases. The old options are returned by `gp_mc` in a structure `opt`, and some of the fields in this structure are modified. The modified options are then given to `gp_mc`. The line `hmc2('state', sum(100*clock));` re-sets the state of the random number generators in the HMC sampler. The last two lines evaluate the predictive statistics similarly to the EP and Laplace approximations. However, now the statistics are matrices whose columns contain the result for one MCMC sample each. The `gp_mc` function handles the sampling so that it first samples the latent variables from $p(\mathbf{f}\,|\dot{\theta})$ using the scaled Metropolis Hastings after which it samples the hyperparameters from $p(\theta|\dot{\mathbf{f}})$. This is repeated until `opt.nsamples` samples are drawn.

The MCMC solution for the predicted class probabilities is visualized in the figure 3.4, and the figure 3.5 shows the marginal predictive posterior for two latent variables. From the figures it can be seen that there are little differences between the three different approximations. Here MCMC is the most accurate, then comes EP and Laplace is the worst. However, the inference times line up in the opposite order. The difference between the approximations is not always this big and in the section **??** we give an example of model where Laplace approximation and EP work as well as MCMC.

Chapter 4

Sparse Gaussian processes

The challenges with using Gaussian process models are the fast increasing computationa time and memory requirements. The evaluation of the inverse and determinant of the covariance matrix in the log marginal likelihood (or its approximation) and its gradient scale as $O(n^3)$ in time, which restricts the implementation of the GP models to moderate size data sets. For this reason there are number of sparse Gaussian processes introduced in the literature. Here, we will discuss sparse GPs that utilize compactly supported covariance functions and fully (FIC) and partially (PIC) independent sparse approximations.

4.1 Compactly supported covariance functions

A compactly supported covariance function is a function that gives zero correlation between data points whose distance exceeds a certain threshold leading to a sparse covariance matrix. The challenge in constructing CS covariance functions is to guarantee their positive definiteness. A full support covariance function can not be cut arbitrarily to obtain a compact support, since the resulting function would not, in general, be positive definite. Sansò and Schuh (1987) provide one of the early implementations of spatial prediction with CS covariance functions. Their functions are build by self-convoluting known finite support symmetric kernels (such as a linear spline). These are, however, special functions for one or two dimensions. Later Wu (1995) introduced radial basis functions with compact support and a generic procedure to construct them. Wendland (1995) developed them further and later, for example, Gaspari and Cohn (1999) Gneiting (1999, 2002), and Buhmann (2001) worked more on the subject.

The CS functions studied in this work are Wendland's piecewise polynomials $k_{pp,q}$ (Wendland, 2005), such as:

$$k_{\text{pp2}} = \frac{\sigma^2_{\text{pp}}}{3}(1-r)_+^{j+2}\left((j^2+4j+3)r^2+(3j+6)r+3\right), \tag{4.1}$$

where $j = \lfloor d/2 \rfloor + 3$ and $r^2 = \sum_{k=1}^{d}(x_{i,k} - x_{j,k})^2/l_k^2$. These functions correspond to processes that are q times mean square differentiable and are positive definite up to input dimension d. Thus, the degree of the polynomial has to be increased alongside the input dimension. The dependence of CS covariance functions to the input dimension is very fundamental. There are no radial compact support functions that are positive

definite on every $\Re^d$ but they are always restricted to a finite number of dimensions (see e.g. Wendland, 1995, theorem 9.2).

The key idea with using CS covariance functions is that, roughly speaking, one uses only the nonzero elements of the covariance matrix in the calculations. This may speed up the calculations substantially since in some situations only a fraction of the elements of the covariance matrix are non-zero. In practice, efficient sparse matrix routines are needed (Davis, 2006), which are nowadays a standard utility in many statistical computing packages, such as Matlab or R, or available as an additional package for them. The CS covariance functions have been rather widely studied in the geostatistics applications. The early works concentrated on their theoretical properties and aimed to approximate the known global support covariance functions (Gneiting, 2002; Furrer et al., 2006; Moreaux, 2008). There the computational speed-up is obtained using efficient linear solvers for the prediction equation $\tilde{\mathbf{f}} = \mathbf{K}_{\tilde{\mathrm{f}},\mathrm{f}}(\mathbf{K}_{\mathrm{f},\mathrm{f}} + \sigma^2)^{-1}\,\mathbf{y}$. The hyperparameters are fitted to either the empirical covariance or the global support covariance. Kaufman et al. (2008) study the maximum likelihood estimates for tapered covariance functions (those are products of global support and CS covariance functions). There, the magnitude can be solved analytically and the length-scale is optimized using a line search in one dimension. The benefits from a sparse covariance matrix have been immediate since the problems collapse into solving sparse linear systems. However, utilizing the gradient of the log posterior of the hyperparameters needs some extra sparse matrix tools.

The problematic part is the trace in the derivative of the log marginal likelihood

$$\begin{aligned}\frac{\partial}{\partial\theta}\log p(\mathbf{y}|\mathbf{X},\theta) =& \frac{1}{2}\mathbf{y}^{\mathrm{T}}(\mathbf{K}_{\mathrm{f},\mathrm{f}} + \sigma^2\mathbf{I})^{-1}\frac{\partial(\mathbf{K}_{\mathrm{f},\mathrm{f}} + \sigma^2\mathbf{I})}{\partial\theta}(\mathbf{K}_{\mathrm{f},\mathrm{f}} + \sigma^2\mathbf{I})^{-1}\mathbf{y} \\ & - \frac{1}{2}\mathrm{tr}\left((\mathbf{K}_{\mathrm{f},\mathrm{f}} + \sigma^2\mathbf{I})^{-1}\frac{\partial(\mathbf{K}_{\mathrm{f},\mathrm{f}} + \sigma^2\mathbf{I})}{\partial\theta}\right). \qquad (4.2)\end{aligned}$$

The trace would require us to form the full covariance matrix if evaluated naively. Luckily, Takahashi et al. (1973) introduced an algorithm whereby we can evaluate a sparsified version of the inverse of a sparse matrix. This can be utilized in the gradient evaluations as described by Vanhatalo and Vehtari (2008). The same problem was considered by Storkey (1999) who used the covariance matrices of Toeplitz form, which are fast to handle due their banded structure. However, constructing Toeplitz covariance matrices is not possible in two or higher dimensions without approximations. Also EP algorithm requires special considerations with CS covariance functions. The posterior covariance in EP (2.16) does not remain sparse, for which reason it has to be expressed implicitly during the updates. These issues are discussed in (Vanhatalo et al., 2010; Vanhatalo and Vehtari, 2010).

4.2 FIC and PIC sparse approximations

Snelson and Ghahramani (2006) proposed a sparse pseudo-input Gaussian process (SPGP), which Quiñonero-Candela and Rasmussen (2005) named later fully independent training conditional (FITC). The original idea in SPGP was that the sparse approximation is used only in the training phase and predictions are conducted using the exact covariance matrix, where the word 'training' comes to the name. If the approximation is used also for the predictions, the word training should drop out leading to FIC. In this case, FIC can be seen as a non-stationary covariance function on its own (Snelson,

2007). The partially independent conditional (PIC) sparse approximation is an extension of FIC (Quiñonero-Candela and Rasmussen, 2005; Snelson and Ghahramani, 2007), and they are both treated here following Quiñonero-Candela and Rasmussen (2005).

The approximations are based on introducing an additional set of latent variables $\mathbf{u} = \{u_i\}_{i=1}^m$, called *inducing variables*. These correspond to a set of input locations $\mathbf{X}_u$, *inducing inputs*. The latent function prior is approximated as

$$p(\mathbf{f}\,|\mathbf{X}) \approx q(\mathbf{f}\,|\mathbf{X}, \mathbf{X}_u) = \int q(\mathbf{f}\,|\mathbf{X}, \mathbf{X}_u, \mathbf{u}) p(\mathbf{u}\,|\mathbf{X}_u) \mathrm{d}\mathbf{u}, \tag{4.3}$$

where $q(\mathbf{f}\,|\mathbf{X}, \mathbf{X}_u, \mathbf{u})$ is the inducing conditional. The above decomposition leads to the exact prior if the true conditional $\mathbf{f}\,|\,\mathbf{u} \sim \mathrm{N}(\mathbf{K}_{\mathrm{f,u}}\,\mathbf{K}_{\mathrm{u,u}}^{-1}\,\mathbf{u}, \mathbf{K}_{\mathrm{f,f}} - \mathbf{K}_{\mathrm{f,u}}\,\mathbf{K}_{\mathrm{u,u}}^{-1}\,\mathbf{K}_{\mathrm{u,f}})$ is used. However, in FIC framework the latent variables are assumed to be conditionally independent given $\mathbf{u}$, in which case the inducing conditional factorizes, $q(\mathbf{f}\,|\,\mathbf{u}) = \prod q_i(f_i|\,\mathbf{u})$. In PIC latent variables are set into blocks which are conditionally independent of each others, given $\mathbf{u}$, but the latent variables within a block have a multivariate normal distribution with original covariance. The approximate conditionals of FIC and PIC can be summarized as

$$q(\mathbf{f}\,|\mathbf{X}, \mathbf{X}_u, \mathbf{u}) = \mathrm{N}(\mathbf{f}\,|\,\mathbf{K}_{\mathrm{f,u}}\,\mathbf{K}_{\mathrm{u,u}}^{-1}\,\mathbf{u}, \mathrm{mask}\left(\mathbf{K}_{\mathrm{f,f}} - \mathbf{K}_{\mathrm{f,u}}\,\mathbf{K}_{\mathrm{u,u}}^{-1}\,\mathbf{K}_{\mathrm{u,f}}|\mathbf{M}\right)), \tag{4.4}$$

where the function $\mathbf{\Lambda} = \mathrm{mask}\,(\cdot|\mathbf{M})$, with matrix $\mathbf{M}$ of ones and zeros, returns a matrix $\mathbf{\Lambda}$ of size $\mathbf{M}$ and elements $\mathbf{\Lambda}_{ij} = [\cdot]_{ij}$ if $\mathbf{M}_{ij} = 1$ and $\mathbf{\Lambda}_{ij} = 0$ otherwise. An approximation with $\mathbf{M} = \mathbf{I}$ corresponds to FIC and an approximation where $\mathbf{M}$ is block diagonal corresponds to PIC. The inducing inputs are given a zero-mean Gaussian prior $\mathbf{u} \sim \mathrm{N}(\mathbf{0}, \mathbf{K}_{\mathrm{u,u}})$ so that the approximate prior over latent variables is

$$q(\mathbf{f}\,|\mathbf{X}, \mathbf{X}_u) = \mathrm{N}(\mathbf{f}\,|\mathbf{0}, \mathbf{Q}_{\mathrm{f,f}} + \mathbf{\Lambda}), \tag{4.5}$$

where the matrix $\mathbf{K}_{\mathrm{f,u}}\,\mathbf{K}_{\mathrm{u,u}}^{-1}\,\mathbf{K}_{\mathrm{u,f}}$ is of rank m and $\mathbf{\Lambda}$ is a rank n (block) diagonal matrix. The prior covariance can be seen as a non-stationary covariance function of its own where the inducing inputs $\mathbf{X}_u$ and the matrix $\mathbf{M}$ are free parameters similar to hyperparameters, which can be optimized alongside θ (Snelson and Ghahramani, 2006; Lawrence, 2007).

The computational savings are obtained by using the Woodbury-Sherman-Morrison lemma to invert the covariance matrix in (4.5) as

$$(\mathbf{Q}_{\mathrm{f,f}} + \mathbf{\Lambda})^{-1} = \mathbf{\Lambda}^{-1} - \mathbf{V}\mathbf{V}^{\mathrm{T}}, \tag{4.6}$$

where $\mathbf{V} = \mathbf{\Lambda}^{-1}\,\mathbf{K}_{\mathrm{f,u}}\,\mathrm{chol}[(\mathbf{K}_{\mathrm{u,u}} + \mathbf{K}_{\mathrm{u,f}}\,\mathbf{\Lambda}^{-1}\,\mathbf{K}_{\mathrm{f,u}})^{-1}]$ (e.g. Harville, 1997). There is a similar result also for the determinant. With FIC the computational time is dominated by the matrix multiplications, which need time $O(m^2 n)$. With PIC the cost depends also on the sizes of the blocks in $\mathbf{\Lambda}$. If the blocks were of equal size $b \times b$, the time for inversion of $\mathbf{\Lambda}$ would be $O(n/b \times b^3) = O(nb^2)$. With blocks at most the size of the number of inducing inputs, that is $b = m$, the computational cost in PIC and FIC are similar. Intuitively, PIC approaches FIC in the limit of a block size one and the exact GP in the limit of a block size n. A formal treatment of this is given by Snelson (2007).

4.3 Deterministic training conditional, subset of regressors and variational sparse approximation

The deterministic training conditional is based on the works by Csató and Opper (2002) and Seeger et al. (2003) and is earlier called Projected Latent Variables method (See Quiñonero-Candela and Rasmussen, 2005, for more details). The approximation can be constructed similarly as FIC and PIC by defining the inducing conditional, which in the case of DTC is

$$q(\mathbf{f}\,|\mathbf{X}, \mathbf{X}_u, \mathbf{u}) = \mathrm{N}(\mathbf{f}\,|\,\mathbf{K}_{\mathrm{f,u}}\,\mathbf{K}_{\mathrm{u,u}}^{-1}\,\mathbf{u}, 0) \tag{4.7}$$

This implies that the approximate prior over latent variables is

$$q(\mathbf{f}\,|\mathbf{X}, \mathbf{X}_u) = \mathrm{N}(\mathbf{f}\,|\mathbf{0}, \mathbf{Q}_{\mathrm{f,f}}). \tag{4.8}$$

The deterministic training conditional is not strictly speaking a proper Gaussian process since it uses different covariance function for the latent variables appointed to the training inputs and for the latent variables at the prediction sites, $\tilde{f}$. The prior covariance for $\tilde{f}$ is the true covariance $\mathbf{K}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}}$ instead of $\mathbf{Q}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}}$ (compare with the equation (4.8)). This does not affect the predictive mean since the cross covariance $\mathrm{Cov}[\mathbf{f}, \tilde{f}] = \mathbf{Q}_{\mathrm{f},\tilde{\mathrm{f}}}$, but it gives larger variance than with $\mathbf{Q}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}}$. An older version of DTC is the subset of regressors (SOR) sparse approximation which utilizes $\mathbf{Q}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}}$. The reason for replacing $\mathbf{Q}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}}$ with the exact covariance in DTC is that SOR resemples singular Gaussian distribution for which reason the predictive variance may be negative in some cases. DTC fixes this problem as discussed by Quiñonero-Candela and Rasmussen (2005). DTC and SOR are identical in other respects than in the variance evaluation. In spatial statistics, SOR has been used also by Banerjee et al. (2008) with a name Gaussian predictive process model. DTC alongside EP was proposed for spatial statistics by Cornford et al. (2005).

The approximate prior of the variational approximation by Titsias (2009) is exactly the same as that of DTC. The difference between the two approximations is that in variational setting the inducing inputs and covariance function parameters are optimized differently. Titsias (2009) discusses how the inducing inputs and hyperparameters can be seen as variational parameters that should be chosen to optimize the variational lower bound between the true GP posterior and the sparse approximation for it. This setting leads to optimization of modified marginal log likelihood

$$V(\theta, \mathbf{X}_u) = \log[N(\mathbf{y}\,|0, \sigma^2\mathbf{I} + \mathbf{Q}_{\mathrm{f,f}})] - \frac{1}{2\sigma^2}\mathrm{tr}(\mathbf{K}_{\mathrm{f,f}} - \mathbf{K}_{\mathrm{f,u}}\,\mathbf{K}_{\mathrm{u,u}}^{-1}\,\mathbf{K}_{\mathrm{u,f}}) \tag{4.9}$$

with Gaussian observation model. With non-Gaussian observation model the variational lower bound is similar but $\sigma^2\mathbf{I}$ is replaced by $\mathbf{W}^{-1}$ (with Laplace approximation) or $\tilde{\mathbf{\Sigma}}$ (with EP).

4.4 Regression demos with sparse GPs

In this section we will apply sparse GPs for regression problem. We first restate the inference problem. The data consists of i.i.d observations with Gaussian noise. That is the observation model is

$$y_i = f(\mathbf{x}_i) + \epsilon_i, \tag{4.10}$$

where $f(\mathbf{x})$ is given the Gaussian process prior and $\epsilon_i \sim N(0, \sigma_n^2)$.

4.4.1 GP with compactly supported covariance function: demo_ppcsCov

GPstuff utilizes the sparse matrix routines from SuiteSparse written by Tim Davis (http://www.cise.ufl.edu/research/sparse/SuiteSparse/). This package should be installed before using the compactly supported covariance functions.

We will analyze a US annual precipitation data from year 1995, which contains 5776 data points. The GP constructed utilizes compactly supported covariance function `gpcf_ppcs2`, for which reason the inference is much faster than with globally supported covariance function (such as `gpcf_sexp`). The data was previously used by Vanhatalo and Vehtari (2008) and can be downloaded from (http://www.image.ucar.edu/Data/).

The data analyzis with compactly supported covariance functions is performed exactly the same way as with globally supported covariance functions, such as squared exponential. The user interface of GPstuff makes no difference between these two types of covariance functions but the code is optimized so that it uses sparse matrix routines whenever the covariance matrix is sparse. Thus, we can construct the model, find the MAP estimate for the hyperparameters and predict to new input locations in a familiar way:

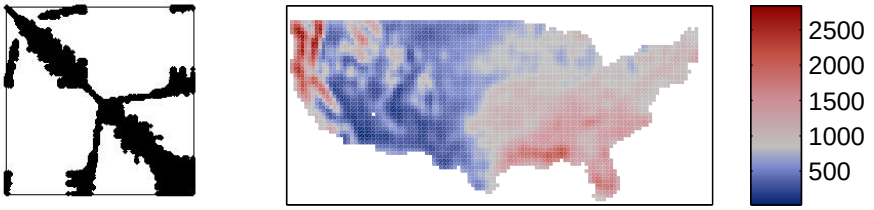
```
% Create covariance function
pl2 = prior_gamma('init', 'sh', 5, 'is', 1);
pm2 = prior_t('init', 'nu', 1, 's2', 150);
gpcf2 = gpcf_ppcs2('init', 'nin', nin, 'lengthScale', [1 2], 'magnSigma2', 3);
gpcf2 = gpcf_ppcs2('set', gpcf2, 'lengthScale_prior', pl2, 'magnSigma2_prior', pm2);

pn = prior_t('init', 'nu', 4, 's2', 0.3);
gpcfn = gpcf_noise('init', 'noiseSigma2', 1, 'noiseSigma2_prior', pn);

gp = gp_init('init', 'FULL', 'regr', {gpcf2}, {gpcfn}, 'jitterSigma2', 0.001.^2);

w=gp_pak(gp);
fe=str2fun('gp_e');
fg=str2fun('gp_g');
opt = scg2_opt;
opt.tolfun = 1e-3;
opt.tolx = 1e-3;
opt.display = 1;

w=scg2(fe, w, opt, fg, gp, x, y);
gp = gp_unpak(gp,w);

Ef = gp_pred(gp, x, y, xx);
```

The following lines show how to evaluate the sparsity of the covariance matrix and how to plot the non-zore structure of the matrix. With this data the covariance matrix is rather sparse since only about 5% of its elements are non-zero. The structure of the covariance matrix is plotted after the AMD permutation. In the section 7.2.2 we discuss a demo with non-Gaussian likelihood and compactly supported covariance function.

```
K = gp_trcov(gp,x);
nnz(K) / prod(size(K))

p = amd(K);
spy(K(p,p), 'k')
```

4.4.2 GP with Sparse approximations: demo_sparseApprox

This demo analyzes the same data that was discussed in the section 3.1 alongside the demonstration program `demo_regression1`. GP's considered are one with a piece wise polynomial CS covariance function, and FIC, PIC, variational and DTC sparse GP approximation.



(a) The nonzero elements of $\mathbf{K}_{\mathrm{f,f}}$.

(b) The posterior predictive mean surface.

Figure 4.1: The nonzero elements of $\mathbf{K}_{\mathrm{f,f}}$ with $k_{\mathrm{pp,2}}$ function with lenght-scale values [0.8 0.6], and the posterior predictive mean of the latent function in the `demo_ppcsCov` data set.

GP with FIC sparse approximation

The sparse approximation is always a property of the Gaussian process structure so that we can construct the model similarly to the full GP models. The difference is that we have to define the type of the sparse approximation, for example, FIC, and set the inducing inputs to the GP structure. Since the inducing inputs are considered as extra hyperparameters common to all of the covariance functions (there may be more than one covariance function in additive models) they are set into the GP structure instead of the covariance function structure. If we want to optimize the inducing inputs alongside the hyperparameters they need to have a prior as well. Initializing GP structure gives them a uniform prior by default. The model construction for FIC is done as follows:

```
gpcf2 = gpcf_noise('init', 'noiseSigma2', 0.2^2);
gpcf3 = gpcf_sexp('init', 'lengthScale', [1 1], 'magnSigma2', 0.2^2);

[u1,u2]=meshgrid(linspace(-1.8,1.8,6),linspace(-1.8,1.8,6));
X_u = [u1(:) u2(:)];
gp_fic = gp_init('init', 'FIC', 'regr', {gpcf3}, {gpcf2}, 'jitterSigma2', 0.001, 'X_u', X_u)
```

The first two lines initialize the covariance function and the noise function and the last three lines initialize the inducing inputs and the GP structure. It should be noticed that if we do not explicitly set the priors for the covariance function parameters or the inducing inputs they are given a uniform prior. Our model is now ready and we can start to optimize the parameters. The posterior predictive mean and the inducing inputs are shown in Figure 4.2.

```
fe=str2fun('gp_e');
fg=str2fun('gp_g');
opt = scg2_opt;
opt.tolfun = 1e-3;
opt.tolx = 1e-3;
opt.display = 1;
opt.maxiter = 50;

w = gp_pak(gp_fic);
w=scg2(fe, w, opt, fg, gp_fic, x, y);
gp_fic = gp_unpak(gp_fic,w);
```

Sometimes it is more efficient to optimize the hyperparameters and inducing inputs separately, so that we iterate the separate optimization steps until convergence. This can be done, for example, in the following way

```
iter = 1
e = gp_e(w,gp_fic,x,y)
e_old = inf;
while iter < 100 & abs(e_old-e) > 1e-3
    e_old = e;

    gp_fic = gp_init('set', gp_fic, 'infer_params', 'covariance');
    w = gp_pak(gp_fic);
    w=scg2(fe, w, opt, fg, gp_fic, x, y);
    gp_fic = gp_unpak(gp_fic,w);

    gp_fic = gp_init('set', gp_fic, 'infer_params', 'inducing');
    w = gp_pak(gp_fic);
    w=scg2(fe, w, opt, fg, gp_fic, x, y);
    gp_fic = gp_unpak(gp_fic,w);
    e = gp_e(w,gp_fic,x,y);
    iter = iter +1;
    [iter e]
end
```

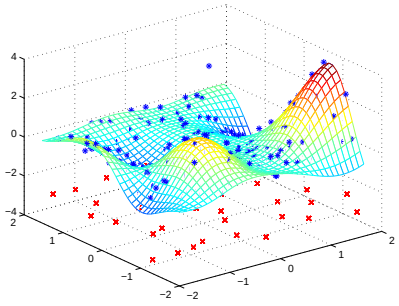
The parameters to be optimized are defined by the field `infer_params` in the GP structure. This field is by default `'covariance+inducing+likelihood'`, which means that covariance function parameters (`'covariance'`), inducing inputs (`'inducing'`) and parameters of the likelihood (`'likelihood'`) are all optimized (or sampled with `gp_mc` or integrated over with `gp_ia`). Since we do not have likelihood structure (our likelihood is defined by a noise covariance function in the regression case) the last option is redundant. In the above example, we alternate between the optimization for inducing input and hyperparameters by setting `infer_params` to either one of them.

The field `infer_params` in the GP structure regulates which parameters are considered fixed and which are inferred in the group level. There may also be situations when we want to fix one of the parameters inside these groups. For example, noise variance or one length-scale. If this is the case, then the parameter to be fixed should be given an empty prior. If the parameter has a prior structure it is an indicator that we want to infer that parameter. An example of fixing one likelihood parameter is given in the section **??**.

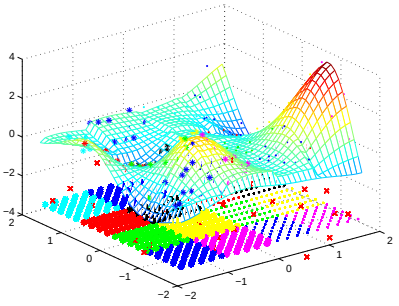
GP with PIC sparse approximation

The difference between FIC and PIC is that we need to appoint every data point into one block. The block structure is common to all of the covariance functions similarly to the inducing inputs for which reason the blocks are stored in the GP structure. The information on the blocks is stored in a cell array which contains a vector of indices for every block. With this data set we divide the two dimensional input space into 16 equally sized square blocks and appoint the training data into these according to the input co-ordinates. This and the initialization of the GP structure are done as follows:

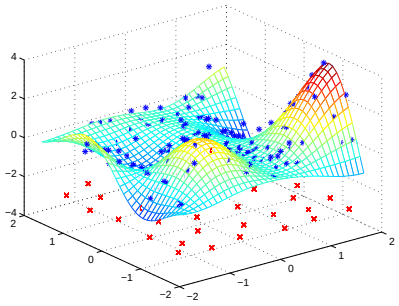
```
% Initialize the inducing inputs in a regular grid over the input space
[u1,u2]=meshgrid(linspace(-1.8,1.8,6),linspace(-1.8,1.8,6));
X_u = [u1(:) u2(:)];

% Initialize test points
[p1,p2]=meshgrid(-1.8:0.1:1.8,-1.8:0.1:1.8);
p=[p1(:) p2(:)];

% set the data points into clusters. Here we construct two cell arrays.
%  trindex  contains the block index vectors for training data. That is
%           x(trindex{i},:) and y(trindex{i},:) belong to the i'th block.
%  tstindex contains the block index vectors for test data. That is test
%           inputs p(tstindex{i},:) belong to the i'th block.
%
```



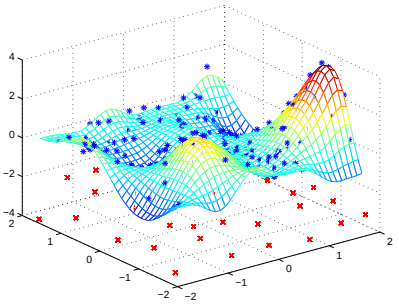
(a) FIC sparse approximation.



(b) PIC sparse approximation.



(c) Variational sparse approximation.



(d) DTC/SOR sparse approximation.

Figure 4.2: The posterior predictive mean of the latent function in the `demo_sparseApprox` data set obtained with FIC, PIC, variational and DTC/SOR sparse approximations. The red crosses show the optimized inducing inputs and the block areas for PIC are colored underneath the latent surface.

```
b1 = [-1.7 -0.8 0.1 1 1.9];
mask = zeros(size(x,1),size(x,1));
trindex={}; tstindex={};
for i1=1:4
    for i2=1:4
        ind = 1:size(x,1);
        ind = ind(: , b1(i1)<=x(ind',1) & x(ind',1) < b1(i1+1));
        ind = ind(: , b1(i2)<=x(ind',2) & x(ind',2) < b1(i2+1));
        trindex{4*(i1-1)+i2} = ind';
        ind2 = 1:size(p,1);
        ind2 = ind2(: , b1(i1)<=p(ind2',1) & p(ind2',1) < b1(i1+1));
        ind2 = ind2(: , b1(i2)<=p(ind2',2) & p(ind2',2) < b1(i2+1));
        tstindex{4*(i1-1)+i2} = ind2';
    end
end

% Create the PIC GP data structure and set the inducing inputs and block indeces
gpcf1 = gpcf_sexp('init', 'lengthScale', [1 1], 'magnSigma2', 0.2^2);
gpcf2 = gpcf_noise('init', 'noiseSigma2', 0.2^2);

gp_pic = gp_init('init', 'PIC', 'regr', {gpcf1}, {gpcf2}, 'X_u', X_u, 'tr_index', trindex);
gp_pic = gp_init('set', gp_pic, 'jitterSigma2', 0.001);
```

Now the cell array `trindex` contains the block index vectors for training data. It means that, for example, the inputs and outputs `x(trindexi,:)` and `y(trindexi,:)` belong to the i'th block. Similarly the `tstindex` cell array contains the block indeces for the test data inputs `p`.

The optimization of hyperparameters and inducing inputs is done exactly the same way as with FIC or full GP model. Making predictions is, however, little different. We have to give the `gp_pred` function one extra input `tstindex`, which defines how the prediction inputs are allocated in blocks. The lines of code for optimization and prediction are below and Figure 4.2 shows the predicted surface. One should notice that the PIC's prediction is discontinuous whereas the prediction with FIC and full GP are continuous. The discontinueties take place in the block boundaries and are a result of discontinuous covariance function that PIC resemples. This issue is discussed in more detail by Vanhatalo et al. (2010).

```
opt = scg2_opt;
opt.tolfun = 1e-3;
opt.tolx = 1e-3;
opt.display = 1;
opt.maxiter = 50;

w = gp_pak(gp_pic);
w=scg2(fe, w, opt, fg, gp_pic, x, y);
gp_pic = gp_unpak(gp_pic,w);

Ef_pic = gp_pred(gp_pic, x, y, p, 'tstind', tstindex);
```

GP with variational, DTC and SOR sparse approximation

The variational, DTC and SOR sparse approximations are constructed similarly to FIC. The only difference is that the type of the GP structure is either VAR or DTC. The GP structure initializations are done as follows:

```
gp_var = gp_init('init', 'VAR', 'regr', {gpcf3}, {gpcf2}, 'jitterSigma2', 0.001, 'X_u', X_u);
gp_dtc = gp_init('init', 'DTC', 'regr', {gpcf3}, {gpcf2}, 'jitterSigma2', 0.001, 'X_u', X_u);
gp_var = gp_init('init', 'SOR', 'regr', {gpcf3}, {gpcf2}, 'jitterSigma2', 0.001, 'X_u', X_u);
```

Figure 4.2 shows the predictive mean of all the sparse approximations (the mean of SOR is the same as that of DTC). It should be noticed that variational approximation is closest to the full GP solution in Figure 3.2. The next closest to full solution are FIC and PIC approximations. FIC works rather differently on one corner of the region whereas

the latent surface predicted by PIC contains discontinueties. DTC suffers most on the borders of the region. An other sparse GP demo is `demo_compareSparseGP`, which is not discussed here. It demonstrates the differences between FIC, variational and DTC in other context and people interested on the topic should run it. See also the discussions on the differences between these sparse approximations given by (Quiñonero-Candela and Rasmussen, 2005; Snelson, 2007; Titsias, 2009; Alvarez et al., 2010).

4.4.3 Sparse GP models with non-Gaussian likelihoods

The extension of sparse GP models to non-Gaussian likelihoods is very straightforward in GPstuff. User can define the sparse GP just as described in the previous two sections and then continue with the construction of likelihood exactly the same way as with full GP. The Laplace approximation, EP and integration methods can be used with the same commands as with full GP. The only difference is that now user has to decide if he wants to optimize also the inducing inputs.

Chapter 5

Model assesment and comparison

This far we have concentrated on model building and inference without paying attention on how good the models are. There are various means to asses the goodness of the model and its predictive performance and GPstuff provides built in functionalities to many common test statistics. In this chapter, we will briefly discuss the model comparison and assessment in general and introduce a few basic methods that can be conducted routinely with GPstuff's tools.

5.1 Introduction

In prediction problems it is natural to assess the predictive performance of the model by focusing on the predictive distribution of the model (Good, 1952; Bernardo and Smith, 2000). The posterior predictive distribution of an output y_{n+1} for the new input $\mathbf{x}_{n+1}$ given the training data $\mathcal{D} = \{(\mathbf{x}_i, y_i);\, i = 1, 2, \ldots, n\}$ is obtained by marginalizing over the unknown latent variable and hyperparameters θ given the model M

$$p(y_{n+1}|\,\mathbf{x}_{n+1}, \mathcal{D}, M) = \int p(y_{n+1}|\,\mathbf{x}_{n+1}, \theta, \mathcal{D}, M)p(\theta|\,\mathbf{x}_{n+1}, \mathcal{D}, M)d\theta, \qquad (5.1)$$

where $p(y_{n+1}|\,\mathbf{x}_{n+1}, \theta, \mathcal{D}, M) = \int p(y_{n+1}|f_{n+1})p(f_{n+1}|\,\mathbf{x}_{n+1}, \theta, \mathcal{D}, M)df_{n+1}$. In the following, we will assume that knowing $\mathbf{x}_{n+1}$ does not give more information about θ, that is, $p(\theta|\,\mathbf{x}_{n+1}, \mathcal{D}, M) = p(\theta|\mathcal{D}, M)$.

To estimate the predictive performance of the model we would like to compare the posterior predictive distribution to future observations from the same process that generated the given set of training data $\mathcal{D}$. Agreement or discrepancy between the predictive distribution and the observations can be measured with a utility or loss function. Preferably, the utility u would be application-specific, measuring the expected benefit or cost of using the model. Good generic utility function is the log-score which, when used for predictive density, is called log predictive likelihood, $\log p(y_{n+1}|\,\mathbf{x}_{n+1}, \mathcal{D}, M)$. It measures how well the model estimates the whole predictive distribution (Bernardo, 1979) and is thus especially useful in model comparison.

Usually since future observations are not yet available, we need to estimate the

expected utility by taking the expectation over the future data distribution

$$\bar{u} = \mathrm{E}_{(\mathbf{x}_{n+1}, y_{n+1})} \left[u \left(y_{n+1}, \mathbf{x}^{n+1}, \mathcal{D}, M \right) \right]. \tag{5.2}$$

The expected utility for the next sample is also equivalent to taking the expectation over all the future samples if predictions are made one-by-one without updating the model after seeing new observations.

There are several issues which should be taken into account when considering a suitable model for $p(\mathbf{x}_{n+1}, y_{n+1})$. We do not go into details in here, but assume that 1) if $\mathbf{x}_i$ is a random quantity then $\mathbf{x}_{n+1}$ is also random quantity coming from the same distribution and 2) if $\mathbf{x}_i$ is fixed (e.g. by design) then $\mathbf{x}_{n+i} = \mathbf{x}_i$ and we compute mean marginal prediction performance over $\mathbf{x}_{n+i}$. See (Vehtari and Lampinen, 2002) for some additional discussion on the future data distribution assumptions.

There are also several methods for computing the estimate. The main problem in computing the estimate is how to use $(\mathbf{x}_i, y_i), i = 1...n$ both for training the model and estimate the future data distribution without getting too optimistic estimate by using the data twice. GPStuff provides three basic approaches: marginal likelihood, cross-validation and deviance information criterion.

5.2 Marginal likelihood

Marginal likelihood is often used for model selection (see, e.g. Kass and Raftery, 1995). It corresponds to ML II or with model priors to MAP II estimate in the model space, selecting the model with the highest marginal likelihood or highest marginal posterior probability. The use of this is acceptable in model selection by the same justification as in using point estimates for parameters. It works well if the posterior is concentrated around the mode and using just one model may produce as good results as integration over the whole model distribution.

Marginal likelihood can also be considered as a predictive approach via chain rule

$$p(y_{1,\ldots,n}|M) = p(y_1|M)p(y_2|y_1, M), \ldots, p(y_n|y_{1,\ldots,n-1}, M).$$

In an expected utility sense the logarithm of this is an average of predictive log-likelihoods with the number of data points used for fitting ranging from 0 to $n-1$. Marginal likelihood can be sensitive to the first terms, unless the model performance convergences quickly to a constant level, which usually happens if the model has a small effective number of parameters p_{eff} compared to n. In such cases marginal likelihood can be used as an estimate of the predictive performance. Marginal likelihood avoids using the data twice, but gives a pessimistic estimate for the question how good predictions we can make given n data points.

In GPstuff, if MAP estimate and IA estimate are almost the same, marginal likelihood can be used as a quick estimate to compare models, but we recommend using cross-validation for more through model assessment and selection.

5.3 Cross validation

Cross-validation (CV) is an approach to using an empirical density estimate for the future observations while avoiding the double use of the data. As the distribution of $(\mathbf{x}_{n+1}, y_{n+1})$ is unknown, we approximate it by using the samples we already have,

that is, we assume that the distribution can be reasonably well approximated using the (weighted) training data $\{(\mathbf{x}_i, y_i); i = 1, 2, \ldots, n\}$. To avoid the double use of the data and simulate the fact that the future observations are not in the training data, the ith observation $(\mathbf{x}_i, y_i)$ in the training data is left out, and then the predictive distribution for y_i is computed with a model that is fitted to all of the observations except $(\mathbf{x}_i, y_i)$. By repeating this for every point in the training data, we get a collection of leave-one-out cross-validation (LOO-CV) predictive densities

$$\{p(y_i|\, \mathbf{x}_i, D_{\setminus i}, M); i = 1, 2, \ldots, n\}, \tag{5.3}$$

where $\mathcal{D}_{\setminus i}$ denotes all the elements of $\mathcal{D}$ except $(\mathbf{x}_i, y_i)$. To get the expected utility estimate, these predictive densities are compared to the actual y_i's using the utility u, and the expectation is taken over i

$$\bar{u}_{\mathrm{LOO}} = \mathrm{E}_i \left[u(y_i, \mathbf{x}_i, \mathcal{D}_{\setminus i}, M) \right] \tag{5.4}$$

The right hand side terms are conditioned on $n - 1$ data points, making the estimate almost unbiased.

5.3.1 Leave-one-out cross-validation

For GP with a Gaussian noise model and given covariance parameters, the LOO-CV-predictive densities may be computed using analytical solutions (Sundararajan and Keerthi, 2001), which is implemented in `gp_looe` and `gp_loog`. These can be used instead of optimising the marginal likelihood with `gp_e`, `gp_g` to get a point estimate for the covariance parameters. If posterior is tightly concentrated there is no difference, but in other cases the expected predictive performance produced by different estimates is ordered as IA > LOO > ML II.

5.3.2 k-fold cross-validation

For other than Gaussian models or optimised or integrated covariance parameters, the predictive distributions need to be computed for each leave-something-out case. To reduce computation time, in k-fold-CV, we use only k (e.g. $k = 10$) k-fold-CV distributions $p(\theta|D^{(\setminus s(i))}, M)$ and get a collection of k-fold-CV predictive densities

$$\{p(y_i|\, \mathbf{x}_i, \mathcal{D}_{\setminus s(i)}, M); i = 1, 2, \ldots, n\}, \tag{5.5}$$

where $s(i)$ is a set of data points as follows: the data are divided into k groups so that their sizes are as nearly equal as possible and $s(i)$ is the set of data points in the group where the ith data point belongs. The expected utility estimated by the k-fold-CV is then

$$\bar{u}_{\mathrm{CV}} = \mathrm{E}_i \left[u(y_i, \mathbf{x}_i, \mathcal{D}_{\setminus s(i)}, M) \right]. \tag{5.6}$$

Since the k-fold-CV predictive densities are based on smaller training data sets $\mathcal{D}_{\setminus s(i)}$ than the full data set $\mathcal{D}$, the expected utility estimate is slightly biased. This bias can be corrected using a first order bias correction (Burman, 1989):

$$\bar{u}_{\mathrm{tr}} = \mathrm{E}_i[u(y_i, \mathbf{x}_i, \mathcal{D}, M)] \tag{5.7}$$

$$\bar{u}_{\mathrm{cvtr}} = \mathrm{E}_j \left[\mathrm{E}_i[u(y_i, \mathbf{x}_i, \mathcal{D}_{\setminus s_j}, M)] \right] \quad ; \quad j = 1, \ldots, k \tag{5.8}$$

$$\bar{u}_{\mathrm{CCV}} = \bar{u}_{\mathrm{CV}} + \bar{u}_{\mathrm{tr}} - \bar{u}_{\mathrm{cvtr}}, \tag{5.9}$$

where $\bar{u}_{\mathrm{tr}}$ is the expected utility evaluated with the training data given the training data, that is, the training error or the expected utility computed with the marginal posterior predictive densities, and $\bar{u}_{\mathrm{cvtr}}$ is the average of the expected utilities evaluated with the training data given the k-fold-CV training sets.

GPstuff provides `gp_kfcv`, which computes k-fold-CV and bias-corrected k-fold-CV with log-score and root mean squared error (RMSE). `gp_kfcv` provides also basic variance estimates for the predictive performance estimates. First the mean expected utility $\bar{u}_j$ for each k folds is computed. $\bar{u}_j$'s tends to be closer to Gaussian (due to the central limit theorem) and then the variance of the expected utility is computed as (see, e.g., Dietterich, 1998)

$$\mathrm{Var}[\bar{u}] \approx \mathrm{Var}_j[\bar{u}_j]/k. \tag{5.10}$$

Although some information is lost by first taking the sub-expectations, the estimate is useful indicator of the related uncertainty. See (Vehtari and Lampinen, 2002) for more details on estimating the uncertainty in performance estimates.

5.4 DIC

Deviance information criterion (DIC) is another very popular model selection criterion (Spiegelhalter et al., 2002). With parametric models without any hierarchy it is usually written as

$$p_{\mathrm{eff}} = \mathrm{E}_{\theta|\mathcal{D}}[D(\mathbf{y},\theta)] - D(\mathbf{y}, \mathrm{E}_{\theta|\mathcal{D}}[\theta]) \tag{5.11}$$

$$\mathrm{DIC} = \mathrm{E}_{\theta|\mathcal{D}}[D(\mathbf{y},\theta)] + p_{\mathrm{eff}} \tag{5.12}$$

where p_{eff} is the effective number of parameters and $D = -2\log(p(\mathbf{y}\,|\theta))$ is the deviance. Since our models are hierarchical we need to decide the parameters on focus (see Spiegelhalter et al., 2002, for discussion on this). The parameters on the focus are those over which the expectations are taken when evaluating the effective number of parameters and DIC. In the above equations, the focus is in the hyperparameters and in case of hierarchical GP model of GPstuff the latent variables would be integrated out before evaluating DIC. If we have a MAP estimate for the hyperparameters, we may be interested to evaluate DIC statistics with focus on the latent variables. In this case the above formulation would be

$$p_D(\theta) = \mathrm{E}_{\mathbf{f}\,|\mathcal{D},\theta}[D(\mathbf{y},\mathbf{f})] - D(\mathbf{y}, \mathrm{E}_{\mathbf{f}\,|\mathcal{D},\theta}[\mathbf{f}]) \tag{5.13}$$

$$\mathrm{DIC} = \mathrm{E}_{\mathbf{f}\,|\mathcal{D},\theta}[D(\mathbf{y},\mathbf{f})] + p_D(\theta). \tag{5.14}$$

Here the effective number of parameters is denoted differently with $p_D(\theta)$ since now we are approximating the effective number of parameters in $\mathbf{f}$ conditionally on θ, which is different from the p_{eff}. $p_D(\theta)$ is a function of the hyperparameters and it measures to what extent the prior correlations are preserved in the posterior of the latent variables given θ. For non-informative data $p_D(\theta) = 0$ and the posterior is the same as the prior. The greater $p_D(\theta)$ is the more the model is fitted to the data and really large values indicate overfit. $p_D(\theta)$ can be used for assessing the goodness of the Laplace or EP approximation for the conditional posteior of the latent variables as discussed by Rue et al. (2009) and Vanhatalo et al. (2010). The third option is to evaluate DIC with focus on all the variables, $[\mathbf{f}, \theta]$. In this case the expectations are over $p(\mathbf{f}, \theta|\mathcal{D})$.

DIC can also be considered as an approximation to the expected utility but with changed order of computation

$$\mathrm{E}_{(\mathbf{x}_{n+1},y_{n+1})}\left[\int \log\left[p(y_{n+1}|\,\mathbf{x}_{n+1},\theta,M)\right]p(\theta|\mathcal{D},M)d\theta\right], \tag{5.15}$$

where the order of the logarithm and integration is changed in comparison with the predictive utility with log-score (see equations (5.1) and (5.2) and discussion therein). This could be considered as a lower limit in theory (Jensen's inequality), but actual approximation has its own error.

Not going to details, DIC replaces the predictive distribution with plug-in predictive distribution, where plug-in estimate $\hat{\theta}$ is used. DIC can be derived by using Taylor-series expansion or examining the degrees of freedom of the distribution of the deviance, which is affected by dependencies in posterior predictive distribution due to fitting to the data. Given a utility function u, it is possible to use Monte Carlo samples to estimate $E_\theta[\bar{u}(\theta)]$ and $\bar{u}(E_\theta[\theta])$, and then compute an expected utility estimate as

$$\bar{u}_{\mathrm{DIC}} = \bar{u}(E_\theta[\theta]) + 2\big(E_\theta[\bar{u}(\theta)] - \bar{u}(E_\theta[\theta])\big). \tag{5.16}$$

DIC uses deviance which is $-2n$ times the log-score utility.

5.5 Model assesment demos

The model assesment methods are demonstrated in `demo_modelassesment1` and `demo_modelassesment2`. The former compares the sparse GP approximations to the full GP with regression data. The latter compares the logit and probit likelihoods in GP classification. In the following we assume that the model has been constructed and we concentrate only on evaluating the model performance.

5.5.1 demo_modelassesment1

Assume that we have builded our regression model with Gaussian noise and used optimization method to find the MAP estimate for the hyperparameters. We can then evaluate some statistics on how well the model is doing with these hyperparameter estimates. Below we evaluate the effective number of latent variables and DIC statistics. The former is evaluated with two different approximations.

```
p_eff_latent = gp_peff(gp, x, y);
[DIC_latent, p_eff_latent2] = gp_dic(gp, x, y, 'focus', 'latent');
```

Since we have MAP estimate for the hyperparameters the focus is on the latent variables. In this case we can use also `gp_peff` which returns the effective number of parameters approximated as described by Spiegelhalter et al. (2002)

$$p_D(\theta) \approx n - \mathrm{tr}(\mathbf{K}_{\mathrm{f,f}}^{-1}(\mathbf{K}_{\mathrm{f,f}}^{-1} + \sigma_n^{-2}\mathbf{I})^{-1}). \tag{5.17}$$

When the focus is on the latent variables, the function `gp_dic` evaluates the DIC statistics and the effective number of parameters as described by the equations (5.13) and (5.14). The k-fold-CV expected utility estimate can be evaluated as follows.

```
cvres =  gp_kfcv(gp, x, y);
```

The `gp_kfcv` takes the ready made model structure `gp` and the training data `x` and `y`. The function devides the data into k groups, conducts inference separately for each of the training groups and evaluates the expected utilities with the test groups. Now

that we have regression model and no optional parameters are given the inference is conducted using MAP estimate for the hyperparameters. The default division of the data is into 10 groups. The expected utilities and their variance estimates are stored in the structure `cvres` as follows:

```
cvres =
        mlpd_cv: 0.0500
     Var_lpd_cv: 0.0014
       mrmse_cv: 0.2361
    Var_rmse_cv: 1.4766e-04
        mabs_cv: 0.1922
     Var_abs_cv: 8.3551e-05
```

`gp_kfcv` returns also other statistics if more information is needed and the function can be used to save the results automatically. However, these functionalities are not considered here. Readers interested on detailed analysis should read the help text for `gp_kfcv`.

Now we will turn our attention to other inference methods than MAP estimate for the hyperparameters. Assume we have a record structure from `gp_mc` function with Markov chain samples of the hyperparameters stored in it. In this case, we have two options how to evaluate the DIC statistics. We can set the focus on the hyperparameters or all the parameters (that is hyperparameters and latent variables). The two versions of DIC and effective number of parameters are evaluated as follows.

```
rfull = gp_mc(gp, x, y, opt);

[DIC(2), p_eff(2)] =  gp_dic(rfull, x, y, 'focus', 'hyper');
[DIC2(2), p_eff2(2)] =  gp_dic(rfull, x, y, 'focus', 'all');
```

Here the first line performs the MCMC samplin with options `opt`. The last two lines evaluate the DIC statistics. With Mackov chain sample, we cannot use the `gp_peff` function to evaluate $p_D(\theta)$ since that is a special function for models with fixed hyperparameters. The k-fold-CV is conducted with MCMC methods as easily as with MAP estimate. The only difference is that we have to define that we want to use MCMC and to give the sampling options for `gp_kfcv`. These steps are done as follows:

```
opt.nsamples= 100;
opt.repeat=4;
opt.hmc_opt = hmc2_opt;
opt.hmc_opt.steps=4;
opt.hmc_opt.stepadj=0.05;
opt.hmc_opt.persistence=0;
opt.hmc_opt.decay=0.6;
opt.hmc_opt.nsamples=1;
hmc2('state', sum(100*clock));

cvres =  gp_kfcv(gp, x, y, 'inf_method', 'MCMC', 'opt', opt);
```

With integration approximation evaluating the DIC and k-fold-CV statistics is very much similar to the MCMC approach. The same steps as with MCMC are taken with, for example, grid integration as follows.

```
opt.opt_scg = scg2_opt;
opt.int_method = 'grid';
opt.step_size = 2;

gp_array = gp_ia(gp, x, y, opt);

models{3} = 'full_IA';
[DIC(3), p_eff(3)] =  gp_dic(gp_array, x, y, 'focus', 'hyper');
[DIC2(3), p_eff2(3)] =  gp_dic(gp_array, x, y, 'focus', 'all');

% Then the 10 fold cross-validation.
cvres = gp_kfcv(gp, x, y, 'inf_method', 'IA', 'opt', opt);
```

This far we have demonstrated how to use DIC and k-fold-CV functions with full GP. The function can be used with sparse approximations exactly the same way as with full GP and this is demonstrated in `demo_modelassesment1` for FIC and PIC.

5.5.2 demo_modelassesment2

In the last section we considered model assesment with regression model. The same analysis can be conducted for other likelihoods as well. The functions `gp_peff`, `gp_dic` and `gp_kfcv` work similarly for non-Gaussian likelihoods as for a Gaussian one. The only difference is that now we have to integrate over latent variables approximately which is demonstrated for logit and probit likelihoods in `demo_modelassesment2`.

The way latent variables are treated is defined in the field `latent_method` of the GP structure and this is initialized when constructing the model as discussed in the section 3.2.2. Consider now that we have conducted the analyzis with Laplace approximation and MAP estimate for the hyperparameters as in the section 3.2.2. Thus, we have a GP structure `gp` with optimized hyperparameters and `latent_method` set to Laplace approximation. We can then evaluate the DIC and k-fold-CV statistics as follows:

```
p_eff_latent = gp_peff(gp, x, y);
[DIC_latent, p_eff_latent2] = gp_dic(gp, x, y, 'latent');

% Evaluate the 10-fold cross validation results.
cvres = gp_kfcv(gp, x, y);
```

These are exactly the same lines as presented in the previous section. The only difference is that the GP structure `gp` and the data `x` and `y` are different. Since the likelihood is not Gaussian all the integrations over latent variables in `gp_dic` and `gp_kfcv` are done with respect to the approximate conditional posterior $q(\mathbf{f}\,|\hat{\theta}, \mathcal{D})$ obtained from the Laplace approximation. The effective number of parameters returned by `gp_peff` is evaluated as in the equation (5.17) with the modification that $\sigma_n^{-2}\mathbf{I}$ is replaced by $\mathbf{W}$.

If expectation propagation is used for inference the model assesment is conducted similarly as with Laplace approximation. Also the MCMC and IA solutions are evaluated identically to the Gaussian case. For this reason the code is not repeated here.

Chapter 6

Playing around with covariance functions

In the previous chapters we have not paid much attention on the choice of the covariance function. However, GPstuff has rather versatile collection of covariance functions, which can be combined in numerous ways. The different functions are collected in the appendix .1. This chapter demonstrates some of the functions and ways to combine them.

6.1 Neural network covariance function

A good example of covariance function that has very different properties than the standard stationary covariance functions such as squared exponential or Mátern covariance functions is the neural network covariance function. In this section we will demonstrate its use in two simple regression problems. The squared exponential covariance function is taken as a reference and the code is found from the `demo_neuralnetCov`.

The neural network model is approximated with Gaussian process...

Jaakko jatkaa tarinan loppuun!

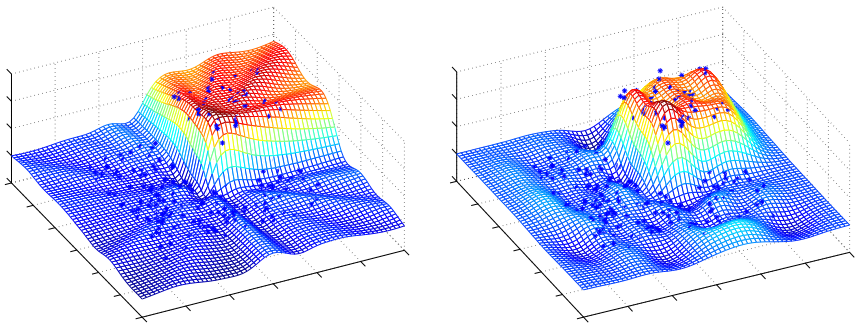
6.2 Additive models

In many practical situations, a GP prior with only one covariance function may be too restrictive since such a construction can model effectively only one phenomenon. For example, the latent function may vary rather smoothly across the whole area of interest, but at the same time it can have fast local variations. In this case, a more reasonable model would be

$$f(\mathbf{x}) = g(\mathbf{x}) + h(\mathbf{x}), \tag{6.1}$$

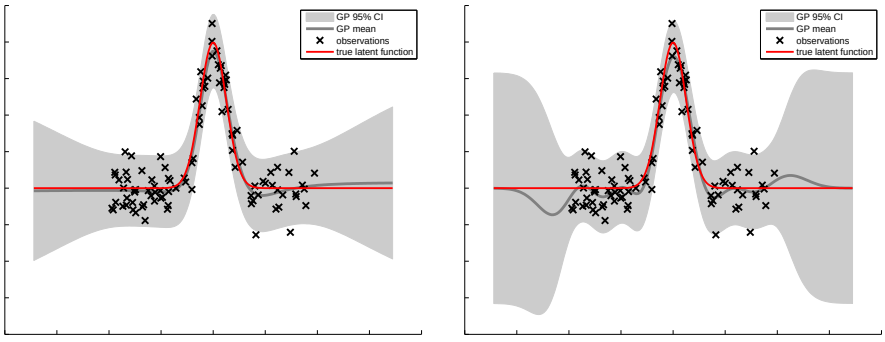
where the latent value function is a sum of two functions, of which the other is slowly, and the other fast varying. We can place GP prior for both of the functions g and h with different covariance functions, which results in a combined prior

$$p(\mathbf{f}\,|\mathbf{X}) = \mathrm{N}(\mathbf{f}\,|\mathbf{0}, \mathbf{K}_{\mathrm{g,g}} + \mathbf{K}_{\mathrm{h,h}}). \tag{6.2}$$



(a) Neural network solution. (b) Squared exponential solution.

Figure 6.1: GP latent mean predictions (using a MAP estimate) with neural network or squared exponential covariance functions. The 2D data is generated from a step function.



(a) Neural network solution. (b) Squared exponential solution.

Figure 6.2: GP solutions (a MAP estimate) with neural network or squared exponential covariance functions.

The marginal likelihood and posterior predictive distribution are as before with $\mathbf{K}_{\mathrm{f,f}} = \mathbf{K}_{\mathrm{g,g}} + \mathbf{K}_{\mathrm{h,h}}$. However, if we are interested on only, say, phenomenon g, we can consider the h part of the latent function as correlated noise and evaluate the predictive distribution

$$\tilde{\mathbf{g}}|\tilde{\mathbf{X}}, \mathcal{D}, \theta \sim N(\mathbf{K}_{\tilde{g},g}(\mathbf{K}_{\mathrm{f,f}} + \sigma^2\mathbf{I})^{-1}\mathbf{y}, \mathbf{K}_{\tilde{g},\tilde{g}} - \mathbf{K}_{\tilde{g},g}(\mathbf{K}_{\mathrm{f,f}} + \sigma^2\mathbf{I})^{-1}\mathbf{K}_{g,\tilde{g}}). \quad (6.3)$$

The multiple length-scale model can be formed also using specific covariance functions. For example, rational quadratic covariance function (`gpcf_rq`) can be seen as a scale mixture of squared exponential covariance functions (Rasmussen and Williams, 2006), and could be useful for data that contain both local and global phenomena. However, using sparse approximations with the rational quadratic would prevent it from modeling local phenomena . The additive model (6.2) suits better for sparse GP formalism since it enables to combine FIC with CS covariance functions.

As discussed in section 4.2, FIC can be interpreted as a realization of a special kind of covariance function. By adding FIC with CS covariance function, for example (4.1), one can construct a sparse additive GP prior

$$\mathbf{f} \mid \mathbf{X}, \mathbf{X}_u, \theta \sim \mathrm{N}(\mathbf{0}, \mathbf{Q}_{\mathrm{f,f}} + \hat{\mathbf{\Lambda}}). \quad (6.4)$$

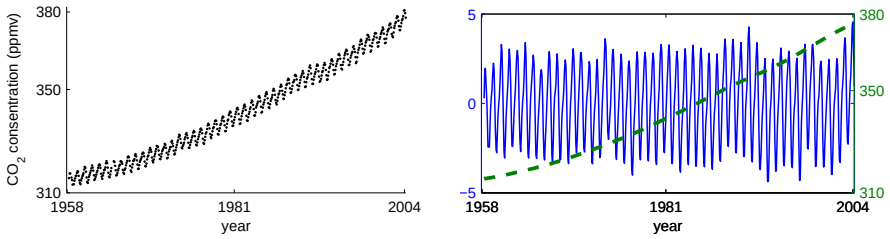
This prior will be referred as CS+FIC. Here, the matrix $\hat{\mathbf{\Lambda}} = \mathbf{\Lambda} + \mathbf{K}_{\mathrm{f,f}}^{\mathrm{CS}}$ is sparse with the same sparsity structure as in $\mathbf{K}_{\mathrm{f,f}}^{\mathrm{CS}}$ and, thus, it is fast to use in computations and cheap to store. CS+FIC can be extended to have more than one component. However, it should be remembered that FIC works well only for long length-scale phenomena and the computational benefits of CS functions are lost if their length-scale gets too large (Vanhatalo et al., 2010). For this reason the CS+FIC should be constructed so that possible long length-scale phenomena are handled with FIC part and the short length-scale phenomena with CS part. The implementation of the CS+FIC model follows closely the implementation of FIC and PIC (for details see Vanhatalo and Vehtari, 2008, 2010).

In the following sections we will demonstrate the additive models with two problems. First we will consider full GP with covariance function that is a sum of periodic and squared exponential covariance function. This GP prior is demonstrated for a Gaussian and non-Gaussian likelihood. The second demo concentrates on sparse GPs in additive models. The FIC, PIC and CS+FIC sparse models are demonstrated with data set that contains both long and short length-scale phenomena.

6.2.1 Additive models demo: demo_periodicCov

In this section we will discuss the demonstration program `demo_periodicCov`. This demonstrates the use of a periodic covariance function `gpcf_periodic` with two data sets, the Mauna Loa CO2 data (see, for example, Rasmussen and Williams, 2006) and the monthly Finnish drowning statistics 2002-2008. The first data is a regression problem with Gaussian noise whereas the second consist of count data that is modeled with Poisson observation model. Here, we will describe only the regression problem the other data can be examined by running the demo.

We will analyze the Maunaloa CO2 data with two additive models. The first one utilizes covariance function that is a sum of squared exponential and piece-wise polynomial $k_{\mathrm{se}}(\mathbf{x}, \mathbf{x}') + k_{\mathrm{pp},2}(\mathbf{x}, \mathbf{x}')$. The solution of this model that shows the long and short length-scale phenomena separately is visualized in Figure 6.3 together with the original data. This model interpolates the underlying function well but as will be demonstrated



(a) The Maunaloa CO_2 data.

(b) The long and short (left scale) term component (right scale).

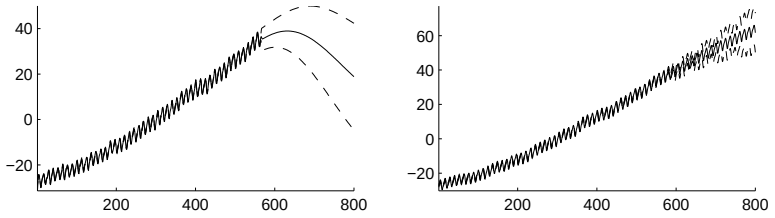
Figure 6.3: The Maunaloa CO_2 data and prediction for the long term and short term component demo_regression2.

later its predictive preperties into the future are not so good. Better predictive performance is obtained by adding up two squared exponential and one periodic covariance function $k_{\text{se}}(\mathbf{x}, \mathbf{x}' \,|\theta_1) + k_{\text{se}}(\mathbf{x}, \mathbf{x}' \,|\theta_2) + k_{\text{periodic}}(\mathbf{x}, \mathbf{x}')$, which is build as follows.

```
gpcf1 = gpcf_sexp('init', 'lengthScale', 67*12, 'magnSigma2', 66*66);
gpcfp = gpcf_periodic('init', 'nin', nin, 'lengthScale', 1.3, 'magnSigma2', 2.4*2.4);
gpcfp = gpcf_periodic('set', gpcfp, 'period', 12,'optimPeriod',1,'lengthScale_exp', 90*12, 'decay', 1);
gpcfn = gpcf_noise('init', 'noiseSigma2', 0.3);
gpcf2 = gpcf_sexp('init', 'lengthScale', 2, 'magnSigma2', 2);
pl = prior_t('init', 's2', 10, 'nu', 3);
pn = prior_t('init', 's2', 10, 'nu', 4);

gpcf1 = gpcf_sexp('set', gpcf1, 'lengthScale_prior', pl, 'magnSigma2_prior', pl);
gpcf2 = gpcf_sexp('set', gpcf2, 'lengthScale_prior', pl, 'magnSigma2_prior', pl);
gpcfp = gpcf_periodic('set', gpcfp, 'lengthScale_prior', pl, 'magnSigma2_prior', pl);
gpcfp = gpcf_periodic('set', gpcfp, 'lengthScale_exp_prior', pl, 'period_prior', pn);
gpcfn = gpcf_noise('set', gpcfn, 'noiseSigma2_prior', pn);

gp = gp_init('init', 'FULL', 'regr', {gpcf1, gpcfp, gpcf2}, {gpcfn}, 'jitterSigma2', 0.003)
```

An additive model is constructed similarly to a model with just one covariance function. The only difference is that now we give more than one covariance function structure for the `gp_init`. The inference with additive model is conducted exactly the same way as in the `demo_regression1`. The below lines summarize the hyperparameter optimization and conduct the prediction for the whole process f and two components g and h whose covariance functions are $k_{\text{se}}(\mathbf{x}, \mathbf{x}' \,|\theta_1)$ and $k_{\text{se}}(\mathbf{x}, \mathbf{x}' \,|\theta_2) + k_{\text{periodic}}(\mathbf{x}, \mathbf{x}')$ respectively. The prediction for latent functions that are related to only subset of the covariance functions used for training is done by giving a fift argument for the `gp_pred`. This argument tells which covariance functions are used for the prediction. If we want to use more than one covariance function for prediction, and there are more than two covariance functions in the model, the fifth argument needs to be a vector.

```
fe=str2fun('gp_e');
fg=str2fun('gp_g');

opt = scg2_opt;
opt.tolfun = 1e-3;
opt.tolx = 1e-3;
opt.display = 1;
opt.maxiter = 50;

w0 = gp_pak(gp);
```



(a) Two additive components.

(b) Three additive components with periodic covariance function.

Figure 6.4: The Maunaloa CO_2 data. Prediction with two different models. On the left model with covariance function $k_{\text{se}}(\mathbf{x}, \mathbf{x}') + k_{\text{pp},2}(\mathbf{x}, \mathbf{x}')$ and on the right a model with covariance function $k_{\text{se}}(\mathbf{x}, \mathbf{x}' \,|\theta_1) + k_{\text{se}}(\mathbf{x}, \mathbf{x}' \,|\theta_2) + k_{\text{periodic}}(\mathbf{x}, \mathbf{x}')$. It can be seen that the latter has more predictive power.

```
w=scg2(fe, w0, opt, fg, gp, x, y);
gp = gp_unpak(gp,w);

[Ef_full, Varf_full, Ey_full, Vary_full] = gp_pred(gp, x, y, x);
[Ef_full1, Varf_full1] = gp_pred(gp, x, y, x, 'predcf', 1);
[Ef_full2, Varf_full2] = gp_pred(gp, x, y, x, 'predcf', [2 3]);
```

The two components `Ef_full1` and `Ef_full2` above are basically identical to the component shown in Figure 6.3, which shows that there is no practical difference in the interpolation performance between the two models considered in this demo. However, the additive model with periodic component has much more predictive power into the future. This is illustrated in Figure 6.4 where one can see that the prediction with non-periodic model starts to decrease towards prior mean very quickly, and does not extrapolate the period, whereas the periodic model extrapolates the almost linear increase and periodic behaviour. The MCMC or grid integration approach for the additive model is identical to regression with only one covariance function and is not repeated here.

6.2.2 Additive models with sparse approximations

The Maunaloa CO2 data set is studied with sparse additive Gaussian processes in the demo `demo_regression2`. There the covariance is $k_{\text{se}}(\mathbf{x}, \mathbf{x}') + k_{\text{pp},2}(\mathbf{x}, \mathbf{x}')$ since periodic covariance doas not work well with sparse approximations. The model construction and inference is conducted similarly as in the previous section so we will not repeat it here. However, it is worth mentioning few things that should be noticed when running the demo.

PIC works rather well for this data set whereas FIC fails to recover the fast varying phenomenon. The reason for this is that the inducing inputs are too sparsely located so that FIC can not reveal the short length-scale phenomenon. In general, FIC is able to model only phenomena whose length-scale is long enough compared to the average distance between adjacent inducing inputs (see Vanhatalo et al., 2010, for details). PIC on the other hand is able to model also fast varying phenomena inside the blocks. Its drawback, however, is that the correlation structure is discontinuous which may result in discontinuous predictions. The CS+FIC model corrects these deficiencies.

In FIC and PIC the inducing inputs are parameters of every covariance function, which means that all the correlations are circulated through the inducing inputs and the shortest lenght-scale the GP is able to model is defined by the locations of the inducing inputs. The CS+FIC sparse GP is build differently. There the CS covariance function does not utilice inducing inputs but evaluates the correlations exactly. This enables the GP model to capture both the long and short lenght-scale phenomena. The GPstuff package is coded so that if the GP structure is defined to be CS+FIC all the CS functions are treated outside FIC approximation. Thus, the CS+FIC model requires that there is at least one CS covariance function and one globally supported function (such as squared exponential). If there are more than two covariance functions in the GP structure all the globally supported functions utilize inducing inputs and all the CS functions are added to $\hat{\Lambda}$

6.3 Additive covariance functions with selected variables

In the demo (`demo_regression_additive`), we demonstrate how covariance functions can be modified so that they are functions of only portion of inputs. We will consider modelling an artificial 2D regression data with additive covariance functions where the covariance functions use either the first or second input variable. That is the covariance is $k_1(x_1, x_1'|\theta_1) + k_2(x_1, x_2'|\theta_2)$, where the covariance functions are of type $k_1(x_1, x_1'|\theta_1) : \Re \mapsto \Re$ instead of $k(\mathbf{x}, \mathbf{x}'|\theta) : \Re^2 \mapsto \Re$, which has been the usual case in previous demos. Remember the notation $\mathbf{x} = [x_1, ..., x_d]^{\mathrm{T}}$ and $\mathbf{x}' = [x_1', ..., x_d']^{\mathrm{T}}$. Also solutions from the covariance function that uses both input variables are shown for comparison. In the regression we assume a Gaussian noise. The hyperparameter values are set to their MAP estimate. The models considered in this demo utilize the following six covariance functions:

- constant and linear
- costant and squared exponential for the first input and linear for the second input
- squared exponential for the first input and squared exponential for the second input
- squared exponential
- neural network for the first input and neural network for the second input
- neural network.

We will demonstrate how to construct the first, second and fifth model.

A linear covariance function with constant term can be constructed in GPstuff as

```
% constant covariance function
gpcf_c = gpcf_constant('init', 'constSigma2', 1);
gpcf_c = gpcf_constant('set', gpcf_c, 'constSigma2_prior', pt);

% linear covariance function
gpcf_l = gpcf_linear('init');
gpcf_l = gpcf_linear('set', gpcf_l, 'coeffSigma2_prior', pt);
```

Gaussian process using this linear covariance function is constructed as previously

```
gp = gp_init('init', 'FULL', 'regr', {gpcf_c gpcf_l}, {gpcf_n});
gp = gp_init('set', gp, 'jitterSigma2', jitter, 'infer_params', 'covariance');
```

where we have added a Gaussian noise covariance function as well. In this model, the covariance function is $c + k_{\text{linear}}(\mathbf{x}, \mathbf{x}'|\theta), \mathbf{x} \in \Re^2$, which means that the components of $\mathbf{x}$ are coupled in the in the covariance function k_{linear}.. The constant term (`gpcf_const`) is denoted by c.

The second model is more flexible. It contains a squared exponential, which is a function of the first input dimension x_1, and a linear covariance function, which is a function of the second input dimension x_2. The additive covarince function is $k_{\text{se}}(x_1, x_1'|\theta_1) + k_{\text{linear}}(x_2, x_2'|\theta_2)$ which is a mapping from $\Re^2$ to $\Re$. With the squared exponential covariance function, the inputs to be used can be selected using a metric structure as follows:

```
% Covariance function for the first input variable
gpcf_s1 = gpcf_sexp('init', 'magnSigma2', 0.15, 'magnSigma2_prior', pt);
% create metric structure:
metric1 = metric_euclidean('init', {[1]},'lengthScales',[0.5], 'lengthScales_prior', pt);
% set the metric to the covariance function structure:
gpcf_s1 = gpcf_sexp('set', gpcf_s1, 'metric', metric1);
```

Here we construct the covariance function structure just as before. We also set a prior structure for the magnitude `pt`. To modify squared exponential covariance function so that it depends only on a subset of inputs is done using a metric structure. In this example we use `metric_euclidean` which allows user to group the inputs so that all the inputs in one group are appointed to same length-scale. The metric structure has function handles which then evaluate, for example, the distance with this modified euclidean metric. For example, for inputs $\mathbf{x}, \mathbf{x}' \in \Re^4$ the modified distance could be

$$r = \sqrt{(x_1 - x_1')^2/l_1^2 + \left((x_2 - x_2')^2 + (x_3 - x_3')^2\right)/l_2^2 + (x_4 - x_4')^2/l_3^2} \qquad (6.5)$$

where the second and third input dimension are given the same lenght-scale. This is different from the previously used, $r = \sqrt{\sum_{i=1}^{d}(x_i - x_i')^2/l_i^2}$ and $r = \sqrt{\sum_{i=1}^{d}(x_i - x_i')^2/l^2}$, that can be defined by the covariance function itself. The metric structure can be used with any stationary covariance function, that is with functions of type $k(\mathbf{x}, \mathbf{x}') = k(r)$. The reason why this property is implemented by using one extra structure is that this way user does not need to modify the covariance function when redefining the distance. Only a new metric file needs to be created. It should be noticed, though, that not all metrics lead to positive definite covariances with all covariance functions. For example, the squared exponential $\sigma^2 \exp(-r^2)$ is not positive definite with metric induced by L_1 norm $r = \sum_{i=1}^{d} |\mathbf{x}_i - \mathbf{x}_i'|/l_i$ whereas the exponential $\sigma^2 \exp(-r)$ is.

A metric structure can not be used with the linear covariance function, since it is not stationary but a smaller set of inputs can be chosen by using the field *selectedVariables*. In this demo, we select only the second input variable as

```
gpcf_l2 = gpcf_linear('init', 'selectedVariables', [2]);
gpcf_l2 = gpcf_linear('set', gpcf_l2, 'coeffSigma2_prior', pt);
```

The result with this model is shown in Figure 6.5(b).

The neural network covariance function is another non-stationary covariance function with which the metric structure can not be used. However, a smaller set of input variables can be chosen similarly as with the linear covariance function using the field *selectedVariables*. In this demo, we consider additive neural network covariance functions, each having one input variable $k_{\text{nn}}(x_1, x_1'|\theta_1) + k_{\text{nn}}(x_2, x_2'|\theta_2)$. In GPstuff this can be done as

```
gpcf_nn1 = gpcf_neuralnetwork('init', 'weightSigma2', 1, 'biasSigma2', 1, 'selectedVariables', [1]);
gpcf_nn1 = gpcf_neuralnetwork('set', gpcf_nn1, 'weightSigma2_prior', pt, 'biasSigma2_prior', pt);

gpcf_nn2 = gpcf_neuralnetwork('init', 'weightSigma2', 1, 'biasSigma2', 1, 'selectedVariables', [2]);
gpcf_nn2 = gpcf_neuralnetwork('set', gpcf_nn2, 'weightSigma2_prior', pt, 'biasSigma2_prior', pt);
```

```
gp = gp_init('init', 'FULL', 'regr', {gpcf_nn1,gpcf_nn2}, {gpcf_n});
gp = gp_init('set', gp, 'jitterSigma2', jitter, 'infer_params', 'covariance');
```

The result from this and other six models are shown in Figure 6.5.

6.4 Product of covariance functions

A product of two or more covariance functions is a valid covariance function as well. Such constructions may be usefull in situations where the phenomenon is known to be separable, such as spatiotemporal modeling for example. Combining covariance functions into product form $k_1(\mathbf{x}, \mathbf{x}') \cdot k_2(\mathbf{x}, \mathbf{x}')$... is very straightforward in GPstuff. There is a special covariance function `gpcf_prod` for this purpose. For example, multipying exponential and Mátern covariance functions is done as follows.

```
gpcf1 = gpcf_exp('init');
gpcf2 = gpcf_matern32('init');

gpcf = gpcf_prod('init', 'functions', {gpcf1, gpcf2});
```

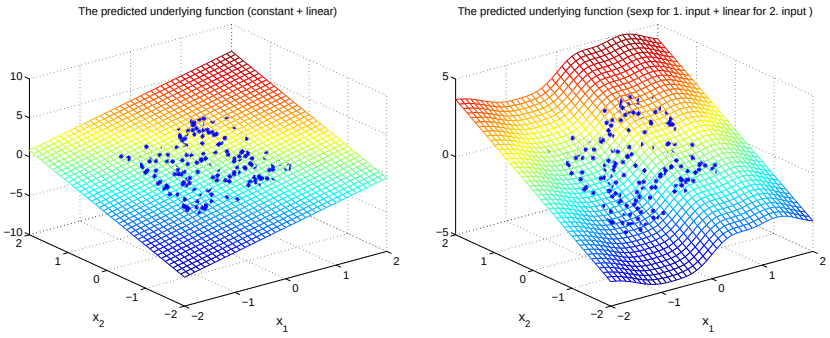
Above we first initilized the two functions to be multiplied and in the third line we constructed a covariance function structure which handles the actual multiplication.

The product covariance can be combined with the metric structures also. For example, if we want to model a temporal component with one covariance function and the spatial component with other we can construct the covariance function as follows.

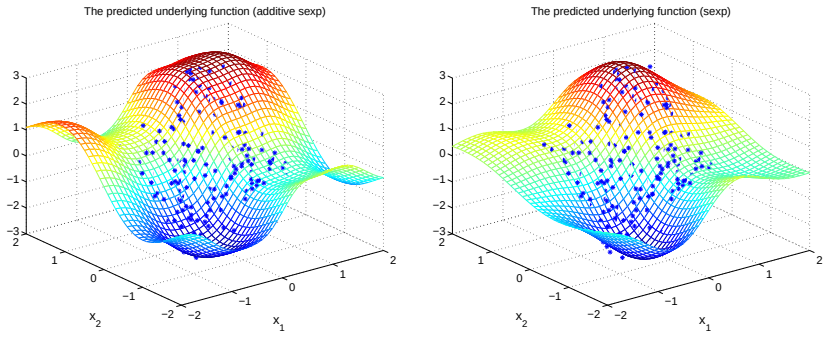
```
metric1 = metric_euclidean('init', {[1]});
metric2 = metric_euclidean('init', {[2 3]});

gpcf1 = gpcf_exp('init', 'metric', metric1);
gpcf2 = gpcf_matern32('init', 'metric', metric2);

gpcf = gpcf_prod('init', 'functions', {gpcf1, gpcf2});
```

The above construction represents the covariance function

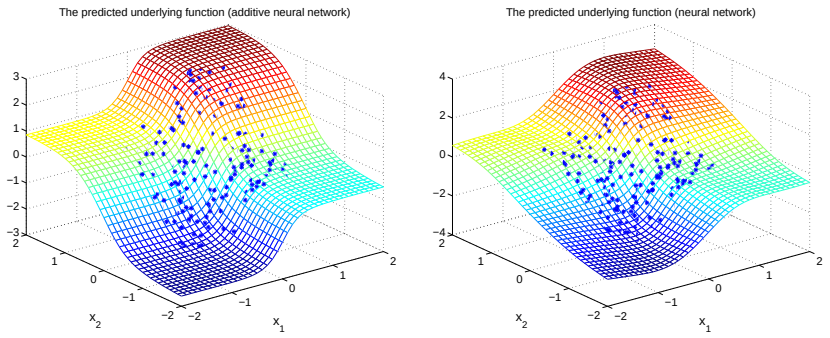
$$k(\mathbf{x}, \mathbf{x}') = k_{\text{exp}}(x_1, x_1') \cdot k_{\text{matern32}}([x_1, x_2]^{\text{T}}, [x_1', x_2']^{\text{T}}) \tag{6.6}$$



(a) $c + k_{\text{linear}}(\mathbf{x}, \mathbf{x}' \,|\theta)$ (b) $k_{\text{se}}(x_1, x_1'|\theta_1) + k_{\text{linear}}(x_2, x_2'|\theta_2)$



(c) $k_{\text{se}}(x_1, x_1'|\theta_1) + k_{\text{se}}(x_2, x_2'|\theta_2)$ (d) $k_{\text{se}}(\mathbf{x}, \mathbf{x}' \,|\theta_1)$



(e) $k_{\text{nn}}(x_1, x_1'|\theta_1) + k_{\text{nn}}(x_2, x_2'|\theta_2)$ (f) $k_{\text{nn}}(\mathbf{x}, \mathbf{x}' \,|\theta_1)$

Figure 6.5: GP latent mean predictions (using a MAP estimate) with different additive and non-additive covariance functions. The 2D toy data is generated from an additive process.

Chapter 7

Special observation models

In this chapter we will introduce few more models that we are able to infer with GP-stuff. These models utilize different observation models than what has been considered this far.

7.1 Robust regression with Student-t likelihood

A commonly used observation model in the GP regression is the Gaussian distribution. This is convenient since the inference is analytically tractable up to the covariance function parameters. However, a known limitation with the Gaussian observation model is its non-robustness, due which outlying observations may significantly reduce the accuracy of the inference. A formal definition of robustness is given, for example, in terms of an outlier-prone observation model. The observation model is outlier-prone of an order n, if $p(f|y_1, ..., y_{n+1}) \to p(f|y_1, ..., y_n)$ as $y_{n+1} \to \infty$ (O'Hagan, 1979; West, 1984). That is, the effect of a single conflicting observation on the posterior becomes asymptotically negligible as the observation approaches infinity. This contrasts heavily with the Gaussian observation model where each observation influences the posterior no matter how far it is from the others. A well-known robust observation model is the Student-t distribution

$$\mathbf{y}\,|\nu, \sigma_t \sim \prod_{i=1}^{n} \frac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)\sqrt{\nu\pi}\sigma_t} \left(1 + \frac{(y_i - f_i)^2}{\nu\sigma_t^2}\right)^{-(\nu+1)/2} \tag{7.1}$$

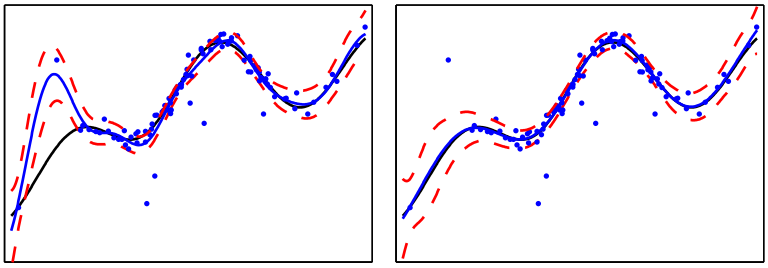
where ν is the degrees of freedom and σ the scale parameter (Gelman et al., 2004). Student-t distribution is outlier prone of order 1, and it can reject up to m outliers if there are at least $2m$ observations in all (O'Hagan, 1979).

The Student-t distribution can be utilized as such or it can be written via the scale mixture representation

$$y_i|f_i \sim N(f_i, \alpha U_i) \tag{7.2}$$

$$U_i \sim \text{Inv-}\chi^2(\nu, \tau^2) \tag{7.3}$$

where each observation has its own noise variance αU_i that is Inv-χ^2 distributed (Neal, 1997; Gelman et al., 2004). The degrees of freedom ν corresponds to the degrees of freedom in the Student-t distribution and $\alpha\tau$ corresponds to σ.



(a) Gaussian observation model. (b) Student-t observation model.

Figure 7.1: An example of regression with outliers. On the left Gaussian and on the right the Student-t observation model. The real function is plotted with black line.

In GPstuff both of the representations are implemented. The scale mixture representation can be inferred only with MCMC and the Student-t observation model with Laplace approximation and MCMC.

7.1.1 Rergression with Student-t distribution

Here we will discuss `demo_robustRegression`. The demo contains five parts: 1) Optimization approach with Gaussian noise, 2) MCMC approach with scale mixture noise model and all parameters sampled 3) Laplace approximation for Student-t likelihood optimizing all parameters, 4) MCMC approach with Student-t likelihood so that $\nu = 4$, and 5) Laplace approximation for Student-t observation model so that $\nu = 4$. We will demonstrate the steps for parts 2, 3 and 5.

The scale mixture model

The scale mixture representation of Student-t observation model is implemented as a special kind of covariance function `gpcf_noiset`. It is very similar to the `gpcf_noise` covariance function in that it returns diagonal covariance matrix $\mathrm{diag}(\alpha\mathbf{U})$. The scale mixture model is efficient to handle with Gibbs sampling since we are able to sample all the parameters (U_i, α, τ) efficiently from their full conditionals with regular built in samplers. For the degrees of freedom ν we use slice sampling. All the sampling steps are stored in the `gpcf_noiset` structure and `gp_mc` sampling function knows to use them if we add `gibbs_opt` field in its options structure.

Below we show the lines needed to perform the MCMC for the scale mixture model. The third line constructs the noise covariance function and the fourth line sets the option `fix_nu` to zero, which means that we are going to sample also the degrees of freedom. The degrees of freedom is many times poorly identifiable for which reason fixing its value to, for example, four is reasonable. This is the reason why it is fixed by default. With this data set its sampling is safe though. The next line initializes the GP structure and the lines after that set the sampling options. The structure `gibbs_opt` contains the options for the slice sampling used for ν. The parameters of the squared exponential covariance function are sampled with HMC and the options for this sampler are set into the structure `hmc_opt`.

```
gpcf1 = gpcf_sexp('init', 'lengthScale', 1, 'magnSigma2', 0.2^2);
```

```
gpcf1 = gpcf_sexp('set', gpcf1, 'lengthScale_prior', pl, 'magnSigma2_prior', pm);
gpcf2 = gpcf_noiset('init','nin', n, 'noiseSigmas2', repmat(1^2,n,1));

% Free nu
gpcf2 = gpcf_noiset('set', gpcf2, 'fix_nu', 0);

gp = gp_init('init', 'FULL', 'regr', {gpcf1}, {gpcf2}, 'jitterSigma2', 0.001) %

hmc_opt.steps=10;
hmc_opt.stepadj=0.08;
hmc_opt.nsamples=1;
hmc2('state', sum(100*clock));
hmc_opt.persistence=1;
hmc_opt.decay=0.6;

gibbs_opt = sls1mm_opt;
gibbs_opt.maxiter = 50;
gibbs_opt.mmlimits = [0 40];
gibbs_opt.method = 'minmax';

% Sample
[r,g,opt]=gp_mc(gp, x, y, 'nsamples', 300, 'hmc_opt', hmc_opt, 'gibbs_opt', gibbs_opt);
```

The Student-t observation model with Laplace approximation

The Student-t observation model is implemented in `likelih_t`. This is used similarly to the observation models in the classification setting. The difference is that now the likelihood has also hyperparameters. These parameters can be optimized alongside the covariance function parameters with Laplace approximation. We just need to give them prior (which is by default log-uniform) and write in the parameter string, which defines the optimized parameters, 'likelihood'. All this is done with the following lines.

```
pl = prior_t('init');
pm = prior_t('init', 's2', 0.3);
gpcf1 = gpcf_sexp('init', 'lengthScale', 1, 'magnSigma2', 0.2^2);
gpcf1 = gpcf_sexp('set', gpcf1, 'lengthScale_prior', pl, 'magnSigma2_prior', pm);

% Create the likelihood structure
pll = prior_logunif('init');
likelih = likelih_t('init', 'nu', 4, 'sigma2', 5^2, 'sigma2_prior', pll, 'nu_prior', pll);
likelih = likelih_t('set', likelih, 'fix_nu', 0)

% Finally create the GP data structure
gp = gp_init('init', 'FULL', likelih, {gpcf1}, {}, 'jitterSigma2', 0.0001); %
gp = gp_init('set', gp, 'infer_params', 'covariance+likelihood');
gp = gp_init('set', gp, 'latent_method', {'Laplace', x, y});

% Find the MAP estimate
w=gp_pak(gp);
fe=str2fun('gpla_e');
fg=str2fun('gpla_g');
opt = scg2_opt;
opt.tolfun = 1e-3;
opt.tolx = 1e-3;
opt.display = 1;

w=gp_pak(gp);
w=scg2(fe, w, opt, fg, gp, x, y);
gp =gp_unpak(gp,w);

% Predictions to test points
[Ef, Varf] = la_pred(gp, x, y, xx');
```

The Student-t observation model with MCMC

When using MCMC for the Student-t observation model we need to define sampling options for covariance function parameters, latent variables and likelihood parameters. After this we can run `gp_mc` and predict as before. All theses steps are shown below.

```
pl = prior_t('init');
pm = prior_t('init', 's2', 0.3);
gpcf1 = gpcf_sexp('init', 'lengthScale', 1, 'magnSigma2', 0.2^2);
gpcf1 = gpcf_sexp('set', gpcf1, 'lengthScale_prior', pl, 'magnSigma2_prior', pm);


% Create the likelihood structure
pll = prior_logunif('init');
likelih = likelih_t('init', 'nu', 4, 'sigma2', 0.5^2, 'sigma2_prior', pll, 'nu_prior', []);

% ... Finally create the GP data structure
gp = gp_init('init', 'FULL', likelih, {gpcf1}, {}, 'jitterSigma2', 0.0001); %
gp = gp_init('set', gp, 'latent_method', {'MCMC', zeros(size(y))', @scaled_mh});
gp = gp_init('set', gp, 'infer_params' , 'covariance+likelihood');

% Set the parameters for MCMC...

% Covariance parameter-options
opt.hmc_opt.steps=5;
opt.hmc_opt.stepadj=0.02;
opt.hmc_opt.nsamples=1;

% Latent-options
opt.latent_opt.display=0;
opt.latent_opt.repeat = 10
opt.latent_opt.sample_latent_scale = 0.05

% Likelihood-options
opt.likelih_hmc_opt.steps=10;
opt.likelih_hmc_opt.stepadj=0.1;
opt.likelih_hmc_opt.nsamples=1;

% Sample
[rgp,g,opt]=gp_mc(gp, x, y, 'nsamples', 400, opt);
```

7.2 Models for spatial epidemiology

Spatial epidemiology concerns both describing and understanding the spatial variation in the disease risk in geographically referenced health data. One of the main classes of spatial epidemiological studies is disease mapping, where the aim is to describe the overall disease distribution on a map and, for example, highlight areas of elevated or lowered mortality or morbidity risk (e.g. Lawson, 2001; Richardson, 2003; Elliot et al., 2001). The spatially referenced health data may be point level, appointing to continuously varying co-ordinates and showing for example home residence of diseased people. More commonly, however, the data are an areal level, referring to a finite subregion of a space, as for example, county or country and telling the counts of diseased people in the area (e.g. Banerjee et al., 2004).

In this section we will consider two disease mapping models. One that utilizes Poisson likelihood and other that uses negative binomial likelihood. The models follow the general approach discussed, for example, by Best et al. (2005). The data are aggregated into areas A_i with co-ordinates $\mathbf{x} = [x_1, x_2]^{\mathrm{T}}$. The mortality/morbidity in an area A_i is modeled as a Poisson or negative Binomial with mean $e_i\mu_i$, where e_i is the standardized expected number of cases in the area A_i, and the μ_i is the relative risk, which is given a Gaussian process prior.

The standardized expected number of cases e_i can be any positive real number that defines the expected mortality/morbidity count for the i'th area. Common practice is to evaluate it following the idea of the directly standardized rate (e.g. Ahmad et al., 2000), where the rate in an area is standardized according to the age distribution of the population in that area. The expected value in the area A_i is obtained by summing the

products of the rate and population over the age-groups in the area

$$e_i = \sum_{r=1}^{R} \frac{Y_r}{N_r} n_{ir},$$

where Y_r and N_r are the total number of deaths and people in the whole area of study in the age-group r, and n_{ir} is the number of people in the age-group r and in the area A_i. In the following demos e_i and $\mathbf{y}$ are calculated from real data that contains deaths to either alcohol related diseases or cerebral vascular diseases in Finland. The examples here are based on the works by Vanhatalo and Vehtari (2007) and Vanhatalo et al. (2010).

7.2.1 Disease mapping with Poisson likelihood: demo_spatial1

The model constructed in this section is the following:

$$\mathbf{y} \sim \prod_{i=1}^{n} \text{Poisson}(\exp(f_i)e_i) \tag{7.4}$$

$$f(\mathbf{x})|\theta \sim \mathcal{GP}\left(0, k(\mathbf{x}, \mathbf{x}'|\theta)\right) \tag{7.5}$$

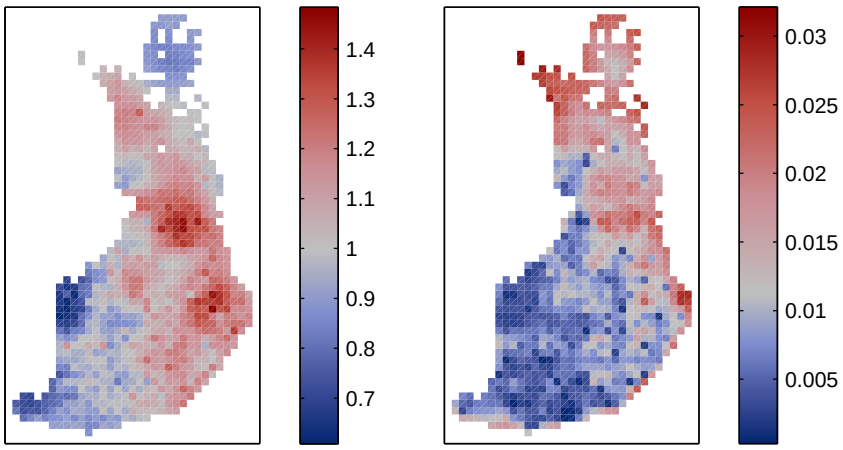
$$\theta \sim \text{half-Student-}t(\nu, \sigma_t^2) \tag{7.6}$$

The vector $\mathbf{y}$ collects the numbers of deaths for each area. The co-ordinates of the areas are in the input vectors $\mathbf{x}$ and θ contains the covariance function parameters. The co-ordinates are defined from lower left corner of the area in 20km steps. The model is constructed with the following lines.

```
gpcf1 = gpcf_matern32('init', 'lengthScale', 5, 'magnSigma2', 0.05);
pl = prior_t('init');
pm = prior_t('init', 's2', 0.3);
gpcf1 = gpcf_matern32('set', gpcf1, 'lengthScale_prior', pl, 'magnSigma2_prior', pm);

likelih = likelih_poisson('init');

gp = gp_init('init', 'FIC', likelih, {gpcf1}, [], 'X_u', Xu, 'jitterSigma2', 0.001);
gp = gp_init('set', gp, 'infer_params', 'covariance');

gp = gp_init('set', gp, 'latent_method', {'Laplace', xx, yy, 'z', ye});
```

We use FIC sparse approximation since the data set is rather large and inferring full GP would be too slow. The inducing inputs `Xu` are set to a regular grid (not shown here) in the two dimensional lattice and they will be considered fixed. The extra parameter `'z'` in the last line tells the Laplace algorithm that now there is an input which effects only the likelihood. This input is stored in `ye` and it is a vector of expected number of deaths $\mathbf{e} = [e_1, ..., e_n]^{\mathrm{T}}$. In all of the previous examples we have had only inputs for the covariance function. However, if there are inputs for likelihood they should be given with optional parameter-value pair whose indicator is `'z'`. The model is now constructed and we can optimize the hyperparameters and evaluate the posteorior predictive distribution of the latent variables.

```
w=gp_pak(gp);
fe=str2fun('gpla_e');
fg=str2fun('gpla_g');
opt = scg2_opt;
opt.tolfun = 1e-3;
opt.tolx = 1e-3;
opt.display = 1;
```



(a) The posterior mean. (b) The posterior variance.

Figure 7.2: The posterior predictive mean and variance of the latent function in the `demo_spatial1` data set obtained with FIC.

```
w=scg2(fe, w, opt, fg, gp, xx, yy, 'z', ye);
gp=gp_unpak(gp,w);

[Ef, Varf] = la_pred(gp, xx, yy, xx, 'z', ye, 'tstind', [1:n]);
```

Here we predicted to the same locations which were used in the training phace. Thus `Ef` and `Varf` contain the posterior mean and variance ($\mathrm{E}[\mathbf{f}\,|\hat{\theta}]$, $\mathrm{Var}[\mathbf{f}\,|\hat{\theta}]$). In this case, the prediction functions (such as `la_pred` for example) require the test index set for FIC also. This is given with parameter-value pair `'tstind', [1:n]`. These have previously been used with PIC (see section 4.4.2). FIC is a limiting case of PIC where each data point forms one block. Whenever we predict to new locations that have not been in the training set we do not have to worry about the test index set since all the test inputs define their own block. However, whenever we predict for exactly the same locations that are in the training set we should appoint the test inputs into the same block with the respective training input. This is done with FIC by giving `gp_pred` a vector with indices telling which of the test inputs are in the training set (`[1:n]` here). The posterior mean and variance of the latent variables are shown in the figure 7.2.

The demo contains also MCMC implementation for the model but it is not discussed here. Using Markov chain sampler for Poisson likelihood is very straightforward extension of its usage in classification model. The only difference is that we have to carry along the extra input $\mathbf{e}$.

7.2.2 Disease mapping with negative Binomial Likelihood

The Negative-Binomial distribution is a robust version of the Poisson distribution similarly as Student-t distribution can be considered a robustified Gaussian distribution

(Gelman et al., 2004). In this section we will demonstrate its usage. The model used is

$$\mathbf{y}\,|r \sim \prod_{i=1}^{n} \frac{\Gamma(r+y_i)}{y_i!\Gamma(r)} \left(\frac{r}{r+\mu_i}\right)^r \left(\frac{\mu_i}{r+\mu_i}\right)^{y_i} \tag{7.7}$$

$$f(\mathbf{x})|\theta \sim \mathcal{GP}\left(0, k(\mathbf{x}, \mathbf{x}'\,|\theta_g)\right), \tag{7.8}$$

$$\theta \sim \text{half-Student-}t(\nu, \sigma_t^2), \tag{7.9}$$

where $\mu_i = \exp(f(\mathbf{x}_i))$ and r is the dispersion parameter covering the variance. The data are simulated so that the latent function is drawn randomly from a GP with piece wise polynomial covariance function and the observed death cases are sampled from a Negative binomial distribution. This is done in order to demonstrate the usage of compactly supported covariance functions with non-Gaussian observation model. The CS covariance functions are used just as globally supported covariance functions but are much faster. The inference is conducted with Laplace approximation and EP. The code for Laplace approximation looks the following

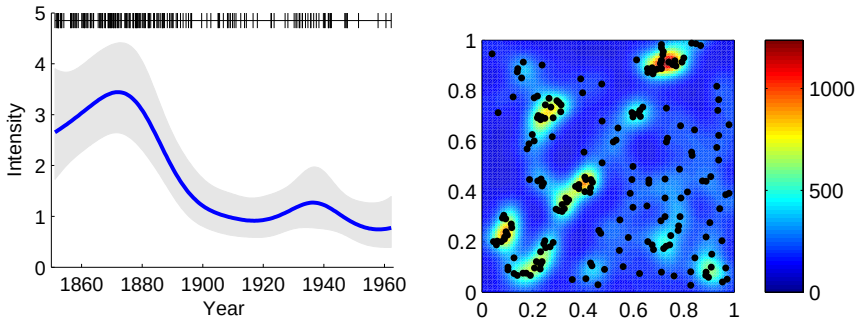
```
gpcf1 = gpcf_ppcs2('init', 'nin', 2, 'lengthScale', 5, 'magnSigma2', 0.05);
pl = prior_t('init');
pm = prior_t('init', 's2', 0.3);
gpcf1 = gpcf_ppcs2('set', gpcf1, 'lengthScale_prior', pl, 'magnSigma2_prior', pm);

% Create the likelihood structure
likelih = likelih_negbin('init');

% Create the GP data structure
gp = gp_init('init', 'FULL', likelih, {gpcf1}, [], 'jitterSigma2', 0.001);

% Set the approximate inference method to EP
gp = gp_init('set', gp, 'latent_method', {'Laplace', xx, yy, 'z', ye});

w=gp_pak(gp);
fe=str2fun('gpla_e');
fg=str2fun('gpla_g');

% set the options for scg2
opt = scg2_opt;
opt.tolfun = 1e-2;
opt.tolx = 1e-2;
opt.display = 1;

% do the optimization and set the optimized hyperparameter values back to the gp structure
w=scg2(fe, w, opt, fg, gp, xx, yy, 'z', ye);
gp = gp_unpak(gp,w);

C = gp_trcov(gp,xx);
nnz(C) / prod(size(C))
p = amd(C);
figure
spy(C(p,p))

% make prediction to the data points
[Ef, Varf] = la_pred(gp, xx, yy, xx, 'z', ye);
```

7.3 Log-Gaussian Cox process

Log-Gaussian Cox-process is an inhomogeneous Poisson process model used for point data, with unknown intensity function $\lambda(x)$ modeled with log-Gaussian process, so that, $g(x) = \log \lambda(x)$ (see Rathbun and Cressie, 1994; **?**).

If the data are points $X = x_i;\ i = 1, 2, \ldots, n$ on a finite region $\mathcal{V}$ in $\mathcal{X}$, then the likelihood of unknown function g is

$$L(X|g) = \exp\left\{-\left(\int_{\mathcal{V}} \exp(g(x))dx\right) + \sum_{i=1}^{K} g(x_i)\right\}. \tag{7.10}$$



(a) Coal mine disasters. (b) Red wood data.

Figure 7.3: Two intensity surfaces estimated with Log-Gaussian Cox process. The figures are from the `demo_lgcp`, where the aim is to study an underlying intensity surface of a point process. On the left a temporal and on the right a spatial point process.

Evaluation of the likelihood would require nontrivial integration over the exponential of the Gaussian process.

? propose to discretise the region $\mathcal{V}$ and assume locally constant intensity in subregions. This transforms the problem to a form equivalent to having Poisson model for each sub-region and likelihood after discretisation is

$$L(X|g) \approx \prod_{i=1}^{n} \text{Poisson}(\exp(g(x_i))). \tag{7.11}$$

? proved the posterior consistency in limit when sizes of subregions go to zero.

1D case data are the coal mine disaster data from R distribution (coal.rda) contains the dates of 191 coal mine explosions that killed ten or more men in Britain between 15 March 1851 and 22 March 1962. Computation time with expectation propagation and CCD integartion over the hyperparameters took 20s.

2D case data are the redwood data from R distribution (redwoodfull.rda) contains 195 locations of redwood trees. Computation time with Laplace approximation and MAP II for hyperparameters took 3s.

In section 7.2.1, we demonstrated how fast no-MCMC inference for this model can be made using Laplace method or expectation propagation for integrating over the latent variables in application from spatial epidemiology. The Log-Gaussian Cox process with the same techniques is implemented in the function `lgcp` for one or two dimensional input data. The usage of the function is demonstrated in `demo_lgcp`. This demo analyzes two data sets. The first one is one dimensional case data with coal mine disasters from R distribution. The data contain the dates of 191 coal mine explosions that killed ten or more men in Britain between 15 March 1851 and 22 March 1962. The analysis is conducted using expectation propagation and CCD integration over the hyperparameters and results are shown in Figure 7.3. The second data are the redwood data from R distribution. This data contain 195 locations of redwood trees in two dimensional lattice. The smoothed intensity surface is shown in Figure 7.3.

7.4 Binomial likelihood

In this demo (`demo_binomial`) we show how binomial likelihood is used in the GPstuff toolbox. The inference is done in this example with Laplace approximation and squared exponential covariance function.

The binomial likelihood is defined as follows:

$$p(\mathbf{y} \mid \mathbf{f}, \mathbf{z}) = \prod_{i=1}^{N} \frac{z_i!}{y_i!(z_i - y_i)!} p_i^{y_i} (1 - p_i)^{(z_i - y_i)} \tag{7.12}$$

where $p_i = \exp(f_i)/(1 + \exp(f_i))$ is the success of probability, and the vector $\mathbf{z}$ denotes the number of trials. In this demo, a Gaussian process prior is assumed for the latent variables $\mathbf{f}$.

The binomial likelihood is initialised in GPstuff as

```
% Create the likelihood structure
likelih = likelih_binomial('init');

% Create the GP data structure
gp = gp_init('init', 'FULL', likelih, {gpcf1}, [], 'jitterSigma2', 1e-3, 'infer_params', 'covariance');
```

To use binomial model, an extra parameter (the number of trials) is needed to be set as a parameter for each function that requires the data $\mathbf{y}$. For example, the model is initialized and optimized as

```
% Set the approximate inference method
gp = gp_init('set', gp, 'latent_method', {'Laplace', x, y, 'z', N});

[wopt, opt, flog]=scg2(fe, w, opt_scg, fg, gp, x, y, 'z', N);
```

To make predictions with binomial likelihood model without computing the predictive density, the total number of trials N_t in test points needs to be provided (in addition to N that is total number of trials in training points). In GPstuff, this is done as following:

```
% Set the total number of trials Nt at the grid points xgrid
[Ef_la, Varf_la, Ey_la, Vary_la] = la_pred(gp, x, y, xgrid, 'z', N, 'zt', Ntgrid);
```

To compute the predictive densities with binomial likelihood... **muista z**

```
% To compute predictive densities at the test points xt, the total number
% of trials Nt must be set additionally:
[Ef_la, Varf_la, Ey_la, Vary_la, py_la] = la_pred(gp, x, y, xt, 'z',
N, 'yt', yt, 'zt', Nt);
```

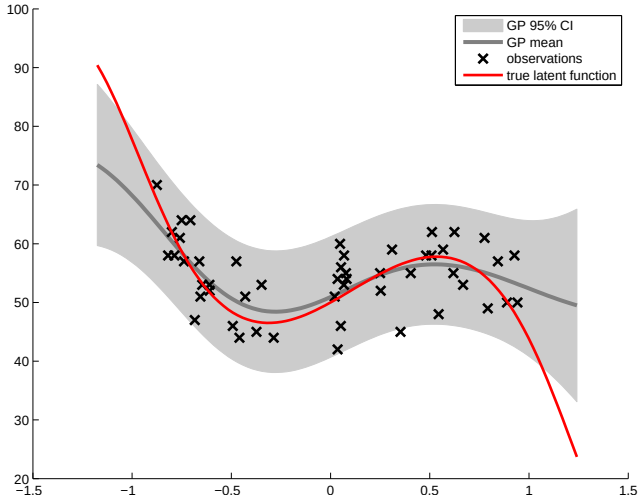


Figure 7.4: GP solutions (a MAP estimate) with squared exponential covariance function and binomial likelihood in a toy example.

Chapter 8

Summary

.1 Covariance functions

In this section we summarize all the covariance functions in the GPstuff package.

Squared exponential covariance function (`gpcf_sexp`)

Probably the most widely-used covariance function is the squared exponential (SE)

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2_{\text{sexp}} \exp\left(-\frac{1}{2}\sum_{k=1}^{d}\frac{(x_{i,k}-x_{j,k})^2}{l_k^2}\right). \tag{1}$$

The length scale l_k governs the correlation scale in input dimension k and the magnitude σ^2_{sexp} the overall variability of the process. A squared exponential covariance function leads to very smooth GPs that are infinitely times mean square differentiable.

Exponential covariance function (`gpcf_exp`)

Exponential covariance function is defined as

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2_{\text{exp}} \exp\left(-\sqrt{\sum_{k=1}^{d}\frac{(x_{i,k}-x_{j,k})^2}{l_k^2}}\right). \tag{2}$$

The parameters l_k and σ^2_{exp} have similar role as with the SE covariance function. The exponential covariance function leads to very rough GPs that are not mean square differentiable.

Mátern class of covariance functions (`gpcf_maternXX`)

The Matern class of covariance functions is given by

$$k_\nu(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2_{\text{m}}\frac{2^{1-\nu}}{\Gamma(\nu)}\left(\sqrt{2\nu}r\right)^\nu K_\nu\left(\sqrt{2\nu}r\right), \tag{3}$$

where $r = \left(\sum_{k=1}^{d}\frac{(x_{i,k}-x_{j,k})^2}{l_k^2}\right)^{1/2}$. The parameter ν governs the smoothness of the process, and K_ν is a modified Bessel function (Abramowitz and Stegun, 1970, sec. 9.6). The Matern covariance functions can be represent in a simpler form when ν is a half integer. The Mátern covariance functions with $\nu = 3/2$ (`gpcf_matern32`) and $\nu = 5/2$ (`gpcf_matern52`) are:

$$k_{\nu=3/2}(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2_{\text{m}}\left(1+\sqrt{3}r\right)\exp\left(-\sqrt{3}r\right) \tag{4}$$

$$k_{\nu=5/2}(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2_{\text{m}}\left(1+\sqrt{5}r+\frac{5r^2}{3}\right)\exp\left(-\sqrt{5}r\right). \tag{5}$$

Neural network covariance function (`gpcf_neuralnetwork`)

A neural network with suitable transfer function and prior distribution converges to a GP as the number of hidden units in the network approaches to infinity (Neal, 1996;

Williams, 1996; Rasmussen and Williams, 2006). A nonstationary neural network covariance function is

$$k(\mathbf{x}_i, \mathbf{x}_j) = \frac{2}{\pi} \sin^{-1}\left(\frac{2\tilde{\mathbf{x}}_i^{\mathrm{T}} \Sigma \tilde{\mathbf{x}}_j}{(1 + 2\tilde{\mathbf{x}}_i^{\mathrm{T}} \Sigma \tilde{\mathbf{x}}_i)(1 + 2\tilde{\mathbf{x}}_j^{\mathrm{T}} \Sigma \tilde{\mathbf{x}}_j)}\right), \tag{6}$$

where $\tilde{\mathbf{x}} = (1, x_1, \ldots, x_d)^{\mathrm{T}}$ is an input vector augmented with 1. $\Sigma = \mathrm{diag}(\sigma_0^2, \sigma_1^2, \ldots, \sigma_d^2)$ is a diagonal weight prior, where σ_0^2 is a variance for bias parameter controlling the functions offset from the origin. The variances for weight parameters are $\sigma_1^2, \ldots, \sigma_d^2$, and with small values for weights, the neural network covariance function produces smooth and rigid looking functions. The larger values for weight variances produces more flexible solutions.

Constant covariance function (`gpcf_constant`)

Perhaps the simplest covariance function is the constant covariance function

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2 \tag{7}$$

with variance hyperparameter σ^2. This function can be used to implement the constant term in the dotproduct covariance function (Rasmussen and Williams, 2006) reviewed below.

Linear covariance function (`gpcf_linear`)

The linear covariance function is

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^{\mathrm{T}} \Sigma \mathbf{x}_j \tag{8}$$

where the diagonal matrix $\Sigma = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_D^2)$ contains the prior variances of the linear model coefficients. Combining this with the constant function above we can form the dotproduct covariance function (Rasmussen and Williams, 2006)

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2 + \mathbf{x}_i^{\mathrm{T}} \Sigma \mathbf{x}_j. \tag{9}$$

Piecewise polynomial functions (`gpcf_ppcsX`)

The piecewise polynomial functions are the only compactly supported covariance functions (see section 4) in GPstuff. There are four of them with the following forms

$$k_{pp0}(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2 (1 - r)_+^j \tag{10}$$

$$k_{pp1}(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2 (1 - r)_+^{j+1} \left((j + 1)r + 1\right) \tag{11}$$

$$k_{pp2}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sigma^2}{3} (1 - r)_+^{j+2} ((j^2 + 4j + 3)r^2 + (3j + 6)r + 3) \tag{12}$$

$$\begin{aligned} k_{pp3}(\mathbf{x}_i, \mathbf{x}_j) = &\frac{\sigma^2}{15} (1 - r)_+^{j+3} ((j^3 + 9j^2 + 23j + 15)r^3 + \\ &(6j^2 + 36j + 45)r^2 + (15j + 45)r + 15) \end{aligned} \tag{13}$$

where $j = \lfloor d/2 \rfloor + q + 1$. These functions correspond to processes that are $2q$ times mean square differentiable at the zero and positive definite up to the dimension d (Wendland, 2005). The covariance functions are named `gpcf_ppcs0`, `gpcf_ppcs1`, `gpcf_ppcs2`, and `gpcf_ppcs3`.

Rational quadratic covariance function (`gpcf_rq`)

The rational quadratic (RQ) covariance function (Rasmussen and Williams, 2006)

$$k_{\mathrm{RQ}}(\mathbf{x}_i, \mathbf{x}_j) = \left(1 + \frac{1}{2\alpha}\sum_{k=1}^{d}\frac{(x_{i,k} - x_{j,k})^2}{l_k^2}\right)^{-\alpha} \tag{14}$$

can be seen as a scale mixture of squared exponential covariance functions with different length scales. The smaller the parameter $\alpha > 0$ is the more diffusive the length scales of the mixing components are. The parameter $l_k > 0$ characterizes the typical length scale of the individual components in input dimension k.

Periodic covariance function (`gpcf_periodic`)

Many real world systems exhibit periodic phenomena, which can be modelled with a periodic covariance function. One possible construction (Rasmussen and Williams, 2006) is

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\sum_{k=1}^{d}\frac{2\sin^2(\pi(x_{i,k} - x_{j,k})/\gamma)}{l_k^2}\right), \tag{15}$$

where the parameter γ controls the inverse length of the peridiocity and l_k the smoothness of the process in dimension k.

Product covariance function (`gpcf_product`)

A product of two or more covariance functions, $k_1(\mathbf{x}, \mathbf{x}') \cdot k_2(\mathbf{x}, \mathbf{x}')...$, is a valid covariance function as well. Combining covariance functions into product form is very straightforward with `gpcf_prod`, for which the user can freely specify the covariance functions to be multiplied with each other from the collection of covariance functions implemented in GPstuff.

.2 Observation models

Here, we summarize all the observation models in GPstuff. Most of them are implemented in files `likelih_*` which reminds that at the inference step they transform to likelihood function.

Gaussian (`gpcf_noise`)

The i.i.d Gaussian noise with variance σ^2 is

$$\mathbf{y} \,|\, \mathbf{f}, \sigma^2 \sim N(\mathbf{f}, \sigma^2\mathbf{I}). \tag{16}$$

Student-t (`likelih_t`, `gpcf_noiset`)

The Student-t observation model (implemented in `likelih_t`) is

$$\mathbf{y} \,|\, \mathbf{f}, \nu, \sigma_t \sim \prod_{i=1}^{n}\frac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)\sqrt{\nu\pi}\sigma_t}\left(1 + \frac{(y_i - f_i)^2}{\nu\sigma_t^2}\right)^{-(\nu+1)/2}, \tag{17}$$

where ν is the degrees of freedom and σ the scale parameter. The scale mixture version of the Student-t distribution is implemented in `gpcf_noiset` and it is parametrised as

$$y_i|f_i, \alpha, U_i \sim N(f_i, \alpha U_i) \tag{18}$$
$$U_i \sim \text{Inv-}\chi^2(\nu, \tau^2), \tag{19}$$

where each observation has its own noise variance αU_i (Neal, 1997; Gelman et al., 2004).

Logit (`likelih_logit`)

The logit transformation gives the probability for y_i of being 1 or -1 as

$$p_{\text{logit}}(y_i|f(\mathbf{x}_i)) = \frac{1}{1+\exp(-y_i f(\mathbf{x}_i))}. \tag{20}$$

Probit (`likelih_probit`)

The probit transformation gives the probability for y_i of being 1 or -1 as

$$p_{\text{probit}}(y_i|f(\mathbf{x}_i)) = \Phi(y_i f(\mathbf{x}_i)) = \int_{-\infty}^{y_i f(\mathbf{x}_i)} N(z|0,1)dz. \tag{21}$$

Poisson (`likelih_poisson`)

The Poisson observation model with expected number of cases $\mathbf{e}$ is

$$\mathbf{y} \mid \mathbf{f}, \mathbf{e} \sim \prod_{i=1}^{n} \text{Poisson}(y_i|\exp(f_i)e_i). \tag{22}$$

Negative-Binomial (`likelih_negbin`)

The negative-binomial is a robustified version of the Poisson distribution. It is parametrized

$$\mathbf{y} \mid \mathbf{f}, \mathbf{e}, r \sim \prod_{i=1}^{n} \frac{\Gamma(r+y_i)}{y_i!\Gamma(r)} \left(\frac{r}{r+\mu_i}\right)^r \left(\frac{\mu_i}{r+\mu_i}\right)^{y_i} \tag{23}$$

where $\mu_i = \mathbf{e}\exp(f(\mathbf{x}_i))$, r is the dispersion parameter governing the variance, e_i is the expected number of cases and y_i is positive integer telling the observed count.

Binomial (`likelih_binomial`)

The binomial observation model with succes probability $p_i = \exp(f(\mathbf{x}_i))/(1+\exp(f(\mathbf{x}_i)))$ is

$$\mathbf{y} \mid \mathbf{f}, \mathbf{z} \sim \prod_{i=1}^{N} \frac{z_i!}{y_i!(z_i-y_i)!} p_i^{y_i}(1-p_i)^{(z_i-y_i)}. \tag{24}$$

Here, z_i denotes the number of trials and y_i is the number of successes.

.3 Hyperpriors

This section lists all the hyperpriors implemented in the GPstuff package.

Normal prior (`prior_normal`)

The normal distribution is parametrised as

$$p(\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(\theta-\mu)^2\right) \tag{25}$$

where μ is a location parameter and σ^2 is a scale parameter.

Log-normal prior (`prior_lognormal`)

The log-normal distribution is parametrised as

$$p(\theta) = \frac{1}{\theta\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(\log(\theta)-\mu)^2\right) \tag{26}$$

where μ is a location parameter and σ^2 is a scale parameter.

Laplace prior (`prior_laplace`)

The Laplace distribution is parametrised as

$$p(\theta) = \frac{1}{2\sigma} \exp\left(-\frac{|\theta-\mu|}{\sigma}\right) \tag{27}$$

where μ is a location parameter and $\sigma > 0$ is a scale parameter.

Student-t prior (`prior_t`)

The Student-t distribution is parametrised as

$$p(\theta) = \frac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)\sqrt{\nu\pi\sigma^2}} \left(1 + \frac{(\theta-\mu)^2}{\nu\sigma^2}\right)^{-(\nu+1)/2} \tag{28}$$

where μ is a location parameter, σ^2 is a scale parameter and $\nu > 0$ is the degrees of freedom.

Scaled inverse-chi-square prior (`prior_sinvchi2`)

The scaled inverse-chi-square distribution is parametrised as

$$p(\theta) = \frac{(\nu/2)^{\nu/2}}{\Gamma(\nu/2)} (s^2)^{\nu/2} \theta^{-(\nu/2+1)} e^{-\nu s^2/(2\theta)} \tag{29}$$

where s^2 is a scale parameter and $\nu > 0$ is a degrees of freedom parameter.

Gamma prior (`prior_gamma`)

The gamma distribution is parametrised as

$$p(\theta) = \frac{\beta^\alpha}{\Gamma(\alpha)}\theta^{\alpha-1}e^{-\beta\theta} \tag{30}$$

where $\alpha > 0$ is a shape parameter and $\beta > 0$ is an inverse scale parameter.

Inverse-gamma prior (`prior_invgam`)

The inverse-gamma distribution is parametrised as

$$p(\theta) = \frac{\beta^\alpha}{\Gamma(\alpha)}\theta^{-(\alpha+1)}e^{-\beta/\theta} \tag{31}$$

where $\alpha > 0$ is a shape parameter and $\beta > 0$ is a scale parameter.

Uniform prior (`prior_unif`)

The uniform prior is parametrised as

$$p(\theta) \propto 1. \tag{32}$$

Log-uniform prior (`prior_logunif`)

The log-uniform prior is parametrised as

$$p(\log(\theta)) \propto 1. \tag{33}$$

Log-log-uniform prior (`prior_loglogunif`)

The log-log-uniform prior is parametrised as

$$p(\log(\log(\theta))) \propto 1. \tag{34}$$

Appendix A

Log transformation of hyperparameters

The inference on the parameters of covariance functions is conducted in the log space, which has the advantage that the parameter space $(0, \infty)$ is transformed into $(-\infty, \infty)$. The change of parameterization has to be taken into account in the evaluation of the probability densities of the model. If parameter θ with probability density $p_\theta(\theta)$ is transformed into the parameter $w = f(\theta)$ with equal number of components, the probability density of w is given by

$$p_w(w) = |J| p_\theta(f^{-1}(w)), \tag{A.1}$$

where J is the Jacobian of the transformation $\theta = f^{-1}(w)$. The parameter transformations are discussed shortly, for example, in Gelman et al. (2004)[p. 24].

Due to the log transformation $w = \log(\theta)$ trasformation the propability densities $p_\theta(\theta)$ are changed to the densities

$$p_w(w) = |J| p_\theta(\exp(w)) = |J| p_\theta(\theta), \tag{A.2}$$

where the Jacobian is $J = \frac{\partial \exp(w)}{\partial w} = \exp(w) = \theta$. Now, in regression model (see section **??**) the posterior of w can be written as

$$p_w(w|\mathcal{D}) \propto p(\mathbf{y}|\mathbf{X}, \theta) p(\theta|\gamma)\theta, \tag{A.3}$$

which leads to energy function

$$\begin{aligned} E(w) &= -\log p(\mathbf{y}|\mathbf{X}, \theta) - \log p(\theta|\gamma) - \log(|\theta|). \\ &= E(\theta) - \log(\theta), \end{aligned}$$

where the absolut value signs are not shown explicitly around θ because it is stricly positive. Thus, the log transformation just adds term $-\log\theta$ in the energy function.

The inference on w requires also the gradients of an energy function $E(w)$. These can be obtained easily with the chain rule

$$\begin{aligned}\frac{\partial E(w)}{\partial w} &= \frac{\partial}{\partial \theta}\left[E(\theta) - \log(|J|)\right]\frac{\partial \theta}{\partial w} \\ &= \left[\frac{\partial E(\theta)}{\partial \theta} - \frac{\partial \log(|J|)}{\partial \theta}\right]\frac{\partial \theta}{\partial w} \\ &= \left[\frac{\partial E(\theta)}{\partial \theta} - \frac{1}{|J|}\frac{\partial |J|}{\partial \theta}\right] J. \end{aligned} \tag{A.4}$$

Here we have used the fact that the last term, derivative of θ with respect to w, is the same as the Jacobian $J = \frac{\partial \theta}{\partial w} = \frac{\partial f^{-1}}{\partial w}$. Now in the case of log transformation the Jacobian can be replaced by θ and the gradient is gotten an easy expression

$$\frac{\partial E(w)}{\partial w} = \frac{\partial E(\theta)}{\partial \theta}\theta - 1. \tag{A.5}$$

Bibliography

Abramowitz, M. and Stegun, I. A. (1970). *Handbook of mathematical functions*. Dover Publications, Inc.

Ahmad, O. B., Boschi-Pinto, C., Lopez, A. D., Murray, C. J., Lozano, R., and Inoue, M. (2000). Age standardization of rates: A new WHO standard. *GPE Discussion Paper Series*, 31.

Alvarez, M. A., Luengo, D., Titsias, M. K., and Lawrence, N. D. (2010). Efficient multioutput Gaussian processes through variational inducing kernels. *JMLR Workshop and conference proceedings*, 9:25–32.

Banerjee, S., Carlin, B. P., and Gelfand, A. E. (2004). *Hierarchical Modelling and Analysis for Spatial Data*. Chapman Hall/CRC.

Banerjee, S., Gelfand, A. E., Finley, A. O., and Sang, H. (2008). Gaussian predictive process models for large spatial data sets. *Journal of the Royal Statistical Society B*, 70(4):825–848.

Bernardo, J. M. (1979). Expected information as expected utility. *Annals of Statistics*, 7(3):686–690.

Bernardo, J. M. and Smith, A. F. M. (2000). *Bayesian Theory*. John Wiley & Sons, Ltd.

Best, N., Richardson, S., and Thomson, A. (2005). A comparison of Bayesian spatial models for disease mapping. *Statistical Methods in Medical Research*, 14:35–59.

Buhmann, M. D. (2001). A new class of radial basis functions with compact support. *Mathematics of Computation*, 70(233):307–318.

Burman, P. (1989). A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods. *Biometrika*, 76(3):503–514.

Christensen, O. F., Roberts, G. O., and Sköld, M. (2006). Robust Markov chain Monte Carlo methods for spatial generalized linear mixed models. *Journal of Computational and Graphical Statistics*, 15:1–17.

Cornford, D., Csató, L., and Opper, M. (2005). Sequential, Bayesian geostatistics: A principled method for large data sets. *Geographical Analysis*, 37:183–199.

Cressie, N. A. C. (1993). *Statistics for Spatial Data*. John Wiley & Sons, Inc.

Csató, L. and Opper, M. (2002). Sparse online Gaussian processes. *Neural Computation*, 14(3):641–669.

Davis, T. A. (2006). *Direct Methods for Sparse Linear Systems*. SIAM.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924.

Diggle, P. J. and Ribeiro, P. J. (2007). *Model-based Geostatistics*. Springer Science+Business Media, LLC.

Diggle, P. J., Tawn, J. A., and Moyeed, R. A. (1998). Model-based geostatistics. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 47(3):299–350.

Duane, S., Kennedy, A., Pendleton, B. J., and Roweth, D. (1987). Hybrid Monte Carlo. *Physics Letters B*, 195(2):216–222.

Elliot, P., Wakefield, J., Best, N., and Briggs, D., editors (2001). *Spatial Epidemiology Methods and Applications*. Oxford University Press.

Finkenstädt, B., Held, L., and Isham, V. (2007). *Statistical Methods for Spatio-Temporal Systems*. Chapman & Hall/CRC.

Furrer, R., Genton, M. G., and Nychka, D. (2006). Covariance tapering for interpolation of large spatial datasets. *Journal of Computational and Graphical Statistics*, 15(3):502–523.

Gaspari, G. and Cohn, S. (1999). Construction of correlation functions in two and three dimensions. *Quarterly Journal of the Royal Meteorological Society*, 125(554):723–757.

Gelfand, A. E., Diggle, P. J., Fuentes, M., and Guttorp, P. (2010). *Handbook of Spatial Statistics*. CRC Press.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2004). *Bayesian Data Analysis*. Chapman & Hall/CRC, second edition.

Geweke, J. (1989). Bayesian inference in econometric models using Monte Carlo integration. *Econometrica*, 57(6):721–741.

Gibbs, M. N. and Mackay, D. J. C. (2000). Variational Gaussian process classifiers. *IEEE Transactions on Neural Networks*, 11(6):1458–1464.

Gilks, W., Richardson, S., and Spiegelhalter, D. (1996). *Markov Chain Monte Carlo in Practice*. Chapman & Hall.

Gneiting, T. (1999). Correlation functions for atmospheric data analysis. *Quarterly Journal of the Royal Meteorological Society*, 125:2449–2464.

Gneiting, T. (2002). Compactly supported correlation functions. *Journal of Multivariate Analysis*, 83:493–508.

Goel, P. K. and Degroot, M. H. (1981). Information about hyperparamters in hierarchical models. *Journal of the American Statistical Association*, 76(373):140–147.

Good, I. J. (1952). Rational decisions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 14(1):107–114.

Grewal, M. S. and Andrews, A. P. (2001). *Kalman Filtering: Theory and Practice Using Matlab*. Wiley Interscience, second edition.

Harville, D. A. (1997). *Matrix Algebra From a Statistician's Perspective*. Springer-Verlag.

Kass, R. E. and Raftery, A. E. (1995). Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795.

Kaufman, C. G., Schervish, M. J., and Nychka, D. W. (2008). Covariance tapering for likelihood-based estimation in large spatial data sets. *Journal of the American Statistical Association*, 103(484):1545–1555.

Kuss, M. and Rasmussen, C. E. (2005). Assessing approximate inference for binary Gaussian process classification. *Journal of Machine Learning Research*, 6:1679–1704.

Lawrence, N. (2007). Learning for larger datasets with the Gaussian process latent variable model. In Meila, M. and Shen, X., editors, *Proceedings of the Eleventh International Workshop on Artificial Intelligence and Statistics*. Omnipress.

Lawson, A. B. (2001). *Statistical Methods in Spatial Epidemology*. John Wiley & Sons, Ltd.

Martino, S. (2007). *Approximate Bayesian Inference for Latent Gaussian Models*. PhD thesis, Norwegian University of Science and Technology.

Matheron, G. (1973). The intrinsic random functions and their applications. *Advances in Applied Probability*, 5(3):439–468.

Minka, T. (2001). *A Family of Algorithms for Approximate Bayesian Inference*. PhD thesis, Massachusetts Institute of Technology.

Moreaux, G. (2008). Compactly supported radial covariance functions. *Journal of Geodecy*, 82(7):431–443.

Neal, R. (1998). Regression and classification using Gaussian process priors. In Bernardo, J. M., Berger, J. O., David, A. P., and Smith, A. P. M., editors, *Bayesian Statistics 6*, pages 475–501. Oxford University Press.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer.

Neal, R. M. (1997). Monte Carlo Implementation of Gaussian Process Models for Bayesian Regression and Classification. Technical Report 9702, Dept. of statistics and Dept. of Computer Science, University of Toronto.

Nickisch, H. and Rasmussen, C. E. (2008). Approximations for binary Gaussian process classification. *Journal of Machine Learning Research*, 9:2035–2078.

O'Hagan, A. (1978). Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society. Series B.*, 40(1):1–42.

O'Hagan, A. (1979). On outlier rejection phenomena in Bayes inference. *Journal of the Royal Statistical Society. Series B.*, 41(3):358–367.

O'Hagan, A. (2004). Dicing with the unknown. *Significance*, 1:132–133.

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6(3):1939–1959.

Rasmussen, C. E. (1996). *Evaluations of Gaussian Processes and Other Methods for Non-linear Regression*. PhD thesis, University of Toronto.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

Rathbun, S. L. and Cressie, N. (1994). Asymptotic properties of estimators for the parameters of spatial inhomogeneous poisson point processes. *Advances in Applied Probability*, 26(1):122–154.

Richardson, S. (2003). Spatial models in epidemiological applications. In Green, P. J., Hjort, N. L., and Richardson, S., editors, *Highly Structured Stochastic Systems*, pages 237–259. Oxford University Press.

Robert, C. P. and Casella, G. (2004). *Monte Carlo Statistical Methods*. Springer, second edition.

Rue, H., Martino, S., and Chopin, N. (2009). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *Journal of the Royal statistical Society B*, 71(2):1–35.

Sanchez, S. M. and Sanchez, P. J. (2005). Very large fractional factorials and central composite designs. *ACM Transactions on Modeling and Computer Simulation*, 15:362–377.

Sansò, F. and Schuh, W.-D. (1987). Finite covariance functions. *Journal of Geodesy*, 61(4):331–347.

Seeger, M. (2005). Expectation propagation for exponential families. Technical report, Max Planck Institute for Biological Cybernetics, Tübingen, Germany.

Seeger, M., Williams, C. K. I., and Lawrence, N. (2003). Fast forward selection to speed up sparse Gaussian process regression. In Bishop, C. M. and Frey, B. J., editors, *Ninth International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics.

Snelson, E. (2007). *Flexible and Efficient Gaussian Process Models for Machine Learning*. PhD thesis, University College London.

Snelson, E. and Ghahramani, Z. (2006). Sparse Gaussian process using pseudo-inputs. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Advances in Neural Information Processing Systems 18*. The MIT Press.

Snelson, E. and Ghahramani, Z. (2007). Local and global sparse Gaussian process approximations. In Meila, M. and Shen, X., editors, *Artificial Intelligence and Statistics 11*. Omnipress.

Spiegelhalter, D. J., Best, N. G., Carlin, B. P., and van der Linde, A. (2002). Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society B*, 64(4):583–639.

Storkey, A. (1999). *Efficient Covariance Matrix Methods for Bayesian Gaussian Processes and Hopfield Neural Networks*. PhD thesis, University of London.

Sundararajan, S. and Keerthi, S. S. (2001). Predictive approaches for choosing hyperparameters in gaussian processes. *Neural Computation*, 13(5):1103–1118.

Takahashi, K., Fagan, J., and Chen, M.-S. (1973). Formation of a sparse bus impedance matrix and its application to short circuit study. In *Power Industry Computer Application Conference Proceedings*. IEEE Power Engineering Society.

Tierney, L. and Kadane, J. B. (1986). Accurate approximations for posterior moments and marginal densities. *Journal of the American Statistical Association*, 81(393):82–86.

Titsias, M. K. (2009). Variational learning of inducing variables in sparse Gaussian processes. *JMLR Workshop and Conference Proceedings*, 5:567–574.

Vanhatalo, J., Pietiläinen, V., and Vehtari, A. (2010). Approximate inference for disease mapping with sparse gaussian processes. *Statistics in Medicine*, 29(15):1580–1607.

Vanhatalo, J. and Vehtari, A. (2007). Sparse log Gaussian processes via MCMC for spatial epidemiology. *JMLR Workshop and Conference Proceedings*, 1:73–89.

Vanhatalo, J. and Vehtari, A. (2008). Modelling local and global phenomena with sparse Gaussian processes. In McAllester, D. A. and Myllymäki, P., editors, *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pages 571–578.

Vanhatalo, J. and Vehtari, A. (2010). Speeding up the binary Gaussian process classification. In Grünwald, P. and Spirtes, P., editors, *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence*, pages 1–9.

Vehtari, A. (2001). *Bayesian Model Assessment and Selection Using Expected Utilities*. PhD thesis, Helsinki University of Technology.

Vehtari, A. and Lampinen, J. (2002). Bayesian model assessment and comparison using cross-validation predictive densities. *Neural Computation*, 14(10):2439–2468.

Wendland, H. (1995). Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics*, 4(1):389–396.

Wendland, H. (2005). *Scattered Data Approximation*. Cambridge University Press.

West, M. (1984). Outlier models and prior distributions in Bayesian linear regression. *Journal of the Royal Statistical Society. Series B.*, 46(3):431–439.

Wiener, N. (1949). *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. MIT Press.

Williams, C. K. I. (1996). Computing with infinite networks. In Mozer, M. C., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems*, volume 9. The MIT Press.

Williams, C. K. I. and Barber, D. (1998). Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351.

Williams, C. K. I. and Rasmussen, C. E. (1996). Gaussian processes for regression. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8*, pages 514–520.

Wu, Z. (1995). Compactly supported positive definite radial functions. *Advances in Computational Mathematics*, 4(1):283–292.