

A Factor Graph Approach to Gaussian Process Preference Learning

M.G. Schoonderbeek

Email: m.g.schoonderbeek@student.tue.nl

October 28, 2014

Abstract—Gaussian Process Preference Learning (GPPL) is considered to be the state-of-the-art algorithm for learning about a person’s preferences over a continuous domain of tunable parameter values. Unfortunately, the existing literature is vague about the underlying modular structure of the full algorithm. This hinders application of GPPL to embedded environments or to new application domains. We describe the GPPL algorithm as a Forney-style Factor Graph (FFG), which is a formal framework for distributed signal processing (and probabilistic inference) in a graph of sub-components. One of the benefits of the FFG-based specification of GPPL is that the algorithm can easily be changed to the specific circumstances of the application. We demonstrate the FFG-based GPPL algorithm in MATLAB by a perceptual tuning problem for a noise suppression algorithm.

Keywords - Machine Learning, Preference Learning, Gaussian Processes, Forney-style Factor Graphs, Hearing Aids

I. INTRODUCTION

The motivation for this thesis originates from the need for personalized tuning of hearing aid algorithm parameters by patients themselves. Today the settings of a hearing aid are adapted to the user’s need by specialists. Unfortunately these settings depend on the specialist’s interpretation of what the user explains he perceives. In order to bypass this interpretation, it will be useful that the user himself can tune the settings whenever a hearing problem occurs.

Let’s consider the following scenario. We assume that a subject’s preferences for a tunable parameter value θ of a hearing aid can be described by a function $\tilde{g}(\theta)$. Crucially, $\tilde{g}(\theta)$ is an ‘internal’ evaluation function inside the brain and is as such not directly observable. We can however estimate $\tilde{g}(\theta)$ by a function $g(\theta)$ through listening experiments with the subject. For instance, we can present two sound samples that were generated with different settings θ_1 and θ_2 , and ask which one is preferred by the listener. The preferred parameter value provides information about the shape of the underlying preference function. If we execute enough of these listening trials, the original preference function can be estimated and the best hearing aid setting can be found through maximization of the estimated preference function $g(\theta)$. In Figure 1 we show this type preference learning as an iterative loop of the following four essential steps:

BUILD - The Experiment Design (ED) step chooses a new experiment, which is in our case a certain parameter setting. In order to make an informed choice, this selected experiment will be based on

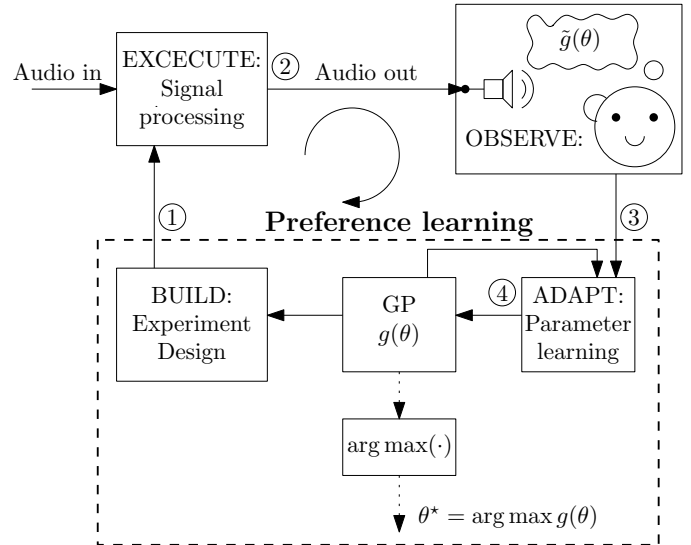


Figure 1. The preference learning loop consist of four steps, 1. Build, 2. Execute, 3. Observe and 4. Adapt, which iterates until the user is satisfied. The dashed box represents the incremental preference learning part.

both the mean and uncertainty estimates about the preference function.

EXECUTE - A signal processing algorithm of the hearing aid will produce an audio sample with this new parameter setting.

OBSERVE - The user will judge this audio sample based on his internal evaluation function. Via an interface this judgment is converted to an observation.

ADAPT - With this new observation and the prior knowledge of previous iterations the latent variables are adapted.

These BUILD-EXECUTE-OBSERVE-ADAPT loops are iterated until a stop criteria is met, eg, until a stable maximum of $g(\theta)$ has been found.

In this work, the estimated preference function will be described by a Gaussian Process (GP), [1]. In the machine learning literature, GP theory is a probabilistic framework that supports Bayesian inference. The probabilistic framework is used by the ED step, since ED needs both the mean and variance of the estimated preference function. The Adapt step needs to combine the prior knowledge with the new observed data, which will be done via Bayesian inference. Gaussian

Process based Preference Learning (GPPL) is considered to be the standard in preference learning, see [2], [3], [4] and [5].

An important motivation for the present work is that separate parts of the preference learning system can easily be adapted to different circumstances. For example, in the Observe step the user can give his feedback in various ways: as a discrete rating, a continuous rating, a pairwise comparison, et cetera. Furthermore, the ED step can change the criterion on which it chooses the new experiment. In order to handle these changes, we aim to break down the GPPL framework into a set of (minimally interdependent) components.

In the present work, we have chosen Forney-style Factor Graphs (FFG) as a formal framework for modularizing the GPPL algorithm. A particularly useful graph structure in FFG theory relates to ladder structures that describe Kalman filters and Hidden Markov models. In these structures, we distinguish forward message passes and backward message passes. A forward pass represents a prediction, and a backward pass a correction step, which is often automatically inherited through the graph structure. We will seek to discover similar forward and backward passes in the FFG realization of GPPL.

A. Problem Statement

Based on the foregoing analysis, we investigate the following questions in this thesis:

- 1) Is it possible to specify the Gaussian Process Preference Learning algorithm as a Message Passing Algorithm (MPA) on a Forney-style factor graph?
- 2) Can we identify forward and backward message passing schedules that correspond to prediction and correction stages in the GPPL algorithm?
- 3) If so, can we establish a relationship between GPPL and to FFG-based Kalman filtering?

B. Contributions

In this thesis report, we describe the following contributions to the existing body of work on GPPL:

- Section IV proposes GPPL as a MPA on a Forney-style Factor Graph.
- During this project, we discovered a fundamental issue of discomfort between the specification of factor graphs and the growing dimensionality of standard Gaussian processes in sequential data processing applications. This problem is addressed in Section V.
- Recursive GPR is re-interpreted as a FFG where messages could be labelled to belong to prediction and correction stages. This showed a clear relationship with a Kalman filter and is described in Section V-A.
- This recursive GPR is rewritten as recursive GPPL in Section V-B, which also shows a relationship with Kalman filters.
- As an experimental validation the parameters of a noise-suppression algorithm will be tuned by the demo in Section VIII. We show that through an intuitive user interface the optimal values can be found without intervention of a hearing aid specialist.

C. Outline

The structure of this thesis is as follows: Section II will give an introduction in Forney-style Factor Graphs (FFG). GPs are introduced in Section III by the derivation of Gaussian Process Prediction(GPP). An FFG for GPPL is proposed in Section IV. Here we discovered that the sequential growing of data is inevitable with normal GPs, which conflicts with the specification of factor graphs. Therefore an FFG for recursive GPs is proposed in Section V, where a fixed basis vector does not conflict with the specification of a factor graph. To ask the right question to the user Experiment Design (ED) is described as an FFG in Section VI. Then Section VII shows that with a minor adaptation of the framework GPPL can be seen as a special case of Gaussian Process Binary Classification (GPBC). As experimental validation the algorithm is used to tune a parameter of a noise-suppression algorithm in Section VIII. We finally conclude with Section IX.

The full perspective on the developed methods do not fit the budget of a twelve-page report. In this report we focus on concepts, results and key derivations. References to full derivations are provided where appropriate.

II. FORNEY-STYLE FACTOR GRAPHS

This section gives a brief introduction in Forney-style Factor Graphs (FFG). We follow the expositions described in [6], [7].

An FFG represents a factorization of a multivariate function as is illustrated by the following factorization:

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = f_A(x_1, x_2) \cdot f_B(x_3, x_4) \cdot f_C(x_2, x_4, x_5) \cdot f_D(x_5, x_6) \quad (1)$$

This factorization can be visualized as shown in Figure 2. An FFG represents a global function and consists of nodes, edges and half-edges. The following rules are obeyed:

- There is a node for every factor, also called local function.
- There is an edge or half-edge for every variable. A half-edge is connected to one node only.
- The node representing some factor f is connected with the edge representing x if and only if f is a function of x .

Implicitly this definition includes that no variable appears in more than two factors. By connecting the right nodes using the right edges, complex models can be build.

As an example we consider the factorization in Eq. (1). The factorized functions become nodes and the variables become (half-)edges, which is shown in Figure 2.

A. Summary-Propagation Algorithm

In signal processing and machine learning, it is often useful to compute the marginal function over one of the variables. For x_5 this is defined by:

$$f(x_5) \triangleq \int f(x_1, x_2, x_3, x_4, x_5, x_6) dx_1 dx_2 dx_3 dx_4 dx_6 \quad (2)$$

Since Eq. (1) shows that the function can be factorized, we are able to pull factors out of the integration in Eq. (2),

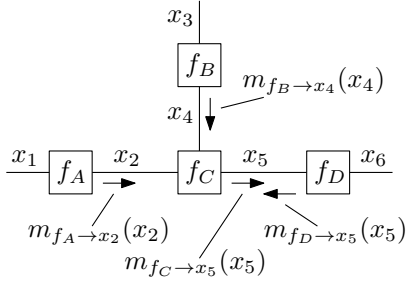


Figure 2. Example of a factor graph by Eq. (1)

which do not depend on the integration. For example f_D is only dependent on x_5 and x_6 . This leads to the following breakdown of the marginal of x_5 .

$$\begin{aligned}
 f(x_5) &= \underbrace{\int f_D(x_5, x_6) dx_6}_{\mu_{f_D \to x_5}(x_5)} \\
 &\underbrace{\int \int f_C(x_{2,4,5}) \cdot \int f_A(x_{1,2}) dx_1 \cdot \int f_B(x_{3,4}) dx_3 dx_{2,4}}_{\mu_{f_C \to x_5}(x_5)} \\
 &= \mu_{f_D \to x_5}(x_5) \cdot \mu_{f_C \to x_5}(x_5)
 \end{aligned} \tag{3}$$

The intermediate terms $\mu_{f \to x}(x)$ can be interpreted as messages flowing along the edges of the graph and are also shown in Figure 2. If both messages from f_C and f_D are available, the product of these two messages will give the marginal of x_5 .

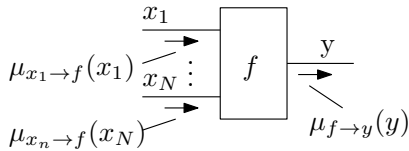


Figure 3. Messages of a generic node

In a more general setting we consider the generic node in Figure 3. We want to compute the message sent to y , given the messages on all other edges x_1, \dots, x_n . This is given by the *Sum-Product rule*:

$$\mu_{f \to y}(y) = \int \cdots \int_N f(y, x_1, \dots, x_N) \cdot \mu_{x_1 \to f}(x_1) \cdots \mu_{x_n \to f}(x_n) dx_1 \cdots dx_N \tag{4}$$

The message out of a factor node along edge y is the product of the function f and all the messages along all other edges, summarized (integrated) over all other variables except y . Since details are all integrated out, we can say that only a ‘summary’ is propagated over the edges. In general, messages on edges can be computed in both directions.

While alternative graph-based frameworks for factorization of functions exist, such as Bayesian networks and Markov

networks, the FFG framework enjoys the following advantages relative to other graphical models:

- FFGs are suitable for hierarchical modelling.
- FFGs are compatible with standard block diagrams.
- They have the simplest formulation of the Sum-Product message update.

B. Addition node

As an example we derive the rules of the addition node. In Figure 4 the addition node represents the factor $f(x, y, z) = \delta(x + y - z)$. It can also be viewed that the variables need to satisfy the constraint $X + Y = Z$. If this constraint is satisfied, we say that the configuration is valid. The message from the node to Z is given by the following equation

$$\begin{aligned}
 \mu_{f \to Z}(z) &= \int \int f(x, y, z) \mu_{X \to f}(x) \mu_{Y \to f}(y) dx dy \\
 &= \int \int \delta(x + y - z) \mu_{X \to f}(x) \mu_{Y \to f}(y) dx dy \tag{5} \\
 &= \mu_{X \to f}(x) + \mu_{Y \to f}(y)
 \end{aligned}$$

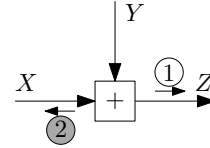


Figure 4. Addition node

Eq. (5) is valid for any message type. For Gaussian messages we can use the analytical property of addition of normally distributed variables, if all operands of a linear operation are Gaussian distributed, the distribution of the result is again in the same family. Therefore the message update can be described by the Gaussian message parameters, the mean μ and variance matrix Σ . Using the sum-product rule, the rule for the addition node is given by

$$\begin{aligned}
 \mu_{f \to Z}(z) &= \mathcal{N}_z(\mu_Z, \Sigma_Z) \\
 \mu_Z &= \mu_X + \mu_Y \\
 \Sigma_Z &= \Sigma_X + \Sigma_Y
 \end{aligned} \tag{6}$$

The edges in the graph indicate the direction of the forward message. However, messages can flow both directions. The messages ① and ② are given by

- ① The forward message of the addition node, following the constraint $X + Y = Z$

$$\begin{aligned}
 m_1(Z) &= \mathcal{N}(\mu_Z, \Sigma_Z) \\
 \mu_Z &= \mu_X + \mu_Y \\
 \Sigma_Z &= \Sigma_X + \Sigma_Y
 \end{aligned}$$

- ② Using the exact same sum-product rule to derive a backward message flowing from f to X we get

$$m_2(X) = \mathcal{N}(\mu_X, \Sigma_X)$$

$$\mu_X = \mu_Z - \mu_Y$$

$$\Sigma_X = \Sigma_Z + \Sigma_Y$$

The important difference in the second message is that the mean of Y is subtracted from Z and the variance is added.

C. Standard building blocks

Usually a factor graph or block also contains branching points, since a variable can be part of multiple factors. In an FFG this is represented by the following factor

$$f(X, Y, Z) \triangleq \delta(X - Y)\delta(X - Z) \quad (7)$$

Here $X = Y = Z$ makes a setting valid. In principle the equality node ‘clones’ a variable, this makes it possible to enforce that a variable does not appear in more than two factors.

In general, for each node definition and given distribution of the incoming messages, we could execute the sum-product rule and tabulate the outgoing message distribution in a lookup table. Table I stores the result of this exercise for common linear components and Gaussian messages. Constructing graphs with these nodes gives rise to automatically executable message passing based algorithms.

The benefit of tabulated results of sum-product messages is that marginalization in complex graphs can be executed by passing messages around the graph and executing the local tabulated sum-product rules at the nodes of the graph. As it turns out, a large subset of well-known signal processing algorithms, including the Kalman filter and the forward-backward algorithm in hidden Markov models can be interpreted as (sum-product) message passing algorithms on a FFG.

D. State space model

In this section the Gaussian message update rules for the standard building blocks, shown in Table I, are applied to derive the well-known Kalman filter recursions as message passing on an FFG.

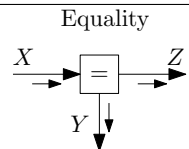
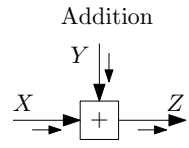
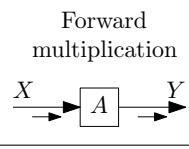
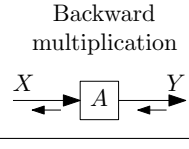
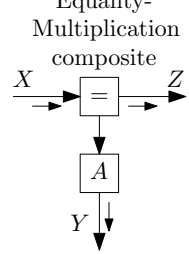
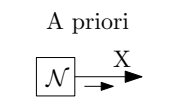
The Kalman filter is a Bayesian estimator for the state x_n of a linear state-space model given noisy observations $\{z_1, \dots, z_n\}$. The dynamic process has hidden state variables which cannot be observed directly. The control input noise u_n and the observation noise w_n are modelled as i.i.d. Gaussian distributions. The state-space model can be described by the following probabilistic density function

$$P(x_0, x_1, \dots, x_n, z_1, \dots, z_n) = P(x_0) \prod_{k=1}^n P(x_k | x_{k-1}) \prod_{k=1}^n P(z_k | x_k) \quad (8)$$

In [8] it is worked out how the Kalman filter can be broken down to messages on FFG with elementary nodes $\boxed{+}$, $\boxed{=}$ and \boxed{A} , which are given in Table I. The factor graph representation of one time-step of the Kalman filter is shown in Figure 5.

In the factor graph representation the probability density function in Eq. (8) is broken down into smaller factors

Table I
UPDATE EQUATIONS FOR STANDARD BUILDING BLOCKS

	Node	Update Rule
1		$\begin{aligned} \mu_Z &= (\Upsilon_X + \Upsilon_Y)^\# (\Upsilon_X \mu_X + \Upsilon_Y \mu_Y) \\ \Sigma_Z &= \Sigma_X (\Sigma_X + \Sigma_Y)^\# \Sigma_Y \\ \Upsilon_Z &= \Upsilon_X + \Upsilon_Y \\ \xi_Z &= \xi_X + \xi_Y \end{aligned}$
2		$\begin{aligned} \mu_Z &= \mu_X + \mu_Y \\ \Sigma_Z &= \Sigma_X + \Sigma_Y \\ \Upsilon_Z &= \Upsilon_X (\Upsilon_X + \Upsilon_Y)^\# \Upsilon_Y \\ \xi_Y + (\Sigma_X + \Sigma_Y)^\# (\Sigma_X \xi_X + \Sigma_Y \xi_Y) \end{aligned}$
3		$\begin{aligned} \mu_Y &= A \mu_X \\ \Sigma_Y &= A \Sigma_X A^\top \\ \Upsilon_Y &\stackrel{3}{=} A^{-\top} \Upsilon_X A^{-1} \\ \xi_Y &= (A \Sigma_X A^\top)^\# A \Sigma_X \xi_X \stackrel{1}{=} A^{-\top} \xi_X \end{aligned}$
4		$\begin{aligned} \mu_X &= (A^\top \Upsilon_Y A)^\# A^\top \Upsilon_Y \mu_Y \stackrel{2}{=} A^{-1} \mu_Y \\ \Sigma_X &\stackrel{3}{=} A^{-1} \Sigma_Y A^{-\top} \\ \Upsilon_X &= A^\top \Upsilon_Y A \\ \xi_X &= A^\top \xi_Y \end{aligned}$
5		$\begin{aligned} \mu_Z &= \mu_X + \Sigma_X A^\top G (\mu_Y - A \mu_X) \\ \Sigma_Z &= \Sigma_X - \Sigma_X A^\top G A \Sigma_X \\ \Upsilon_Z &= \Upsilon_X + A^\top \Upsilon_Y A \\ \xi_Z &= \xi_X + A^\top \xi_Y \\ G &= (\Sigma_Y + A \Sigma_X A^\top)^{-1} \end{aligned}$
6		$\begin{aligned} \mu_X &= \mu \\ \Sigma_X &= \Sigma \\ \Upsilon_X &= \Upsilon \end{aligned}$
<p># denotes the Moore-Penrose pseudoinverse ¹ if A and Σ_x are positive definite ² if A and Υ_x are positive definite ³ if A is invertible</p>		

$$P(x_n | x_{n-1}) = \mathcal{N}(A x_{n-1}, \Sigma_{n-1} + b \Sigma_w b^\top) \quad (9)$$

$$P(z_n | x_n) = \mathcal{N}(c^\top x_n, c^\top \Sigma_n c + \Sigma_w) \quad (10)$$

$$P(x_0) = \mathcal{N}(\mu_0, \Sigma_0) \quad (11)$$

Our goal is similar, we want to describe GPPL as a sequence of messages on an FFG. We extend the model with experiment design to choose the best experiments. The following probability function will be calculated:

$$P(f_0, f_1, \dots, f_n, x_1, \dots, x_n, y_1, \dots, y_n) = P(f_0) \prod_{k=1}^n P(x_k | y_{k-1}) \prod_{k=1}^n P(f_k | f_{k-1}) \prod_{k=1}^n P(y_k | f_k) \quad (12)$$

This probability density function can be broken into smaller factors $P(f_0)$, $P(x_n | f_{n-1})$, $P(f_n | f_{n-1})$ and $P(y_n | f_n)$. One

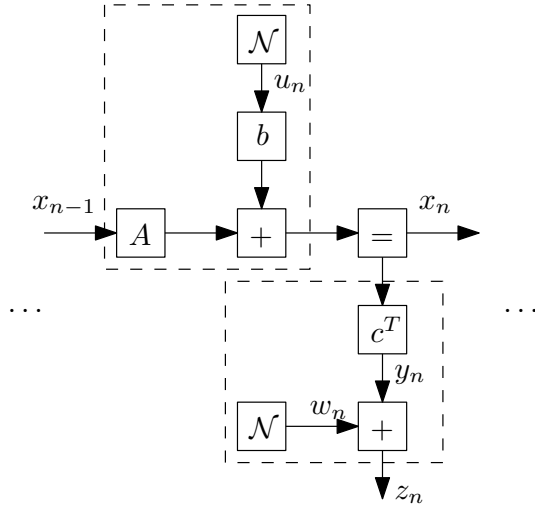


Figure 5. Factor graph representation of a Kalman filter.

time-step of this modularized GPLL is represented by the factor graph shown in Figure 6. The next sections will describe the components of this factor graph.

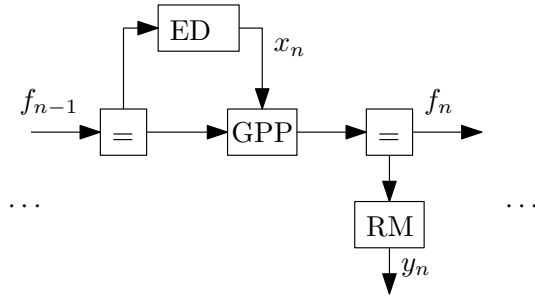


Figure 6. Factor graph representation of Eq. (12)

You might have noticed that we use f as latent variable in the factor graph and g as perception model for the user. In Section VII we see how g is reconstructed from f for pairwise preference learning.

III. GAUSSIAN PROCESSES

Gaussian Processes (GP) can be used to model as framework for preference learning. This section will provide the notations that are used throughout this thesis. Furthermore it gives Gaussian Process Prediction (GPP) as an MPA on an FFG, which represents $P(f_n|y_{n-1})$.

A. Notation

GPs allow non-parametric learning of a regression function from noisy data. They can be considered as Gaussian distributions over functions conditioned on the data [1].

For GP regression we assume that a set of data is drawn from a noisy process:

$$y = f(x) + \epsilon \quad (13)$$

where $x_n \in \mathbb{R}^q$ are the inputs, $y_n \in \mathbb{R}$ are the observations and $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is Gaussian noise. The vector of observations

will be written as $y^n = [y_1, \dots, y_n]^\top$ and the matrix with inputs vectors is written as $X_n = [x_1, \dots, x_n]^\top$. A GP is used to infer the latent function $f(\cdot)$ from the data. A positive semi-definite covariance function $k(x_p, x_q) \triangleq \text{cov}\{f_p, f_q\}$ specifies the covariance between inputs, where $f_n \triangleq f(x_n)$. A typical covariance kernel function, also called kernel, is the squared exponential kernel

$$k(x_i, x_j) = \alpha^2 \cdot \exp\left(-\frac{1}{2}(x_i - x_j)^\top \sigma^{-1}(x_i - x_j)\right) \quad (14)$$

B. Gaussian Process Prediction

Given a set of observations, GPP estimates a probability distribution for new samples. Figure 7 shows an example where six noisy observations are made from the thin blue curve. The thick curve shows the mean and the shaded area shows the standard deviation around the estimated points.

For any finite set of inputs X_b a GP provides a multivariate Gaussian distribution of outputs [1]. This distribution of f_n for an arbitrary test input x_n is a univariate Gaussian with mean and variance

$$\begin{aligned} \mu_f(x_n) &= k(X_b, x_n)^\top K_b^{-1} \mu_f(x^b) \\ \sigma_f(x_n) &= k(x_n, x_n) - k(X_b, x_n)^\top K_b^{-1} k(X_b, x_n) \end{aligned} \quad (15)$$

Here $K_b \triangleq k(X_b, X_b)$.

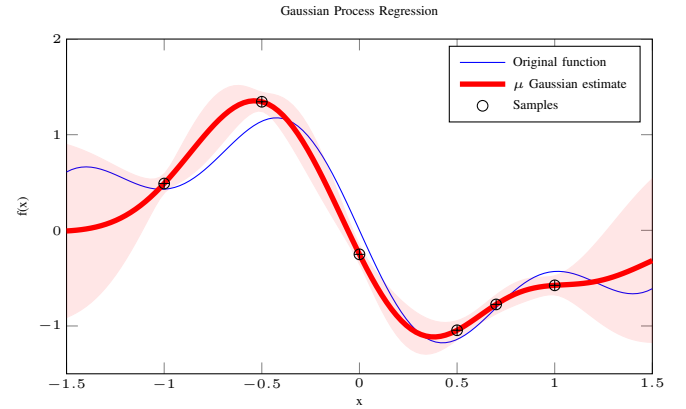


Figure 7. Predicted function with GPP. Noisy samples are drawn from the original function. In red the estimate is drawn with one standard deviation around it.

In order to determine $P(f_n|y_{n-1})$, we will calculate the joint prior $P(f_n, f^b|y_{n-1}) = P(f_n|f^b) \cdot P(f^b|y_{n-1})$, where the chain rule is applied. Here f^b is the vector of known latent variables, for example f_{n-1} .

After some algebraic transformations, see [1, Appendix A], where the basic properties of Gaussian distributions and the Woodbury formula is utilized, we can write down the joint prior as

$$\begin{aligned} P(f_n, f^b|y^b) &= P(f_n|f^b) \cdot P(f^b|y^b) \\ &= \mathcal{N}(\mu_n, D) \cdot \mathcal{N}(\mu^b, \Sigma_b) \\ &= \mathcal{N}\left(\begin{bmatrix} \mu^b \\ \mu_n \end{bmatrix}, \begin{bmatrix} \Sigma_b & \Sigma_b J^\top \\ J \Sigma_b & \Sigma_n \end{bmatrix}\right) \end{aligned} \quad (16)$$

with

$$\begin{aligned}\mu_n &= J\mu^b \\ \Sigma_n &= D + J\Sigma_b J^T \\ D &= k(x_n, x_n) - Jk(X_b, x_n) \\ J &= k(X_b, x_n)^T K_b^{-1}\end{aligned}\quad (17)$$

For notational convenience we used J and D to make the derivations in the FFG more clear. We will derive Eq. (16) by sum-product message passing on an FFG and construct the GPP node for the factor $P(f_n | y^b)$. Figure 8 shows this GPP node. The arrows represent the messages that are sent over the edges for the local computations. The circled numbers give a possible message schedule.

A GP model is a non-parametric model and as such keeps all the observed data in its memory banks. The datarail is used to store the data, push new observations into storage and pass the database along over the time steps in the graph.

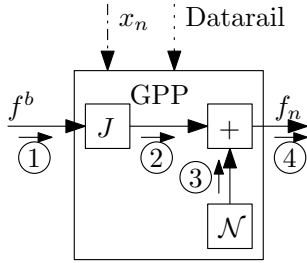


Figure 8. Eq. (16) by sum-product message passing on an FFG. The datarail is used to push new observations and pass the data along over the time steps in the graph.

- ① This consist the estimated latent variable or the noisy observations used for prediction.

$$m_1(f^b) = \mathcal{N}(\mu^b, \Sigma_b)$$

- ② Forward matrix multiplication with J from Eq. (17) (Table I-3).

$$m_2(\cdot) = \mathcal{N}(J\mu^b, J\Sigma_b J^T)$$

- ③ Gaussian variance D from Eq. (17).

$$m_3(\cdot) = \mathcal{N}(0, D)$$

- ④ Predicted f_n . Addition of ④ and ⑤.

$$\begin{aligned}m_4(f_n) &= \mathcal{N}(\mu_n, \Sigma_n) \\ \mu_n &= J\mu^b \\ \Sigma_n &= D + J\Sigma_b J^T\end{aligned}$$

Which coincide with the derivation treated before. The interesting part of this derivation is that via a sequence of automatic local computations, Eq. (16) can be calculated. This can now be embedded as a composite node in the GPPL algorithm. This composite node can be connected via the edge

given in Figure 8 and we do not need to bother about the local computations any more.

Like this GPP, other complex algorithms can be driven by factor graphs via the same mechanism, this allows embedding algorithms in applications much easier.

IV. GAUSSIAN PROCESS BINARY CLASSIFICATION

In order to describe GP based preference learning, we will first describe GP based binary classification (GPBC) as an extension of GP regression. Later, in section Section VII, we will see that GPPL is a special case of GPBC. This section will modularize the algorithm to present it as an MPA on an FFG. The algorithm is separated into two parts, a prediction part, and a correction part.

Binary classification associates experiments x_n with binary class labels $y_n \in \{-1, +1\}$. Our task is to predict the class membership y_n of a new test point x_n , which is achieved using a latent variable f_n . A cumulative Gaussian sigmoid function $\Phi(f_n)$ is used to squash the latent variable f_n to a probability interval $[0, 1]$.

Figure 9 shows an example of binary classification. Experiments $x_n = [\theta_1, \theta_2]$ are made and have either the class label ‘blue crosses’ or ‘yellow circles’. Our task is to give the probability that a new experiment belongs to the blue crosses. In Figure 9 this is shown by the grey-scale map and the contours.

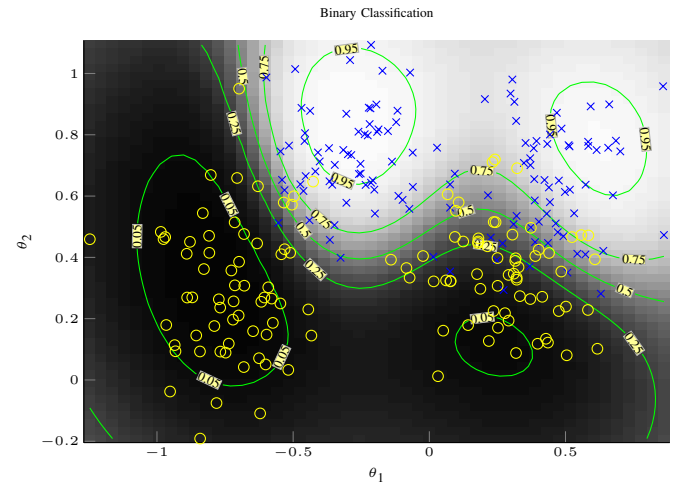


Figure 9. Predictive probability with contours of binary classification. The numbers represent the probability a new experiment belongs to the blue cross class.

In Figure 10 we see the processing that is involved on the basis of a new observation y_n . A full FFG would comprise a chain of such graph sections, one for each time step. At the moment a new experiment is picked, the prediction of the class label will be made. The GPP node, which is discussed in previous section, predicts the latent variable corresponding to this experiment and adds it to the existing latent variable, $P(f_n, f^{n-1} | y^{n-1})$. The Response Model (RM) will convert the latent variables into predictions of class labels $P(y_n | f_n)$. Once a new observation is done, this information will be used

Table II
CUMULATIVE GAUSSIAN LOG LIKELIHOOD AND ITS FIRST AND SECOND DERIVATIVE

$\log P(y_i f_i)$	$\nabla_{f_i} \log P(y_i f_i)$	$\nabla_{f_i}^2 \log P(y_i f_i)$
$\log \Phi(y_i f_i)$	$\frac{y_i \mathcal{N}(f_i)}{\Phi(y_i f_i)}$	$-\frac{\mathcal{N}(f_i)^2}{\Phi(y_i f_i)} - \frac{y_i f_i \mathcal{N}(f_i)}{\Phi(y_i f_i)}$

Laplace approximation to iteratively update the posterior $P(f^n | X_n, y^n)$.

$$\begin{aligned} m_6(f^n) &= Q(f^n | X_n, y^n) = \mathcal{N}(\mu^n, \Sigma_n) \\ \mu^{\text{new}} &= (K_n^{-1} + W_n)^{-1} (W_n \mu^n + \nabla_f \log P(y^n | f^n)) \\ \Sigma_n &= (K_n^{-1} + W_n)^{-1} \end{aligned}$$

By Bayes' rule the posterior over the latent variables is

$$P(f^n | X_n, y^n) = \frac{P(y^n | f^n) P(f^n | X_n)}{P(y^n | X_n)} \quad (18)$$

The Laplace method makes a second order Taylor expansion around the maximum of $\log P(f^n | X_n, y^n) = \Psi(f^n)$. This Gaussian has mean $\mu^n = \text{argmax}_{f^n} \log P(f^n | X_n, y^n)$ and covariance $\Sigma_n = (-\nabla_f^2 \log P(f^n | X_n, y^n)|_{f^n=\mu^n})^{-1}$. Since $P(y^n | X_n)$ is independent of f^n , the unnormalized posterior is considered to fit the approximation. The complete derivation can be found in [1, Section 3.4], which leads to the following iterative update

$$f_n^{\text{new}} = (K_n^{-1} + W_n)^{-1} (W_n f_n^{\text{old}} + \nabla_f \log P(y^n | f^n)) \quad (19)$$

These iterations continue until $\Psi(f^n)$ has converged. At each iteration we need an update of the likelihood $P(y^n | f^n)$ with its first and second derivative. This means there are messages sent back and forth between the inference module and the RM which is represented by the dots besides message ④ and ⑦. This finally leads to the approximated Gaussian

$$Q(f^n | X_n, y^n) = \mathcal{N}(\mu^n, (K_n^{-1} + W_n)^{-1}) \quad (20)$$

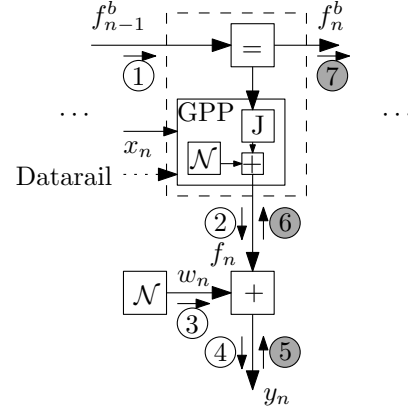
V. RECURSIVE GAUSSIAN PROCESSES AS FORNEY-STYLE FACTOR GRAPH

Despite the fact that new data is received sequentially, the Laplace method evaluates the complete latent space, which does not allow sequential updating. Furthermore GP are non-parametric, which leads to a growing dimension of the data every time step. This is not desirable for the formulation of Factor graphs. At the moment GPPL is combined with other algorithms or a smoothing function is applied, the growth of the dimension can lead to complex protocols. Therefore we propose recursive Gaussian Process Regression (GPR) formulated as an MPA on an FFG based on [9]. A fixed basis vector is used to map all the data. After that, we show that GPR can easily be changed to binary classification by changing the RM.

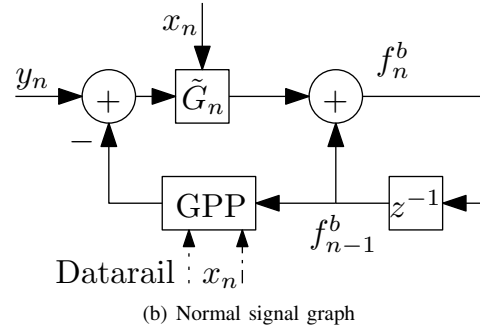
A. Gaussian Process Regression

In [9], recursive GPR is proposed for sequential learning with a fixed basis vector. This algorithm is visualized as a

Kalman filter in a normal signal-flow graph in Figure 11(b). The corresponding FFG for the Kalman filter is shown in Figure 11(a). The arrows represent the messages along the edges of the graph. The update schedule is defined by the circled numbers and is divided in two groups: firstly, the prediction of the new observation is calculated via the white messages. Secondly, the new data is utilized for correcting the estimate via the grey messages. The composite GPP node is derived in Section III-B.



(a) Forney-style Factor Graph with message schedule



(b) Normal signal graph

Figure 11. Two representations of recursive GPR. Both represent the update for time step n .

- ① State estimate at time $n - 1$

$$m_1(f_{n-1}^b) = \mathcal{N}(\mu_{n-1}^b, \Sigma_{n-1}^b)$$

- ② Prediction as is derived in Section III-B. The GPP node can be seen as a matrix multiplication J followed by an addition of Gaussian distribution $\mathcal{N}(0, D)$. The local update equations for these nodes are given by Table I-2 and Table I-3 respectively.

$$m_2. = P(f_n | f_{n-1}^b, x_n) = \mathcal{N}(\mu_n, \Sigma_n)$$

$$\mu_n = J \cdot \mu_{n-1}^b$$

$$\Sigma_n = D + J \Sigma_{n-1}^b J^T$$

$$D = k(x_n, x_n) - J \cdot k(X_b, x_n)$$

$$J = k(x_n, X_b) \cdot K_b$$

- ③ Gaussian distributed measurement noise.

$$m_3(w_n) = \mathcal{N}(0, \sigma_w^2)$$

- ④ Prediction of the observation with measurement noise.

$$m_4(y_n) = \mathcal{N}(\mu_n, \Sigma_n)$$

$$\mu_n = J \cdot \mu_{n-1}^b$$

$$\Sigma_n = D + J \Sigma_{n-1}^b J^T + \sigma_w^2$$

- ⑤ The observation.

$$m_2(y_n) = \mathcal{N}(y_n, 0)$$

- ⑥ Observation with Gaussian noise.

$$m_4(\cdot) = \mathcal{N}(y_n, \sigma_w^2)$$

- ⑦ Update of the state with information from the new observation. Since we created the factor for prediction, we can use the same modules for the update. The backward matrix multiplication would require a matrix inversion, which is not possible since J is singular. Therefore the equation-multiplication composite block, table II-5 in supplement, is used.

$$m_5(f_n^b) = \mathcal{N}(\mu_n^b, \Sigma_n^b)$$

$$\mu_n^b = \mu_{n-1}^b + \frac{\Sigma_{n-1}^b J^T (y_n - J \mu_{n-1}^b)}{D + \sigma_w^2 + J \Sigma_{n-1}^b J^T}$$

$$\Sigma_n^b = \Sigma_{n-1}^b - \frac{\Sigma_{n-1}^b J^T J \Sigma_{n-1}^b}{D + \sigma_w^2 + J \Sigma_{n-1}^b J^T}$$

$$J = k(x_n, X_{n-1}) \cdot k(X_{n-1}, X_{n-1})^{-1}$$

This coincides with the result from [?]:

$$\mu^p = J \mu_{n-1}^b \quad (21)$$

$$\Sigma^p = J \Sigma_{n-1}^b J^T + D \quad (22)$$

$$\mu_n^b = \mu_{n-1}^b + \tilde{G}_n \cdot (y_n - \mu^p) \quad (23)$$

$$\Sigma_n^b = \Sigma_{n-1}^b - \tilde{G}_n J \Sigma_{n-1}^b \quad (24)$$

With Kalman gain:

$$\tilde{G}_n = \frac{\Sigma_{n-1}^b J^T}{\sigma_w^2 + J \cdot \Sigma_{n-1}^b J^T + D} \quad (25)$$

Crucially, writing the prediction step of recursive GPR as a factor graph automatically inherits the Kalman correction of the basis vector. Since prediction is intuitive this proves that factor graphs can be useful for creating algorithms and make them easy adaptable.

B. Gaussian Process Binary Classification

In GPR the observation is a real number, $y \in \mathcal{R}$. In GPPL this observation is a binary label. This means the RM is changed to the cumulative Gaussian sigmoid. Figure 12 shows

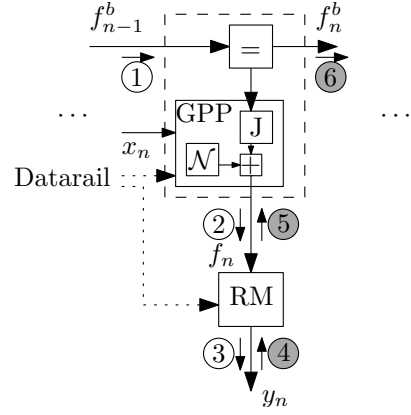


Figure 12. FFG for recursive GPBC

the FFG for binary classification. A possible message schedule is given by the circled numbers and is treated below.

- ① The latent variable from the previous time step.

$$m_1(f_{n-1}^b) = \mathcal{N}(\mu_{n-1}^b, \Sigma_{n-1}^b)$$

- ② Prediction of f_n as in Section V-A.

$$m_2(f_n) = \mathcal{N}(\mu_n, \Sigma_n)$$

$$\mu_n = J \mu_{n-1}^b$$

$$\Sigma_n = J \Sigma_{n-1}^b J^T + D$$

- ③ Prediction of y_n just as in Section IV.

$$m_3(y_n) = B(\mu_y)$$

$$\mu_y = P(y_n)$$

$$= \int P(y_n | v) P(f_n) df_n$$

$$= \int \Phi\left(\frac{f_n}{\sigma_n}\right) \mathcal{N}(\mu_n, \sigma_n^2) df_n$$

$$= \Phi\left(\frac{\mu_n}{v \sqrt{1 + \sigma_n^2/v^2}}\right)$$

- ④ The observation y_n .

$$m_4(y_n) = \mathcal{N}(y_n, 0)$$

- ⑤ The Laplace approximated variable f_n .

$$m_5(f_n) \approx \mathcal{Q}(f_n | y_n, f_{n-1}^b) = \mathcal{N}(\mu_{lp}, \Sigma_{lp})$$

$$\mu_{lp}^{\text{new}} = (D^{-1} + W_n)^{-1} \cdot$$

$$(W_n(\mu_{lp}^{\text{old}} - \mu_n) + \nabla_f \log P(y_n | f_n)) + \mu_n$$

$$\Sigma_{lp} = (D^{-1} + W_n)^{-1}$$

By Bayes' rule the posterior over the new latent variable is

$$P(f_n | y_n, f_{n-1}^b) = \frac{P(y_n | f_n) P(f_n | f_{n-1}^b)}{P(y_n | f_{n-1}^b)} \quad (26)$$

This posterior cannot be solved analytically, since $P(y_n|f_n)$ is non-Gaussian. Therefore we again apply the Laplace approximation to get a Gaussian distribution, $P(f_n|y_n, f_{n-1}^b) \approx \mathcal{N}(\mu_{lp}, \sigma_{lp}^2)$. The difference with the previous derivation is that prior is different, namely

$$\begin{aligned} P(f_n|f_{n-1}^b) &= \mathcal{N}(J \cdot \mu_{n-1}^b, D) \\ D &= k(x_n, x_n) - J \cdot k(X_b, x_n) \\ J &= k(x_n, X_b) \cdot K_b \end{aligned} \quad (27)$$

Since $P(y_n|f_{n-1}^b)$ is independent of f_n we can again consider the un-normalized logarithmic posterior to get the best fit. Appendix A shows the complete derivation for a Laplace update with non-zero mean prior. The posterior is found by Newton iterations,

$$\begin{aligned} \mu_{lp}^{\text{new}} - \mu_n &= (D^{-1} + W_n)^{-1} \cdot \\ & (W_n(\mu_{lp}^{\text{old}} - \mu_n) + \nabla_f \log P(y_n|f_n)) \end{aligned} \quad (28)$$

Then the approximated posterior is defined as

$$Q(f_n|X_n, y_n) = \mathcal{N}(\mu_{lp}, (D^{-1} + W_n)^{-1}) \quad (29)$$

- 6 For the correction of the basis vector we use the approximated Gaussian distribution Q to update the basis vector. Since J is singular, we will again use the equation-multiplication node defined in Table I. This automatically inherits a Kalman update to incorporate the new data in the basis vector.

$$\begin{aligned} m_6(f_n^b) &= \mathcal{N}(\mu_n^b, \Sigma_n^b) \\ \mu_n^b &= \mu_{n-1}^b + \frac{\Sigma_{n-1}^b J^T (\mu_{lp} - J \mu_{n-1}^b)}{D^{-1} + W_n)^{-1} + D + J \Sigma_{n-1}^b J^T} \\ \Sigma_n^b &= \Sigma_{n-1}^b - \frac{\Sigma_{n-1}^b J^T J \Sigma_{n-1}^b}{(D^{-1} + W_n)^{-1} + D + J \Sigma_{n-1}^b J^T} \\ D &= k(x_n, x_n) - J \cdot k(X_b, x_n) \\ J &= k(x_n, X_b) \cdot K_b \end{aligned}$$

Two side notes have to be made. The first note is that the algorithm becomes parametric, since you have to choose your basis vector. Secondly, some information gets lost, since it is mapped onto this basis vector.

VI. EXPERIMENT DESIGN

As the hearing aids will be adapted to humans, we cannot afford to ask thousands of questions, so the dataset which can be used will be small. A new experiment should be as informative as possible. This section develops an algorithm to select informative listening experiments. As before, we will specify the algorithm in the context of a message passing algorithm on a FFG.

The ED node represents the following factor

$$f(x_n) = \delta(x_n - \arg \max(h(x_{test}))) \quad (30)$$

The Experiment Design (ED) node is a so-called composite node and consists of the following three sub-nodes:

- 1) a GPP node as described in Section III-B that estimate the preference function.
- 2) an Active Learning (AL) node that evaluates the test samples.
- 3) a argmax node that chooses the test sample with the highest rating.

Figure 13 shows the ED node. The circled numbers show a possible message schedule.

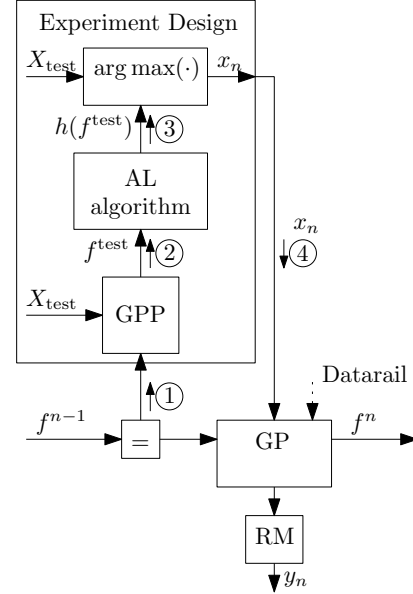


Figure 13. Representation of functions within the Experiment Design composite node. The GPR makes a prediction on the latent variables corresponding to candidate points. The AL algorithm evaluates these points and the best candidate will be drawn.

- 1 A copy of the latent variable is used for prediction.

$$m_1(f^{n-1}) = \mathcal{N}(\mu_{f^{n-1}}, \Sigma_{f^{n-1}})$$

- 2 The GPR node as described in Section IV predicts the distribution of the test samples.

$$m_2(f^{\text{test}}) = \mathcal{N}(\mu_{f^{\text{test}}}, \Sigma_{f^{\text{test}}})$$

- 3 An Active Learning algorithm gives a grade to the test samples.

$$m_3(f^{\text{test}}) = h(f^{\text{test}})$$

Various Active Learning (AL) algorithms can be used to evaluate test samples. Since we are interested in finding the maximum preference value, we use a combination between exploration and exploitation. Exploration means that experiments with an high uncertainty are chosen. Exploitation exploits certain area's, which means in our case that experiments with a high mean are done.

$$h(f^{\text{test}}) = \mu_{f^{\text{test}}} + \alpha \sigma_{f^{\text{test}}}^2 \quad (31)$$

In [4], the author proposes some algorithms in this form and the quality depends on the situation. So for now an α that gives reasonable results is chosen.

- ④ The sample x_n with the highest learning rate is taken.

$$m_4(x_n) = \mathcal{N}(\mu_{x_n}, 0)$$

$$\mu_{x_n} = X_{\text{test}}[\text{index}]$$

Where index the index number of $h_{\max} = \arg \max_{f^{\text{test}}}(h(f^{\text{test}}))$.

Since we cannot get the Latent variable from a new experiment, this node is a forward-only node.

VII. GAUSSIAN PROCESS PREFERENCE LEARNING AS A SPECIAL CASE OF GAUSSIAN PROCESS BINARY CLASSIFICATION

We will use GPPL to estimate the users' preference function. A simple trick, proposed by [5], enforces symmetry on GPBC and converts binary classification to preference learning. The reason to apply this trick is that GPBC is much better documented than GPPL. The conversion is explained and the derivation for reconstruction of the original preference function is given in this section.

The perception model $\tilde{g}(\theta)$ of the user is estimated by pairwise GPPL. To find $g(\theta)$ the user listens iteratively to two samples with settings $\theta_{1,n}$ and $\theta_{2,n}$. These pairs of items $x_n = [\theta_{1,n}, \theta_{2,n}]$ are associated binary labels $y_n \in \{-1, +1\}$, where $y_n = +1$ means $\theta_{1,n} \succ \theta_{2,n}$. Since users do not provide consistent answers to the same question, a setting will be perceived as $g(\theta) + \epsilon$, where $\epsilon = \mathcal{N}(0, \sigma_\epsilon^2)$ denotes an Gaussian noise around the preference level. Because of the uncertainty ϵ , the probability that the user indeed prefers one setting over another is cumulative Gaussian distributed. By comparing multiple pairs of θ_1 and θ_2 the original unknown preference function can be found.

$$\begin{aligned} P(y = 1 | \theta_1, \theta_2, g) &= P(\theta_1 \succ \theta_2) \\ &= P(g(\theta_1) + \epsilon > g(\theta_2) + \epsilon) \\ &= \Phi\left(\frac{g(\theta_1) - g(\theta_2)}{\sqrt{2}\sigma_\epsilon}\right) \\ &= \Phi(f(\theta_1, \theta_2)) \end{aligned} \quad (32)$$

Here the likelihood only depends on the difference between $g(\theta_1)$ and $g(\theta_2)$. So we define $f(\theta_1, \theta_2) \triangleq g(\theta_1) - g(\theta_2)$ and we choose the variance of ϵ such that $\sqrt{2}\sigma_\epsilon = 1$.

Since $f(\theta_1, \theta_2)$ is obtained from g via a linear operation, f is also a GP with mean μ_{pref} and variance k_{pref} .

$$\begin{aligned} g(x) &\sim \mathcal{N}(\mu_g, k) \\ f(x) &\sim \mathcal{N}(\mu_{\text{pref}}, k_{\text{pref}}) \end{aligned} \quad (33)$$

The mean μ_{pref} is given by the following equation.

$$\mu_{\text{pref}}(\theta_1, \theta_2) = \mu_g(\theta_1) - \mu_g(\theta_2) \quad (34)$$

The preference-judgement kernel is obtained by taking the covariance between two inputs.

$$\begin{aligned} k_{\text{pref}}(x_i, x_j) &= \text{cov}[f(\theta_{1,i}, \theta_{2,i}), f(\theta_{1,j}, \theta_{2,j})] \\ &= k(\theta_{1,i}, \theta_{1,j}) + k(\theta_{2,i}, \theta_{2,j}) - \\ &\quad k(\theta_{1,i}, \theta_{2,j}) - k(\theta_{2,i}, \theta_{1,j}) \end{aligned} \quad (35)$$

In this equation we choose the covariance $k(\theta_1, \theta_2)$ to be the squared exponential kernel, since this gives a smooth function, which is desirable for audio perception.

Figure 14 gives us a 2D representation of our unknown preference function. This illustration is just for conceptual understanding of the algorithm. As we can see the predicted probability is anti-correlated over the diagonal $\theta_1 = \theta_2$, due to the preference kernel.

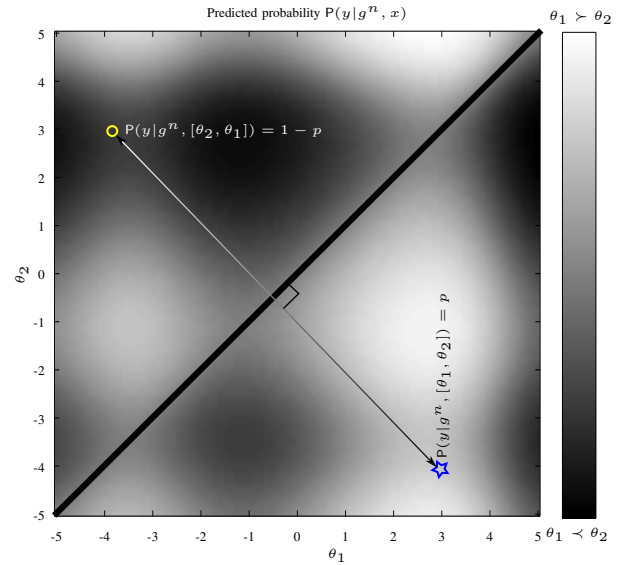


Figure 14. Probability distribution of $P(y|g^n, x)$. A training sample is given by a blue stars or a yellow circles, which represents the preference of θ_1 over θ_2 and θ_2 over θ_1 respectively. Learning from these observations gives us $P(y|g^n, x)$, which is illustrated with a grey scale level. As can be seen the preference is anti-correlated over the diagonal $\theta_1 = \theta_2$, which is what we expect, since $P(g(\theta_1) \succ g(\theta_2)) = 1 - P(g(\theta_2) \succ g(\theta_1))$

To retrieve the original preference function from this 2D graph we are going to estimate the mean and variance. We will choose an arbitrary $\hat{\theta}_2$ and a test input θ_n and use it in the joint probability $f(\theta_n, \hat{\theta}_2)$, which gives us Eq. (36). This means in principle that the preference function $g(\theta_n)$ is identical for all arbitrary $\hat{\theta}_2$ with an offset $\mu_g(\hat{\theta}_2)$.

$$\mu_g(\theta_n) = \mu_{\text{pref}}(\theta_n, \hat{\theta}_2) - \mu_g(\hat{\theta}_2) \quad (36)$$

where

$$\mu_g(\hat{\theta}_2) = \sum_{\theta_n} \mu_{\text{pref}}(\theta_n, \hat{\theta}_2) \quad (37)$$

To calculate the variance of g we assume the inputs θ_1 and θ_2 are independent and stacked $\theta_b = [\theta_1, \theta_2]^T$. Following the

prediction rules of GP in Eq. (15) we get the variance

$$\Sigma_g(\theta_n) = k(\theta_n, \theta_n) - k(\theta_b, \theta_n)^\top K_b^{-1} k(\theta_b, \theta_n) \quad (38)$$

VIII. EXPERIMENTAL VALIDATION: TUNING OF A NOISE SUPPRESSION ALGORITHM

In order to validate and demonstrate the developed methods, we present here the application of GPPL (on an FFG) to the task of tuning a parameter in an audio noise reduction algorithm. Audio noise algorithms are of great use in mobile phones, hearing aids and speech recognition systems.

Users of hearing aids perceive a certain quality of audio which can be represented by a preference function. The most preferred parameter value of this algorithm is estimated via pairwise GPPL. A graphical interface shows buttons to respond, to play sound samples and it shows the user his estimated preference function.

Our validation experiment was fully implemented in MATLAB. We implemented the GPPL framework as an FFG by extending SumProductLab, which is a freely available MATLAB toolbox for sum-product algorithms [10].

Using the framework of Section IV the demonstration is constructed the same way as is explained in the introduction and consist of the following steps

BUILD - Every iteration the user will compare a pair of settings. One of these settings will be the winning setting of the previous iteration. The Experiment Design (ED) chooses the second setting, which is in our case a certain parameter setting. To make an informed choice, this new experiment is based on both the mean and the variance of the estimated preference function. This pair of settings is sent to the execute step as the new experiment. The AL function $h(x_{\text{test}})$ is shown in lower-left plot of Figure 15.

EXECUTE - The execute step is in our case the hearing aid with a noise-suppression algorithm. This algorithm is considered as a ‘black box’ where a setting is given as input and an audio sample is returned. As can be seen in Figure 15 the user can in his turn play sample 1 or 2, which will be played on some speakers.

OBSERVE - The user will judge these audio samples based on his preference function. Once the user decides which audio sample he prefers he can click ‘Win 1’ or ‘Win 2’. This observation is used as input for the preference learning algorithm

ADAPT - With this new observation and the prior knowledge of previous iterations the latent variables are adapted. The latent variable is sent to the ED.

This iterates until the user is satisfied and the best parameter setting is found. The upper-left plot in Figure 15 shows the predicted preference function of the user. The lower-right plot represents the 2D representation of $P(y|g, x)$. Just as in Section VII the blue crosses and yellow circles represent the past observations and the grey-scale surface the probability $g(\theta_1) \succ g(\theta_2)$.

IX. CONCLUSION

This paper discussed several questions as mentioned in the problem statement and presented answers to these questions.

It is possible to construct Gaussian Process Preference Learning as a MPA on a Forney-style Factor Graph. However, we found a fundamental issue between the specification of factor graphs and the growing dimensionality of the latent variables of standard Gaussian Processes in sequential data processing applications.

Therefore we proposed to construct an FFG based on recursive Gaussian Processes by [9]. The original algorithm has been re-interpreted as probabilistic inference by message passing on the graph. As a result, the final graph structure for GPPL is simple and resembles the structure of a Kalman filter.

To find the unknown preference function as fast as possible, the graphs can easily be extended with Experiment Design. An Active Learning (AL) algorithms make sure new experiments consist a much information as possible. This experiment design is represented by an FFG node and can easily be adapted to problem-specific circumstances.

The experimental validation showed that the parameter of a noise-reduction algorithm can be tuned by the user himself via pairwise GPPL. The user listens to sound samples and gives his judgement, which finally leads to the most preferred setting.

In this thesis we assumed that the parameters of the kernel functions, the so-called hyper-parameters, are fixed values. These hyper-parameters can be optimized to give the best possible fit of the data. This optimization is performed by maximizing the marginal likelihood and is implemented for the GPBC.

To avoid numerical instability and reduce the number of matrix inversions some tricks have been applied in the algorithms, which are given in [1, Section 3.4.3].

Future work could focus on implementation and validation of the recursive GP. This can further be extended by learning the hyper-parameter optimization in the FFG as described by [?] and optimizing the positions of the basis vector.

X. ACKNOWLEDGEMENT

I would like to thank Marco Cox, Thijs van de Laar, Marija Milenkovic and René Duijkers of the ‘SPS Brats’ team for the help I got during the team meetings. I also would like to thank my graduation panel led by Tjalling Tjalkens for their time and feedback. I especially thank Bert de Vries for his guidance throughout the project. The book, [1], and toolbox, [11], of Rasmussen helped me very much to understand Gaussian Processes.

REFERENCES

- [1] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Massachusetts Institute of Technology, 2006.
- [2] W. Chu and Z. Ghahramani, “Preference Learning with Gaussian Processes,” in *Proceedings of the 22nd international conference on Machine learning*. London: ACM, 2005, pp. 137–144.
- [3] P. C. Groot, T. Heskes, and T. M. H. Dijkstra, “Nonlinear Perception of Hearing-Impaired People Using Preference Learning with Gaussian Processes,” *Radboud University Nijmegen, Tech. Rep. ICIS-R08018*, 2008.

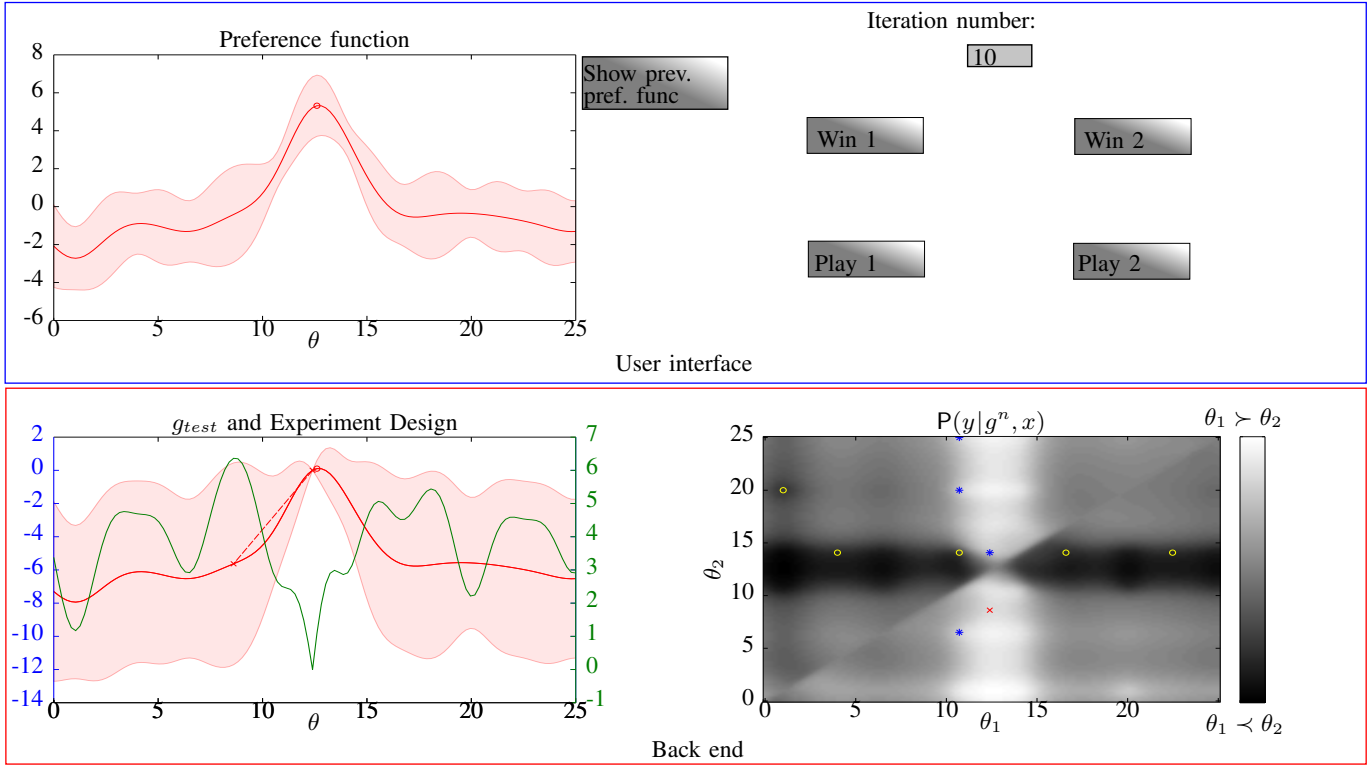


Figure 15. Graphical interface of the demonstration. The user listens to the two different fragments and can then click which one he prefers ('Win 1' or 'Win 2'). This procedure continues until the user is satisfied. In the top-left corner the estimated preference function is plotted. At the back end the expected improvement determines the new experiment to compare with the winning sample of the previous iteration. At the bottom-right corner the $P(y_n | g^n)$ is given by a grey-scale map.

- [4] E. Brochu, V. M. Cora, and N. de Freitas, "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning," *arXiv preprint arXiv:1012.2599*, Dec. 2010. [Online]. Available: <http://arxiv.org/abs/1012.2599>
- [5] F. Huszar, "A GP classification approach to preference learning," Computational and Biological Learning Lab, Department of Engineering, University of Cambridge Cambridge,, Cambridge, Tech. Rep., 2011.
- [6] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 498–519, 2001.
- [7] H.-A. Loeliger, "An Introduction to Factor Graphs," *IEEE Signal Processing Magazine*, no. January, pp. 28–41, 2004.
- [8] S. Kori, "A Factor Graph Approach to Signal Modelling," Ph.D. dissertation, Swiss Federal Institute of Technology, Zürich, Zurich, 2005.
- [9] M. F. Huber, "Recursive Gaussian process regression," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, May 2013, pp. 3362–3366. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6638281>
- [10] H. Leung, "No Title," 2010. [Online]. Available: <http://www.mathworks.com/matlabcentral/fileexchange/26607-sumproductlab-for-factor-graphs>
- [11] C. E. Rasmussen and H. Nickisch, "GPMPL toolbox." [Online]. Available: <http://www.gaussianprocess.org/gpml/code/matlab/doc/>

APPENDIX A LAPLACE INFERENCE

The framework is based on GP. By Bayes' rule the posterior can be inferred. However, at the moment the RM consist a classifier, the likelihood is non-Gaussian. There are various ways to approximate the posterior and we choose to do this with Laplace method. The method makes a second order Taylor expansion around the maximum of the posterior.

In the thesis the Laplace method is applied in two algorithms, normal GPBC and recursive GPBC. The largest difference is that normal GPBC recalculates the complete latent space and the recursive only calculates the latent variable that corresponds to the new experiment. This section will only give the derivation for the recursive case, since this prior is more complex.

By Bayes' rule the posterior over the new latent variable is

$$P(f_n | y_n, f_{n-1}^b) = \frac{P(y_n | f_n) P(f_n | f_{n-1}^b)}{P(y_n | f_{n-1}^b)} \quad (39)$$

The posterior cannot be solved analytically, since $P(y_n | f_n)$ is non-Gaussian. Therefore we apply the Laplace approximation to get a Gaussian distribution, $P(f_n | y_n, f_{n-1}^b) \approx \mathcal{N}(\mu_{lp}, \sigma_{lp}^2)$. The prior is given by

$$\begin{aligned} P(f_n | f_{n-1}^b) &= \mathcal{N}(\mu_n, D) \\ \text{Where } \mu_n &= J \cdot \mu_{n-1}^b \\ D &= k(x_n, x_n) - J \cdot k(X_b, x_n) \\ J &= k(x_n, X_b) \cdot K_b \end{aligned} \quad (40)$$

Since $P(y_n | f_{n-1}^b)$ is independent of f_n we can consider

the unnormalized logarithmic posterior to get the best fit.

$$\begin{aligned}\Psi(f_n) &\triangleq \log P(y_n | f_n) + \log P(f_n | f_{n-1}^b) \\ &= \log P(y_n | f_n) - \frac{1}{2}(\mu_{lp} - \mu_n)^\top D^{-1}(\mu_{lp} - \mu_n) \quad (41) \\ &\quad - \frac{1}{2} \log |K_n| - \frac{n}{2} \log 2\pi\end{aligned}$$

Differentiating $\Psi(f_n)$ with respect to f_n gives us Eq. (42) and Eq. (43).

$$\nabla_f \Psi(f_n) = \nabla_f \log P(y_n | f_n) - D^{-1}(\mu_{lp} - \mu_n) \quad (42)$$

$$\nabla_f^2 \Psi(f_n) = \nabla_f^2 \log P(y_n | f_n) - D^{-1} = -W_n - D^{-1} \quad (43)$$

In which weight matrix W_n is defined as $-\nabla_f^2 \log P(y_n | f_n)$. Setting the first derivative $\nabla_f \Psi(f_n)$ equal to 0 gives us the best fit.

$$\nabla_f \Psi(f_n) \stackrel{!}{=} 0 \rightarrow f_n = D(\nabla_f \log P(y_n | f_n)) \quad (44)$$

Using this condition $\Psi(f_n)$ will be found by Newton iterations,

$$\begin{aligned}(\mu_{lp}^{\text{new}} - \mu_n) &= (\mu_{lp}^{\text{old}} - \mu_n) - (\nabla_f^2 \Psi(f_n))^{-1} \nabla_f \Psi(f_n) \\ &= (\mu_{lp}^{\text{old}} - \mu_n) + (D^{-1} + W_n)^{-1} \cdot \\ &\quad (\nabla_f \log P(y_n | f_n) - D^{-1}(\mu_{lp}^{\text{old}} - \mu_n)) \quad (45) \\ &= (D^{-1} + W_n)^{-1} \cdot \\ &\quad (W_n(\mu_{lp}^{\text{old}} - \mu_n) + \nabla_f \log P(y_n | f_n))\end{aligned}$$

These iterations continue until $\Psi(f_n)$ is converged. Then we define the approximated posterior with the Gaussian distribution

$$Q(f_n | X_n, y^n) = \mathcal{N}(\mu_{lp}, (D^{-1} + W_n)^{-1}) \quad (46)$$

This distribution will be used as pseudo input for the correction of the basis vector.

APPENDIX B

ABBREVIATIONS AND SYMBOLS

A table with abbreviations is given in table III and the meaning of symbols in IV.

Table III
ABBREVIATIONS

Abbreviation	Meaning
AL	Active Learning
ED	Experiment Design
FFG	Forney-style Factor Graph
GP	Gaussian Process
GPBC	Gaussian Process Binary Classification
GPP	Gaussian Process Predictoin
GPPL	Gaussian Process Preference Learning
GPR	Gaussian Process Regression
MAP	Maximum A Posteriori
ML	Maximum Likelihood
MPA	Message Passing Algorithm
RM	Response Model
SPA	Summary-Propagation Algorithm

Table IV
SYMBOLS

Symbol	Meaning
\propto	Proportional to
\succ	Preferred over
\sim	distributed according to
B	$I + W_n^{-\frac{1}{2}} K_n W_n^{-\frac{1}{2}}$
cov	Covariance
diag	Places the elements on the diagonal of a matrix
exp	Exponential function
E	Expectation
$E_{Q(x)}[z(x)]$	expectation of $z(x)$ when $x \sim Q(x)$
D	$k(x_n, x_n) - J \cdot k(X_b, x_n)$
I	Identity matrix
f	GP Latent variable
J	$k(x_n, X_b)k(X_b, X_b)$
$k(x, x)$	Kernel function
K_n	$k(X_n, X_n)$, covariance matrix
μ	Mean
$\mathcal{N}(x \mu, \Sigma)$	Nominal Gaussian
∇	Partial derivative
∇^2	The Hessian matrix of second derivatives
$\Phi(z)$	Probit or cumulative unit of Gaussian, $\Phi(z) = (2\pi)^{-1/2} \int_{-\infty}^z \exp(-t^2/2) dt$
$\Psi(f^n)$	Log posterior over the latent function f^n
$P(y x)$	probability with conditional random variable y given x
$Q(f^n X_n, y^n)$	Approximation to the posterior $P(f^n X_n, y^n)$
$\sigma(z)$	Any sigmoid function
σ^2	Variance of a signal
Σ	Variance matrix
θ	Algorithm parameter for a hearing aid
t^\top	Transpose of vector t
W_n	$\text{diag}[-\nabla_{f_1}^2 \log P(y_1 f_1), \dots, -\nabla_{f_n}^2 \log P(y_n f_n)]$
X_n	Set of training points up till time step n
x_n	Experiment on time n
y	Target value