A tutorial on Gaussian process regression with a focus on exploration-exploitation scenarios

Eric Schulz*, Maarten Speekenbrink*, Andreas Krause**

Abstract

This tutorial introduces the reader to Gaussian process regression as a tool to model, actively explore and exploit unknown functions. Gaussian process regression is a powerful, non-parametric Bayesian approach towards regression problems that can be utilized in exploration and exploitation scenarios. This tutorial aims to provide an accessible introduction to these techniques. We will introduce Gaussian processes as a distribution over functions used for Bayesian non-parametric regression and demonstrate different applications of it. Didactic examples will include a simple regression problem, a demonstration of kernel-encoded prior assumptions, a pure exploration scenario within an optimal design framework, and a bandit-like exploration-exploitation scenario where the goal is to recommend movies. Beyond that, we describe a situation in which an additional constraint (not to sample below a certain threshold) needs to be accounted for and summarize recent psychological experiments utilizing Gaussian processes. Software and literature pointers will be provided.

Keywords: Gaussian process, Exploration-Exploitation, Bandit Problems

*Department of Experimental Psychology, University College London.
**Department of Computer Science, Swiss Federal Institute of Technology Zürich.

1. Introduction

No matter if we try to find a function that describes participants' behaviour (Cavagnaro, Aranovich, McClure, Pitt, and Myung, 2014), estimate parameters of psychological models (Wetzels, Vandekerckhove, Tuerlinckx, and Wagenmakers, 2010), try to sequentially optimize stimuli in an experiment (Myung and Pitt, 2009), or model how participants themselves learn to interact with their environment (Meder and Nelson, 2012), many problems require us to assess unknown functions that map inputs to outputs, $f : X \to Y$ (Mockus, 2010). Often times, the shape of the underlying function might be unknown, the function can be hard to evaluate analytically, or other requirements such as design costs might complicate the process of information acquisition. In these situations, Gaussian process regression can serve as a useful tool for performing regression both passively (for example, with the intent to predict newly observed points) as well as actively (for example, with the intent to minimize queries that lead to a pre-defined goal such as producing the highest possible output) (Williams and Rasmussen, 2006). Gaussian process regression is a non-parametric Bayesian approach (Gershman and Blei, 2012) towards regression problems. It can capture many different relations between inputs and outputs by utilizing a theoretically infinite number of parameters and letting the data decide upon the level of complexity through the means of Bayesian inference (Williams, 1998).

This tutorial will introduce Gaussian process regression as an approach towards modeling, actively learning and optimizing unknown functions. It is intended for a general readership and mostly contains practical examples and high level explanations. It consists of six main parts: The first part will intro-

duce the mathematical underpinnings of Gaussian process regression. The second part will show how different kernels encode various prior assumptions about the underlying function. Next, we will show how Gaussian processes can be used in problems of optimal experimental design, when the goal is pure exploration, i.e., to learn a function as well as possible. The fourth part will describe how Gaussian process-based Bayesian Optimization (here defined as an *exploration-exploitation problem*) works. In the fifth part, we will talk about ways of utilizing Gaussian process exploration-exploitation methods in situations with additional requirements and show one example where the goal is to avoid outputs that are below a certain threshold. We will conclude by summarizing current research that treats Gaussian process regression as a psychological model to assess function learning.

This tutorial attempts to provide an accessible and practical introduction to various applications of Gaussian process regression. As a tutorial like this can never be fully comprehensive, we have tried to provide detailed references and software pointers whenever possible. The tutorial will mainly focus on applications of Gaussian process-based methods.

2. Gaussian processes – distributions over functions

2.1. Motivation

Let f denote an (unknown) function which maps inputs x to outputs y: $f : X \to Y$ (Mockus, 2010). Throughout the following examples, we will have one of three different goals. *Modeling* a function f means mathematically representing the relation between inputs and outputs. An accurate model of f allows us to predict the output for many possible input values. In practice, this means generating accurate predictions for newly observed

points after some observations of both inputs and outputs have been collected. *Exploration* in this context means to actively choose the input points for which to observe the outputs in order to accurately model the function. In pure exploration problems, this is the only objective. In *exploration-exploitation* problems, we are concerned with obtaining the best possible outputs, and exploring the function serves the purpose of doing so most efficiently. In such active learning scenarios, the following 2 ingredients are needed:

1. A model of f that can be used to learn about its shape.
2. A method to select inputs based on the current knowledge of f.

Within *exploration* scenarios the next inputs are chosen in order to learn about the function as quickly and accurately as possible, whereas the intention in *exploration-exploitation* scenarios is to actively find those inputs that produce the best (e.g., highest) output values as quickly as possible in order to maximise the total reward accrued within a particular period of time. As a valid model of the underlying function f is crucial for all three goals, we will first focus on Gaussian processes as a powerful and expressive method to model unknown functions, before then applying this tool to exploration and exploitation scenarios later on.

2.2. Modeling functions: the weight space view

Let us start by considering a standard approach to model functions: linear regression (here defined as Bayesian regression). Imagine we have collected the observations shown in Table 1 and that we want to predict the value of y for a new input point $x_\star = 3$. In linear regression, we assume the

Table 1: Observations for the regression example. Inputs x_t and corresponding outputs y_t observed at times $t = 1, \ldots, 6$.

t	x_t	y_t
1	0.9	0.1
2	3.8	1.2
3	5.2	2.1
4	6.1	1.1
5	7.5	1.5
6	9.6	1.2

outputs are a linear function of the inputs with additional noise:

$$\begin{aligned} y_t &= f(x_t) + \epsilon_i \\ &= \beta_0 + \beta_1 x_t + \epsilon_t, \end{aligned}$$

where the noise term ϵ_t follows a normal distribution

$$\epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon^2)$$

with mean 0 and variance σ_ϵ^2. As this will be useful later, we can write this in matrix algebra as

$$y_t = \mathbf{x}_t^\top \mathbf{w} + \epsilon_i$$

defining the vectors

$$\mathbf{x}_t = \begin{bmatrix} 1 \\ x_t \end{bmatrix}, \qquad \mathbf{w} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

To predict the output for $x^\star$, we need to estimate the weights from the previous observations

$$\mathbf{X}_t = \begin{bmatrix} 1 & 0.9 \\ 1 & 3.8 \\ \vdots & \vdots \\ 1 & 9.6 \end{bmatrix}, \qquad \mathbf{y}_t = \begin{bmatrix} 0.1 \\ 1.2 \\ \vdots \\ 1.2 \end{bmatrix}.$$

Using a Bayesian framework, we do so through the posterior distribution over the weights. If we use a Gaussian prior over the weights $p(\mathbf{w}) = \mathcal{N}(0, \mathbf{\Sigma})$ and the Gaussian likelihood $p(\mathbf{y}_t|\mathbf{X_t}, \mathbf{w}) = \mathcal{N}(\mathbf{X_t}^\top \mathbf{w}, \sigma_\epsilon^{\mathbf{2}}\mathbf{I})$, then this posterior distribution is

$$\begin{aligned} p(\mathbf{w}|\mathbf{y}_t, \mathbf{X}_t) &\propto p(\mathbf{y}_t|\mathbf{X}_t, \mathbf{w})p(\mathbf{w}) \\ &= \mathcal{N}\left(\frac{1}{\sigma_\epsilon^2}\mathbf{A}_t^{-1}\mathbf{X}_t\mathbf{y}_t, \mathbf{A}_t^{-1}\right) \end{aligned} \tag{1}$$

where $\mathbf{A}_t = \mathbf{\Sigma}^{-1} + \sigma_\epsilon^{-2}\mathbf{X}_t\mathbf{X}_t^\top$. As inference here is performed over the weights (i.e., we try to find the best estimate for the β-weights given the data), this is also sometimes referred to as "the weight space view of regression". To predict the output $y_\star$ at a new test point $\mathbf{x}_\star$, we ignore the error term and focus on the expected value which is provided by the function f, predicting $f_\star = y_\star - \epsilon_\star = f(\mathbf{x}_\star)$. In the predictive distribution of $f_\star$, we average out our uncertainty regarding the weights

$$\begin{aligned} p(f_\star|\mathbf{x}_\star, \mathbf{X}_t, \mathbf{y}_t) &= \int p(f_\star|\mathbf{x}_\star, \mathbf{w})p(\mathbf{w}|\mathbf{X}_t, \mathbf{y}_t)d\mathbf{w} \\ &= \mathcal{N}\left(\frac{1}{\sigma_\epsilon^2}\mathbf{x}_\star^\top\mathbf{A}_t^{-1}\mathbf{X}_t\mathbf{y}_t, \mathbf{x}_\star^\top\mathbf{A}_t^{-1}\mathbf{x}_\star\right) \end{aligned} \tag{2}$$

A good prediction is the mean of this predictive distribution and comparing the mean to that in (1), we see that we can simply multiply the posterior

mean of $\mathbf{w}$ with the new input $\mathbf{x}_\star$, resulting in the prediction $0.56+3\times0.12 = 0.92$.

While linear regression is often chosen to model functions, it assumes the function has indeed a linear shape. However, only few relations in the real world are truly linear, and we need a way to model non-linear dependencies as well. One possible adjustment is to use a projection of the inputs $\mathbf{x}$ onto a "feature space" by using a function $\phi(\mathbf{x})$. A common projection is to use polynomials, resulting in polynomial regression. Take a cubic regression as an example, which assumes a function $f(x) = \beta_0+\beta_1 x+\beta_2 x^2+\beta_3 x^3$. Deriving the posterior for this model is similar to the linear regression described before, only that the input matrix $\mathbf{X}_t$ is replaced by the projection

$$\mathbf{\Phi}_t = \phi(\mathbf{X}_t) = \begin{bmatrix} 1 & 0.9 & 0.81 & 0.729 \\ 1 & 3.8 & 14.44 & 54.872 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 9.6 & 92.16 & 884.736 \end{bmatrix}.$$

In our example, this would result in the prediction $f_\star = -0.67 + 0.98 \times 3 - 0.13 \times 3^2 + 0.01 \times 3^3 = 1.37$.

Projecting input variables into a feature space offers considerable flexibility and allows one to model functions of any shape. However, this flexibility is also a drawback. There are infinitely many projections possible and we have to choose one either a priori or by model comparison within a set of possible projections. Especially if the problem is to explore and exploit a completely unknown function, this approach will not be beneficial as there is little guidance to which projections we should try. Gaussian process regression, which we will discuss next, offers a principled solution to this problem in which projections are chosen implicitly, effectively letting "the data de-

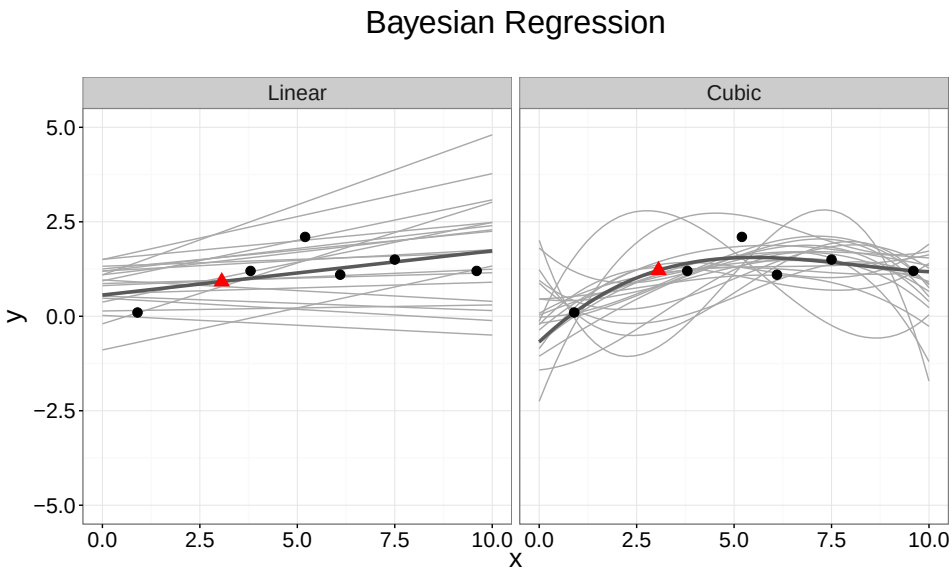

Figure 1: Example of performing Bayesian linear and cubic regression. Grey lines indicate predictions for different sampled posterior weights. Black dots mark empirical observations. Dark grey lines mark the current mean posterior predictions. The red triangle shows the prediction for a new data point $x_\star = 3$.

cide" on the complexity of the function.

2.3. Modeling functions: the function space view

In the weight space view of the previous section, we focused on distributions over weights. As each set of weights implies a particular function, a distribution over weights implies a distribution over functions. In Gaussian process regression, we focus on such distributions over functions directly. A Gaussian process defines a distribution over functions such that, if we pick any two points in a function (i.e. two different input-output pairs), observations of the outputs at these two points follow a joint (multivariate) Gaussian distribution. More formally, a Gaussian process is defined as a collection of random variables, any finite number of which have a joint

(multivariate) Gaussian distribution.

In Gaussian process regression, we assume the output y of a function f at input $\mathbf{x}$ can be written as

$$y = f(\mathbf{x}) + \epsilon \tag{3}$$

with $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$. Note that this is similar to the assumption made in linear regression, in that we assume an observation consists of an independent "signal" term $f(\mathbf{x})$ and "noise" term ϵ. New in Gaussian process regression, however, is that we assume that the signal term is also a random variable which follows a particular distribution. This distribution is subjective in the sense that the distribution reflects our uncertainty regarding the function. The uncertainty regarding f can be reduced by observing the output of the function at different input points. The noise term ϵ reflects the inherent randomness in the observations, which is always present no matter how many observations we make. In Gaussian process regression, we assume the function $f(\mathbf{x})$ is distributed as a Gaussian process:

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

A Gaussian process $\mathcal{GP}$ is a distribution over functions and is defined by a *mean* and a *covariance* function. The mean function $m(\mathbf{x})$ reflects the expected function value at input $\mathbf{x}$:

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

i.e. the average of all functions in the distribution evaluated at input $\mathbf{x}$. The prior mean function is often set to $m(\mathbf{x}) = 0$ in order to avoid expensive posterior computations and only do inference via the covariance directly. The covariance function $k(\mathbf{x}, \mathbf{x}')$ models the dependence between the function

values at different input points $\mathbf{x}$ and $\mathbf{x}'$:

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

The function k is commonly called the *kernel* of the Gaussian process (Jäkel, Schölkopf, and Wichmann, 2007). The choice of an appropriate kernel is based on assumptions such as smoothness and likely patterns to be expected in the data (more on this later). A sensible assumption is usually that the correlation between two points decays with the distance between the points according to a power function. This just means that closer points are expected to behave more similarly than points which are further away from each other. One very popular choice of a kernel fulfilling this assumption is the radial basis function kernel, which is defined as

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\lambda^2}\right).$$

The radial basis function provides an expressive kernel to model smooth functions. The two hyper-parameters λ (called the length-scale) and σ_f^2 (the signal variance) can be varied to increase or reduce the correlation between points and consequentially the smoothness of the resulting function.

Once a mean function and kernel are chosen, we can use the Gaussian process to draw a priori function values, as well as posterior function values conditional upon previous observations.

2.3.1. Sampling functions from a GP

Although Gaussian processes are continuous, sampling a function from a Gaussian process is generally done by selecting a set of input points. Theoretically, a function can be represented as a vector of infinite size; however, as we only have to make predictions for finitely many points in practice, we

can simply draw outputs for these points by using a multivariate normal distribution with a covariance matrix generated by the kernel. Let $\mathbf{X}_\star$ be a matrix with on each row a new input point $\mathbf{x}_i^\star$, $i = 1, \ldots, n$. To sample a function, we first compute the covariances between all inputs in $\mathbf{X}_\star$ and collect these in an $n \times n$ matrix:

$$K(\mathbf{X}_\star, \mathbf{X}_\star) = \begin{bmatrix} k(\mathbf{x}_1^\star, \mathbf{x}_1^\star) & k(\mathbf{x}_1^\star, \mathbf{x}_2^\star) & \ldots & k(\mathbf{x}_1^\star, \mathbf{x}_n^\star) \\ k(\mathbf{x}_2^\star, \mathbf{x}_1^\star) & k(\mathbf{x}_2^\star, \mathbf{x}_2^\star) & \ldots & k(\mathbf{x}_2^\star, \mathbf{x}_n^\star) \\ \vdots & \vdots & & \vdots \\ k(\mathbf{x}_n^\star, \mathbf{x}_1^\star) & k(\mathbf{x}_n^\star, \mathbf{x}_2^\star) & \ldots & k(\mathbf{x}_n^\star, \mathbf{x}_n^\star) \end{bmatrix}$$

Choosing the usual prior mean function $m(\mathbf{x}) = 0$, we can then sample values of f at inputs $\mathbf{X}_\star$ from the $\mathcal{GP}$ by sampling from a multivariate normal distribution

$$\mathbf{f}_\star \sim \mathcal{N}(\mathbf{0}, K(\mathbf{X}_\star, \mathbf{X}_\star))$$

where we use the notation $\mathbf{f}_\star = [f(\mathbf{x}_1^\star), \ldots, f(\mathbf{x}_n^\star)]^\top$. Note that $\mathbf{f}_\star$ is a sample of the function values. To sample observations $\mathbf{y}_\star$, we would have to add an additional and independent sample of the noise term ϵ.

2.3.2. Posterior predictions from a GP

Suppose we have collected observations $\mathcal{D}_t = \{\mathbf{X}_t, \mathbf{y}_t\}$ and we want to make predictions for new inputs $\mathbf{X}_\star$ by drawing $\mathbf{f}_\star$ from the posterior distribution $p(f|\mathcal{D}_t)$. By definition, previous observations $\mathbf{y}_t$ and function values $\mathbf{f}_\star$ follow a joint (multivariate) normal distribution. This distribution can be written as

$$\begin{bmatrix} \mathbf{y}_t \\ \mathbf{f}_\star \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{X}_t, \mathbf{X}_t) + \sigma_\epsilon^2 \mathbf{I} & K(\mathbf{X}_t, \mathbf{X}_\star) \\ K(\mathbf{X}_\star, \mathbf{X}_t) & K(\mathbf{X}_\star, \mathbf{X}_\star) \end{bmatrix}\right)$$

where $\mathbf{I}$ is an identity matrix (with 1's on the diagonal, and 0 elsewhere) and σ_ϵ^2 is the assumed noise level of observations (i.e. the variance of ϵ). Using standard results, the conditional distribution $p(\mathbf{f}_\star|\mathbf{X}_t, \mathbf{y}_t, \mathbf{X}_\star)$ is then a multivariate normal distribution with mean

$$K(\mathbf{X}_\star, \mathbf{X}_t)[K(\mathbf{X}_t, \mathbf{X}_t) + \sigma_\epsilon^2\mathbf{I}]^{-1}\mathbf{y}_t$$

and covariance matrix

$$K(\mathbf{X}_\star, \mathbf{X}_\star) - K(\mathbf{X}_\star, \mathbf{X}_t)[K(\mathbf{X}_t, \mathbf{X}_t) + \sigma_\epsilon^2\mathbf{I}]^{-1}K(\mathbf{X}_t, \mathbf{X}_\star)$$

Note that this posterior is also a GP with mean function

$$m_t(\mathbf{x}) = K(\mathbf{x}, \mathbf{X}_t)[K(\mathbf{X}_t, \mathbf{X}_t) + \sigma_\epsilon^2\mathbf{I}]^{-1}\mathbf{y}_t \tag{4}$$

and kernel

$$k_t(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}') - K(\mathbf{x}, \mathbf{X}_t)[K(\mathbf{X}_t, \mathbf{X}_t) + \sigma_\epsilon^2\mathbf{I}]^{-1}K(\mathbf{X}_t, \mathbf{x}') \tag{5}$$

To predict $\mathbf{f}_\star$, we can simply use the mean function in (4), or sample functions from the GP with this mean function and kernel (5) as described in the previous section.

Figure 2 shows an example of samples from a radial basis function GP prior and the posterior mean functions after the data in Table 1 has been observed.

2.3.3. Switching back to the weight view

We can rewrite the mean function (4) as

$$m_t(\mathbf{x}) = \sum_{i=1}^{t} w_i k(\mathbf{x}_i, \mathbf{x})$$

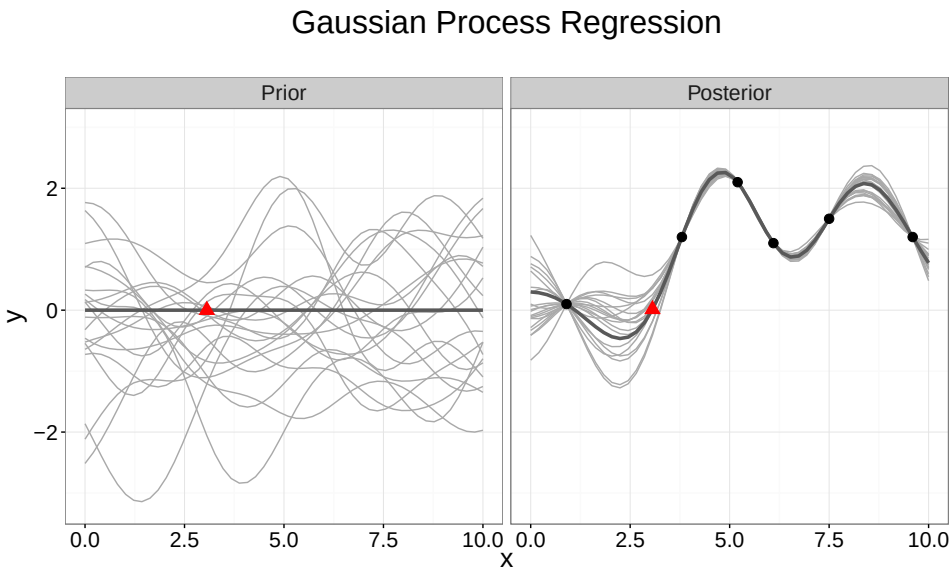

Figure 2: Example of samples from a Gaussian process prior and posterior. Grey lines indicate samples from the GP. Black dots mark empirical observations. The dark grey line marks the current mean of the GP. The red triangle shows the prediction for the new input point.

where each $\mathbf{x}_i$ is a previously observed input value in $\mathbf{X}_t$ and the weights are collected in the vector $\mathbf{w} = \left(K(\mathbf{X}_t, \mathbf{X}_t) + \sigma_\epsilon^2 \mathbf{I}\right)^{-1} \mathbf{y}_t$. What this equation tells us is that Gaussian process regression is equivalent to a linear regression model using basis functions k to project the inputs into a feature space. To make new predictions, every output y_t is weighted by how similar the corresponding input $\mathbf{x}_t$ was to the to be predicted point $\mathbf{x}$ by a similarity measure induced by the kernel. This results in a simple weighted sum to make predictions for new points[1]. The posterior predictive mean then is a

[1]In fact, simple Bayesian linear regression can be recovered by using a linear kernel $k(x, x') = \sigma_b^2 + \sigma_f^2 (x - c)(x' - c)$

linear combination of the features. Therefore, a conceptually infinite parameter space boils down to a finite sum when making predictions. This sum only depends on the chosen kernel k and the data $\mathcal{D}_t$ observed thus far (Kac and Siegert, 1947). This is why Gaussian process regression is referred to as a non-parametric technique. It is not the case that this regression approach has no parameters. Actually, it has theoretically as many parameters w as there are observations. However, in making predictions, we only use a finite sum over all past observations. Details for generating a prediction for $x_\star = 3$ given a radial basis function kernel with length scale $\lambda = 1$, and observation variance $\sigma_\epsilon^2 = 0.01$ are provided in Table 3.

Table 2: Example of generating a prediction using a Gaussian process with a radial basis function kernel. $w_i = \left(K(X,X) + \sigma_\epsilon^2 I\right)^{-1} y_i$; $x_\star$=3;

t	x_t	y_t	w_t	$k(x_t, x_\star)$	$w_t k(x_t, x_\star)$
1	0.9	0.1	0.51	0.38	0.19
2	3.8	1.2	-3.88	0.87	-3.37
3	5.2	2.1	13.3	0.34	4.53
4	6.1	1.1	-12.55	0.12	-1.48
5	7.5	1.5	5.83	0.01	0.06
6	9.6	1.2	-0.34	0.00	0.00
				$\sum_{t=1}^{6} w_t k(x_t, x_\star)$:	-0.06

We can see that Gaussian process regression is a powerful tool to capture many stationary functions. This in turn can be easily applied to contexts where the task is to explore or exploit these functions actively, in a step-wise fashion.

2.4. Optimizing hyper-parameters

The kernel normally contains hyper-parameters such as the length-scale, the signal variance, and the noise variance. These are usually not assumed to be known but rather are learned from the data. As the posterior distribution over the hyper-parameters is generally difficult to obtain, full Bayesian inference of the hyper-parameters is generally not used. Instead, a point estimate of the hyper-parameters is usually computed by maximising the marginal (log) likelihood. This is similar to parameter estimation by maximum likelihood and is also referred to as type-II maximum likelihood (ML-II, cf Williams and Rasmussen, 2006).

Given the data $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ and hyper-parameters θ, the log marginal likelihood is

$$\log p(\mathbf{y}|\mathbf{X}, \theta) = -\frac{1}{2}\mathbf{y}^\top \mathbf{K}_y^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}_y| - \frac{n}{2}\log 2\pi \tag{6}$$

where $\mathbf{K}_y = K(\mathbf{X}, \mathbf{X}) + \sigma_\epsilon^2 \mathbf{I}$ is the covariance matrix of the noisy output values y. The marginal log likelihood can be viewed as a penalized fit measure, where the term $-\frac{1}{2}\mathbf{y}^\top \mathbf{K}_y^{-1}\mathbf{y}$ measures the data fit, that is how well the current kernel parametrization explains the dependent variable, and $-\frac{1}{2}\log|\mathbf{K}_y|$ is a complexity penalization term. The final term $-\frac{n}{2}\log 2\pi$ is a normalization constant. The marginal likelihood is normally maximized through a gradient-ascent based optimization tool such as implemented in MATLAB's `minimize.m` function. These routines make use of the partial derivatives w.r.t. θ:

$$\frac{\partial}{\partial \theta_j}\log p(\mathbf{y}|X, \theta) = \frac{1}{2}\mathbf{y}^\top K^{-1}\mathbf{y} - \frac{1}{2}\operatorname{tr}\left(K^{-1}\frac{\partial K}{\partial \theta_j}\right) \tag{7}$$

$$= \frac{1}{2}\operatorname{tr}\left((\alpha\alpha^\top - K^{-1})\frac{\partial K}{\partial \theta_j}\right) \tag{8}$$

with $\alpha = K^{-1}\mathbf{y}$.

3. Encoding prior assumptions via the kernel

So far we have only focused on the radial basis function kernel to perform Gaussian process inference. However, other kernels are possible and flexibility in choosing the kernel is one of the benefits of Gaussian process regression. The kernel function k directly encodes prior assumptions about the underlying function such as its smoothness and periodicity. Additionally, more complex kernels can be created by combining simpler kernels through addition or multiplication.

3.1. Encoding smoothness

The radial basis function kernel is a special case of a general class of kernel functions called the Matérn kernel. The Matérn covariance between two points with distance $\tau = |\mathbf{x} - \mathbf{x}'|$ is

$$k_\nu(\tau) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{\tau}{\rho}\right) K_\nu \left(\sqrt{2\nu}\frac{\tau}{\rho}\right) \tag{9}$$

where Γ is the gamma function, K_v is the modified Bessel function of the second kind, and ρ and ν are non-negative covariance parameters. A $\mathcal{GP}$ with a Matérn covariance has sample paths that are $\nu - 1$ times differentiable. When $\nu = p + 0.5$, the Matérn kernel can be written as a product of an exponential and a polynomial of order p.

$$k_{p+0.5}(\tau) = \sigma^2 \exp\left(-\frac{\sqrt{2\nu}\tau}{\rho}\right) \frac{\Gamma(p+1)}{(2p+1)} \times \sum_{i=0}^{p} \frac{(p+i)!}{i!(p-i)!} \left(\frac{\sqrt{8\nu}\tau}{\rho}\right)^{p-i} \tag{10}$$

Here, p directly determines how quickly the covariance between two points thins out in dependency of how far the two points are apart. If $p = 0$, then this leads to the Ornstein-Uhlenbeck process kernel

$$k(\tau) = \sigma_f^2 \exp\left(-\frac{\tau}{\lambda}\right), \tag{11}$$

which encodes the prior assumption that the function is extremely unsmooth (rough) and that observations do not provide a lot of information about points that are anything but very close to the points we have observed so far. If $p \to \infty$ in the limit, then the Matérn kernel becomes a radial basis function kernel. This kernel expects very smooth functions for which observing one point provides considerably more information than if we assume very rough underlying functions. Figure 3 shows prior and posterior samples for both the Ornstein-Uhlenbeck process and the radial basis function kernel. Notice how the prior samples are a lot more "rugged" for the former and very smooth for the later. We can also see how encoding different prior smoothness assumptions leads to different posterior samples after having observed the same set of points (the points we used before). In particular, expecting very rough functions a priori leads to posteriors that do not generalize far beyond the encountered observations, whereas expecting very smooth functions leads to posterior samples that generalize more broadly beyond the encountered points.

In most real world applications of Gaussian process regression, the practitioner either chooses the radial basis function kernel and then optimizes its length-scale in order to account for potential mismatches between prior smoothness assumptions and the observed functions or again optimizes the length-scale parameter but a priori chooses a Matérn kernel with $p = 5$ as an intermediate solution to encode the expectation of smooth but not too smooth functions. In general, it seems always better to think hard about the expected properties of the underlying function in order to avoid ill-posed priors (Schulz, Speekenbrink, Hernández-Lobato, Ghahramani, and Gershman, 2016c).

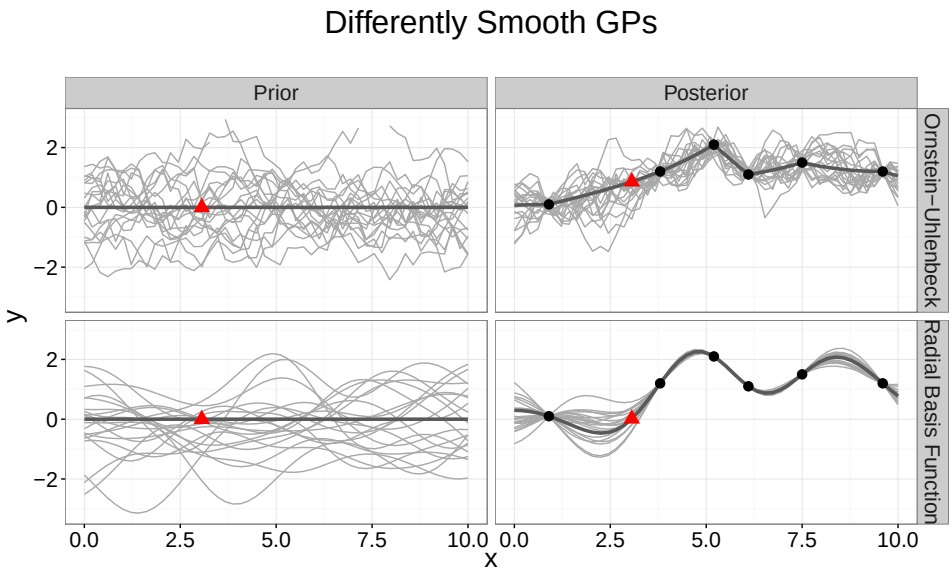

Figure 3: Example of samples from differently smooth Gaussian process priors and their posteriors after having observed the same set of points. Grey lines indicate samples from the GP. Black dots mark empirical observations. The dark grey line marks the current mean of the GP. The red triangle shows the prediction for the new data point.

3.2. Composing kernels

Another advantage of Gaussian process regression is that different kernels can be combined, thereby creating a rich set of interpretable and reusable building blocks (Duvenaud, Lloyd, Grosse, Tenenbaum, and Ghahramani, 2013). For example, adding two kernels together models the data as a superposition of independent functions. Multiplying a kernel with for example the radial basis function kernel, locally smoothes the effect of the first kernel.

Take as an example data set the atmospheric concentration of carbon dioxide over a forty year horizon as shown in Figure 4. We can immediately see a pattern within this data, which is that the CO2-concentration seems to increase over the years, that there seems to be some periodicity by

which at some times within each year the CO2 emission is higher, and that this period may not be perfectly replicated every year. Using a Gaussian process regression framework, we can combine different kernels as building blocks in the attempt to explain these patterns. Figure 4 shows posterior mean predictions for different kernel combinations. The first one shows a

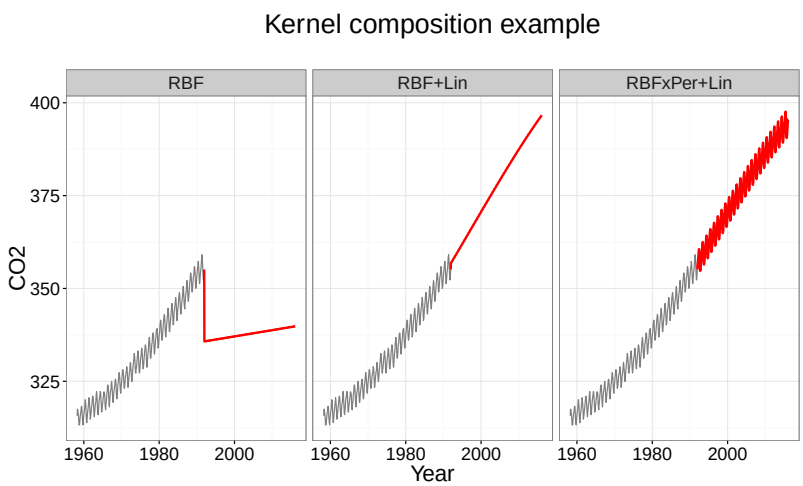

Figure 4: Example of composing kernels by combining simpler kernels in order to explain a complex function. Data were mean-centred before fitting the Gaussian process and predictions were transformed back afterwards. Grey lines show observed CO2 emissions. Red lines show posterior predictions of Gaussian process regressions with different kernels: RBF is a radial basis function kernel, RBF+Lin is a kernel composed by adding a RBF and a linear kernel, RBF×Per + Lin is a kernel composed by multiplying a radial basis and periodic kernel and adding a linear kernel.

radial basis function alone, the second a sum of a radial basis function kernel and a linear kernel, $k(x, x') = (x - c)(x' - c)$, and the third one the sum between a linear kernel and the product between a radial basis function kernel and a periodic kernel, $k(x, x') = \theta_1^2 \exp\left(-\frac{2\sin^2(\pi|x-x'|\theta_2)}{\lambda^2}\right)$. As the radial basis function kernel tends to reverse back to the mean over time, it

does not do a good job capturing the linear trend of the data. Therefore, adding a linear kernel to the radial basis function kernel already seems to improve predictions. Finally, multiplying the radial basis function kernel with a periodic kernel to create a locally smoothed periodic kernel, which is then combined with an increasing trend by adding a linear kernel seems to predict the data reasonably well. This shows that the kernel can also be used to encode structural assumptions about the underlying function more explicitly, especially when one wants to cover more complex patterns than just interpolating smooth functions (see Lloyd, Duvenaud, Grosse, Tenenbaum, and Ghahramani, 2014, for an exciting application of compositional kernels).

4. General set-up for exploration-exploitation problems

Having found a general way to model functions, we can now focus on ways to cleverly explore or exploit the unknown function as we are learning about it over time. Within the Gaussian process approach both pure *exploration* and *exploration-exploitation* can be treated in a similar manner. Both use Gaussian process regression to model the underlying function[2] and estimate the utility of available queries (candidate input points to sample next) through what is called an *acquisition function*. An acquisition function V can be seen as measuring the usefulness (or utility) of candidate input points in terms of allowing one to learn the function as best as possible (exploration) or producing the best possible output (exploitation). The approach then goes on to choose as the next input the one that promises to

[2]In this context, a Gaussian process regression is sometimes also referred to as a "surrogate model" (see Gramacy and Lee, 2008).

produce the highest utility. The way this works is shown in Algorithm 1.

Algorithm 1 General $\mathcal{GP}$ optimization algorithm

Require: Input space $\mathcal{X}$; acquisition function V_t; $\mathcal{GP}$-prior for f with mean function $m(\mathbf{x})$ and kernel $k(\mathbf{x}, \mathbf{x}')$

for $t = 1, 2, \ldots$ **do**

  Choose $\mathbf{x}_t^\star = \arg\max_{\mathbf{x} \in \mathcal{X}} V_t(\mathbf{x})$

  Sample $y_t = f(\mathbf{x}_t^\star) + \epsilon_t$

end for

This algorithm starts out with a Gaussian process distribution over functions, then assesses the usefulness of the available samples by utilizing the acquisition function and selects the point that currently maximizes this function. The value of the utility function $V_t(\mathbf{x})$ thereby always depends on the current posterior of the Gaussian process at time point t (it can change on every trial). Afterwards, the new output at the chosen sample point is observed, the Gaussian process is updated, and the process starts anew. As in this setting the function is always modelled as interpolation within a given candidate set of points, we will use a simple radial basis function kernel to model unknown functions for all of the remaining examples.

5. Gaussian process exploration

The goal in a pure exploration setting is to learn an unknown function as accurately and quickly as possible. In a psychological setting this could mean for example to try and find out what a participant-specific forgetting function might look like and designing stimuli adaptively in order to optimally learn about this function on each subsequent trial of an experiment (e.g., Myung, Cavagnaro, and Pitt, 2013).

5.1. Acquisition function

In the current setting, learning about a function means that the posterior distribution becomes more certain (e.g., less dispersed). A useful measure of the uncertainty about a random variable Y with probability distribution p is the differential entropy

$$H(Y) = -\int p(y) \log p(y) \, \mathrm{d}y = \mathbb{E}[\log p(Y)]$$

The information that an input x provides about the random variable, which we call the information gain, is the (expected) reduction in entropy

$$I(Y; x) = H(Y) - H(Y|x) = -\int p(y) \log p(y) + p(y, x) \log p(y, x) \, \mathrm{d}y$$

For example, if Y follows a d-variate Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, then the entropy is

$$H(Y) = \frac{1}{2} \log (2\pi e)^d |\boldsymbol{\Sigma}|$$

In our setting, we want to learn about the function, i.e. reduce the entropy in the distribution $p(f)$. We can write the information gain as

$$I(f; \mathbf{y}) = \frac{1}{2} \log |I + \sigma^{-2} K|, \tag{12}$$

where $K = [k(x, x')]$. Even though finding the overall information gain maximizer is NP-hard, it can be approximated by an efficient greedy algorithm based on Gaussian process regression. If $F(A) = I(f; \mathbf{y}_A)$, then this algorithm picks $x_t = \arg\max F(A_{t-1} \cup \{\mathbf{x}\})$, that is greedily querying the point whose predicted output is currently most uncertain.

$$V_t(\mathbf{x}) = \arg\max \; s_{t-1}(\mathbf{x}) \tag{13}$$

where $s_t(\mathbf{x}) = k_{t-1}(x, x)$ is the variance of f at input $\mathbf{x}$.

This algorithm starts with a Gaussian process prior for f and at each time $t = 1, \ldots, T$, sequentially samples those input points where the current posterior predictive distribution $p(f|\mathcal{D}_{t-1})$ evaluated at $\mathbf{x}$ shows the highest variance, i.e. the highest predictive uncertainty. This is a "greedy" algorithm in the sense that it focuses on minimizing the current uncertainty, rather than looking further ahead into the future. Even though this algorithm, sometimes also called uncertainty sampling in statistics, looks naïve at first, it can actually be shown to obtain at least a constant fraction of the maximum information gain reachable using at most T samples (see Krause, Singh, and Guestrin, 2008, for more details):

$$F(A_T) \geq \left(1 - \frac{1}{e}\right) \max_{|A| \subseteq T} F(A) \tag{14}$$

where $F(A_T)$ measures the information about f at time point t. This is based on two properties of the acquisition function called *submodularity* and *monotonicity* (Krause and Golovin, 2012). Intuitively, submodularity here corresponds to a diminishing returns property of the acquisition function by which newly sampled points will add less and less information about the underlying function. Montonicity means that information never hurts (it is always helpful to observe more points). Both properties are crucial to show that the greedy algorithm can be successful. A simple example of the Gaussian process uncertainty reduction sampler is shown in Figure 5 below. We have used the same set of observations as before and let the algorithm select a new observation by picking as the next observation the one that currently has the highest predictive uncertainty attached.

5.2. Example: Learning unknown functions

In order to demonstrate how Gaussian process-based exploration works, we will show how the algorithm learns a set of unknown functions and

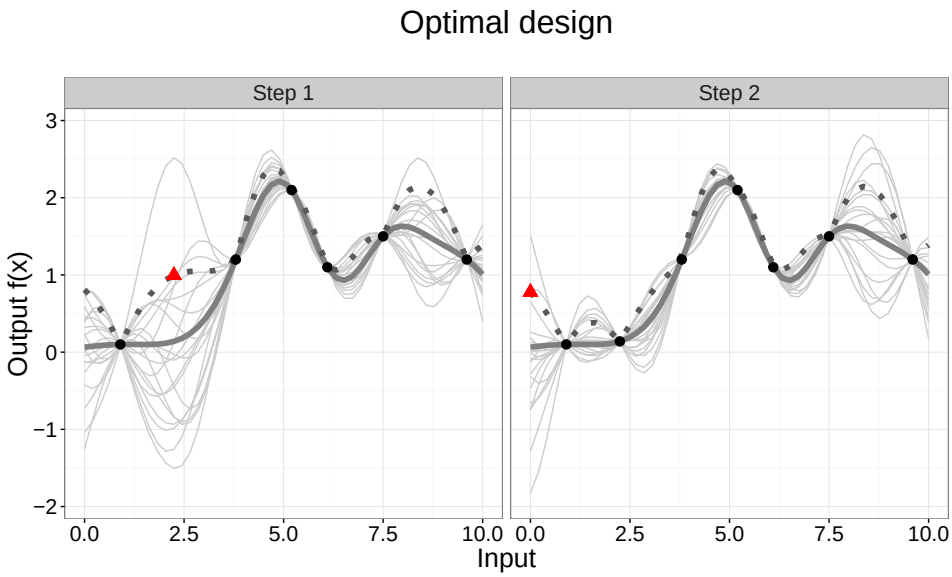

Figure 5: GP-uncertainty reduction example. The dark grey line marks the current mean of the GP. The dashed line shows the mean plus the standard deviation. The light grey lines are samples from the GP. The red triangle marks the current candidate point with the highest attached uncertainty.

compare it to other algorithms. The objective is to learn an unknown function as quickly and accurately as possible. For simplicity, we will focus on a function f which takes a one-dimensional and discretized input $x \in [0, 0.01, 0.02, \ldots, 10]$ and to which it maps an output y.

As GP regression is considered to learn a plurality of different functions well, we will test the algorithm on a number of different functions that are frequently encountered in psychology: a linear, quadratic, cubic, logarithmic, sine, and a non-stationary[3] function. The functions are summarized in

[3]A non-stationary function for our purpose is a function that changes its parametric form over different parts of the input space.

Table 3.

Table 3: Functions used in the Gaussian process exploration simulation.

Function	**Equation**
linear	$f(x) = x$
quadratic	$f(x) = x^2 + x$
cubic	$f(x) = x^3 - x^2 + x$
sine	$f(x) = x \times \sin(x)$
non-stationary	$f(x) = \begin{cases} \sin(\pi x) + \cos(\pi x), & \text{if } x < 8 \\ x, & \text{otherwise} \end{cases}$

In addition to a GP regression model, we also used models that explicitly assume parametric forms of the function. These latter models learn the parameters (the weights) defining the function directly and were defined as a Gaussian process with a polynomial kernel with fixed degrees of freedom, i.e. performing Bayesian linear regression. Each model was set up to learn the underlying function by picking as the next observation the one that currently has the highest uncertainty (standard deviation of the predicted mean).

We let each model run 100 times over 40 trials (on each trial optimizing the hyper-parameters) for each underlying function and averaged the mean squared error over the whole discretized input space for each step. We tested two different versions of learning the underlying functions with a Gaussian process regression, one which selected input points at random (GP-Random), and the uncertainty reduction sampler described above, which learns actively by choosing input points based on their predictive variance (GP-Active). Results are shown in Figure 6.

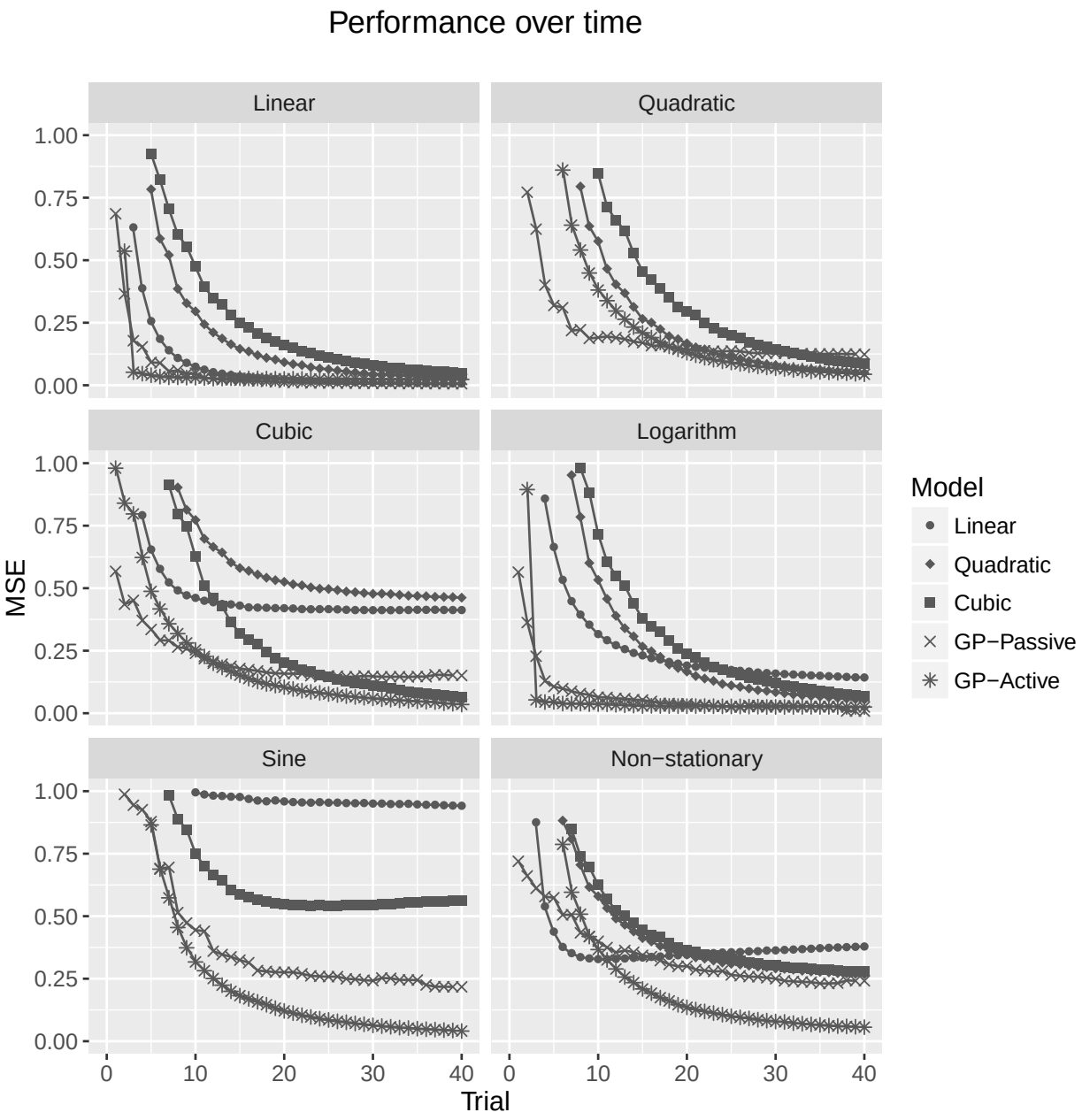

Figure 6: GP-uncertainty reduction example. GP-produced error always goes down.

It can be seen that the Gaussian process model learns all functions efficiently and well. Even when the inputs are sampled at random, the error always goes down for a Gaussian process regression. However, the error generally goes down faster when inputs are selected actively. Only in the cases in which the used learning function is indeed the same as the underlying function (for example, using a linear function to learn an underlying linear function), does another model occasionally learn better than the Gaussian process. In some cases, using a cubic Bayesian regression seems to result into over-fitting of the underlying function which leads to the overall error going up again. Overall, the results indicate that Gaussian process regression is especially useful in cases where the underlying function is not known. For example, one could easily use Gaussian process regression to learn participants' utility function over different experiments or simply use them to generate stimuli that are very informative overall.

6. Exploration-Exploitation and Bayesian Optimization

In an exploration-exploitation scenario the goal is to find the input to a function that produces the maximum output as quickly as possible.

$$\mathbf{x}^* = \arg\max_{\mathbf{x} \in D} f(\mathbf{x}) \tag{15}$$

where $\mathbf{x}^*$ is the input that produces the highest output. One way to measure the quality of this search process is to quantify regret. Regret is the difference between the output of the currently chosen argument and the best output possible

$$r(\mathbf{x}) = f(\mathbf{x}^*) - f(\mathbf{x}). \tag{16}$$

The total regret is the sum of the regret over all trials, and the goal in an exploration-exploitation scenarios is to minimize the cumulative regret:

$$R_t = \sum_{u=1}^{t} r(x_u) \tag{17}$$

Again, finding the strategy that chooses the inputs to minimize the expected cumulative regret is NP-hard. That is, determining the sequence of queries (i.e. input choices) that leads to the lowest total regret is impossible for all but the most trivial cases. However, there is again a greedy trick one can apply in this scenario, which starts by reinterpreting the function maximization – or regret minimization – problem as a multi-armed bandit task (cf Steyvers, Lee, and Wagenmakers, 2009). In a bandit task there are multiple options (arms) with unknown probability of producing a reward and the goal is to choose the best arm in order to maximise the overall reward (the name stems from the one armed bandits that can be found in casinos). In the current situation, we can view the discretized input points as the arms of a multi-armed bandit, and the output of the function at those points as the unknown rewards that are associated to each arm. What distinguishes the current situation from traditional bandit tasks is that the rewards of the arms are correlated in dependency of the underlying covariance kernel. Nevertheless, viewing the task as a multi-armed bandit allows us to use strategies that have been devised for traditional bandit tasks. One popular strategy is called the upper confidence bound (UCB) algorithm, which relies on the following acquisition function:

$$V_t(\mathbf{x}) = m_{t-1}(\mathbf{x}) + \omega\sqrt{s_{t-1}(\mathbf{x})}, \tag{18}$$

where ω is a free parameter that determines the width of the confidence interval and $\sqrt{s_{t-1}(\mathbf{x})}$ is the predictive standard deviation at a point $\mathbf{x}$. For

example, setting $\omega = 1.96$, results in a 95% confidence interval for a single value $\mathbf{x}$ given a Gaussian distribution.

The UCB algorithm chooses the arm for which the upper confidence bound is currently the highest. The upper confidence bound is determined by two factors: the current estimate of the mean of f at a particular point (the higher the estimate, the higher the bound) and the uncertainty attached to that estimate (the higher the uncertainty, the higher the bound). Therefore, the UCB algorithm trades off naturally between expectation and uncertainty. An example of how the UCB algorithm works, using the same data as before, is shown in Figure 7.

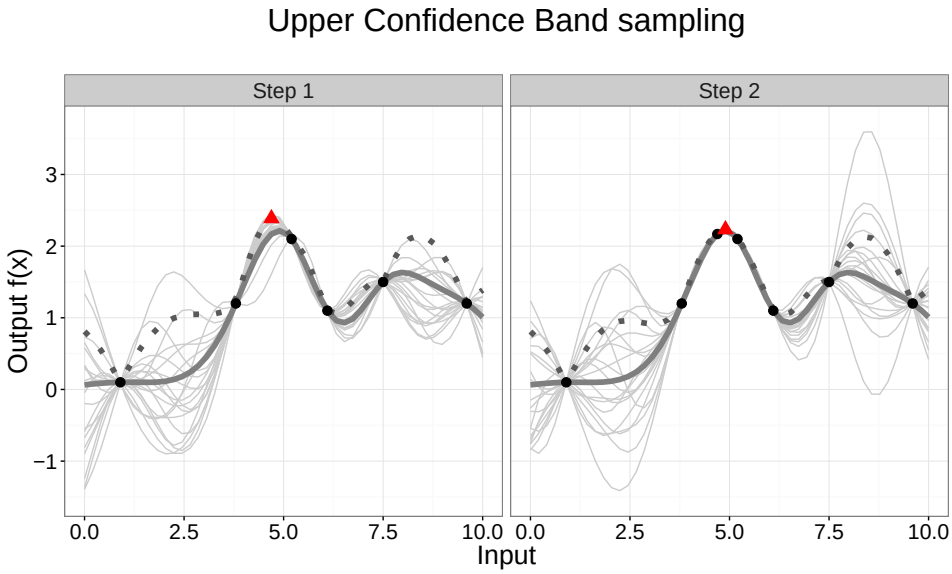

Figure 7: GP-UCB example. The dark grey line marks the current mean of the GP. The dashed line marks the GP's upper confidence bound. The light grey lines are samples from the GP. The red triangle marks the point that currently produces the highest UCB.

Even though the greedy UCB strategy is naïve, it can be shown that its regret is sublinear, again using an argument that relies on the submodularity

and monotonicity of the overall information gain (Srinivas, Krause, Kakade, and Seeger, 2009). Sublinear regret here just means that the regret per round goes down in expectation, thereby guaranteeing that the algorithm picks better points over time.

6.1. GP-UCB Example: Recommending movies

As an example of applying GP-UCB to exploration-exploitation problems, we will use it in an movie recommendation scenario, where the task is to recommend the best movies possible to a user with unknown preferences. This involves both learning how different features of movies influence the liking of a movie and recommending the movies that will be liked the most. For this application, we sampled 5141 movies from the IMDb database and recorded their features such as the year they appeared, the budget that was used to make them, their length, as well as how many people had evaluated the movie on the platform, number of facebook likes of different actors within the movie, genre of the movie, etc. As a proxy for how much the person would enjoy the movie, we used the average IMDb score, which is based on the ratings of registered users. As there were 27 features in total, we performed a Principal Component Analysis extracting 8 components that together explained more than 90% of the variance within the feature sets. These components were then used as an input for the optimization routine. We used a GP-UCB with a radial basis function kernel, set $\omega = 3$ in the UCB acquisition function, initialized the GP with 5 randomly sampled observations, and then let the algorithm pick 20 movies sequentially. This procedure was repeated 50 times. Even though recommender systems normally try to recommend the best movie for a particular user, this approach can be seen as recommending movies to an average user.

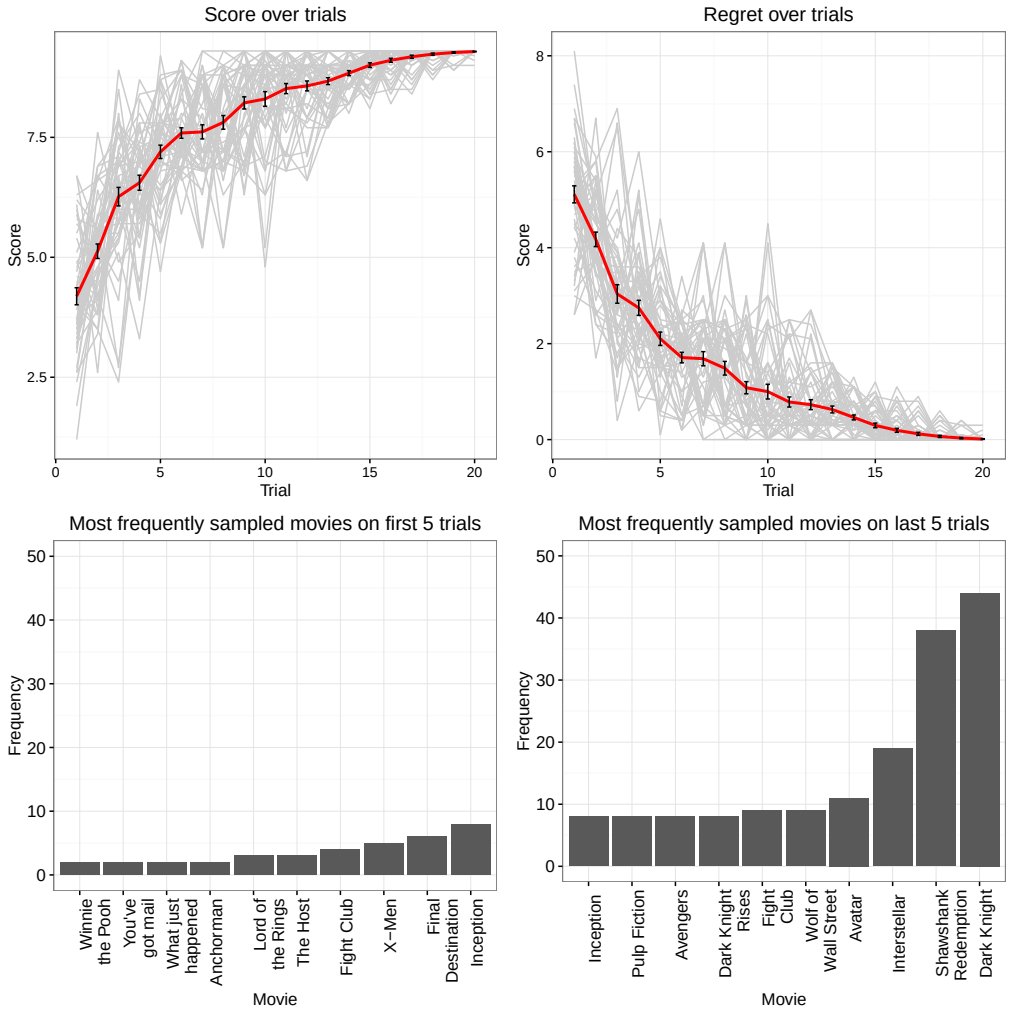

Figure 8: Recommending movies with a GP-UCB algorithm. The score (upper left, errorbars represent the standard error of the mean) goes up over all runs and plateaus very quickly at around the highest value possible (9.3). Vice versa, the overall regret (upper right) goes down over trials an quickly approaches 0. Within the first 5 samples, movies are mostly picked at random and no clear pattern of movies seems to dominate (bottom right). However, within the last 5 trials GP-UCB preferentially samples highly rated movies (bottom right).

Results are shown in Figure 8. It can be seen that the algorithm quickly starts choosing movies that produce high scores which results in the overall mean score to go up and the regret to go down over time. Moreover, the variance of the picked movies also goes down over time as GP-UCB almost exclusively samples highly rated movies later on. Whereas the 10 most frequently sampled movies within the first 5 samples seem to be sampled closely to random, the most frequently sampled movies within the last 5 trials are movies that are generally highly rated. In the end, the algorithm has explored the input space well, learned the unknown preference function of the average user rather well, and returned movies that are on average highly rated. When we let the GP-UCB algorithm run over 200 trials in another, it started only sampling the movie "The Shawshank Redemption", which is the highest rated movie on the internet movie database.

7. Safe exploration-exploitation

Sometimes an exploration-exploitation scenario may come with additional requirements. For example, an additional requirement can be to avoid certain outputs as much as possible. Consider excitatory stimulation treatment, where the task is to stimulate the spinal chord in such a way that certain movements are achieved (Desautels, Choe, Gad, Nandra, Roy, Zhong, Tai, Edgerton, and Burdick, 2015). Here, it is important to stimulate the spinal chord such that optimal recovery is achieved, but not too much as this might lead to painful reactions within the patients.

Again, Gaussian process optimization methods can be used to learn the underlying function of what stimulation leads to which strength of reaction. However, an additional requirement now could be to avoid particu-

larly reactions that result in pain. An algorithm that balances exploration and exploitation whilst avoiding certain outputs is called Safe Optimization (Sui, Gotovos, Burdick, and Krause, 2015). This algorithm adapts the Upper Confidence Bound approach described earlier to accommodate this additional requirement. It works by trading-off two different goals: Firstly, it keeps track of a set of safe options it considers to be above a given safe threshold (points currently showing a high likelihood of being above the threshold) and tries to expand this set as much as it can. Secondly, it maintains a set of potential maximizers (points of likely to produce high outcomes) that, if used as an input, would potentially achieve the highest output. It then chooses as the next input a point within the intersection of these two sets, that is a safe point that is either a maximizer or an expander that has the highest predictive variance and potentially expands the set of maximisers. This algorithm can also be set up to separate the optimization routine from a given set of constraints as described by (Berkenkamp, Krause, and Schoellig, 2016). For technical details, we refer the interested reader to Sui et al. (2015).

7.1. Example: Cautious stimulus optimization

As an illustration of the Safe Optimization algorithm, we apply it to a situation in which the objective is to choose inputs $\mathbf{x}$ in order to learn about the underlying function in a two-dimensional space such that –eventually– points that produce high outputs in y will be sampled whilst avoiding to choose inputs that produce an output below 0. To simplify presentation, we sampled the underlying function from a Gaussian process parameterized by a radial basis function kernel. This can be seen as similar to the case where one wants to present stimuli to participants, but make sure that participants

never react with an intensity below a certain threshold.

Results are shown in Figure 9. It can be seen that the Safe Optimization algorithm explores the function exceptionally well in its attempt to expand the space of possible safe inputs. At the same time, the algorithm does not at any time choose inputs from the white area (producing output values below 0). This algorithm could be applied to optimal design settings that require additional constraints.

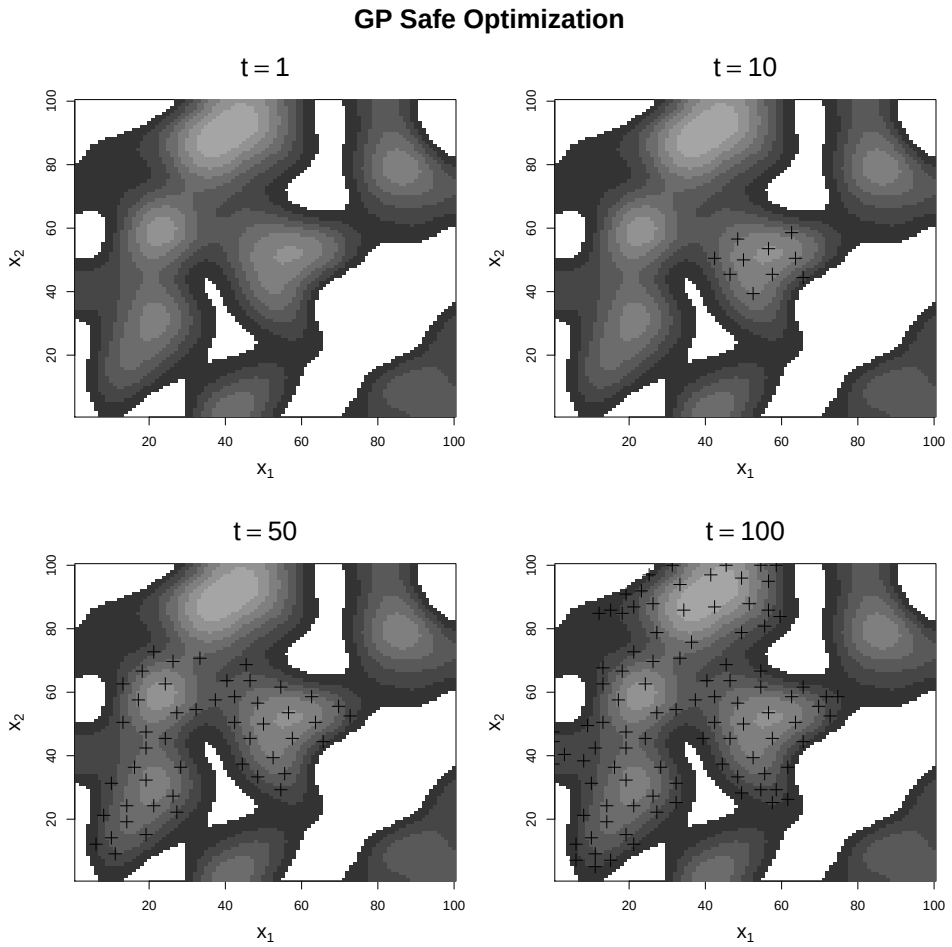

Figure 9: GP-Safe Optimization example showing samples after 1,10, 50 and 100 samples. White represents areas below 0. The black crosses show where the Safe Optimization algorithm has sampled. Lighter areas represent higher scores. The algorithm efficiently explores other safe areas. It never samples points within the surrounding white area as these are below the threshold.

8. Gaussian processes and cognition

We have seen that Gaussian process regression is a powerful tool to model, explore, and exploit unknown functions. However, Gaussian process regression might also be applied in a different, more psychological context, namely as an actual model for human cognition in general and function learning in particular. Recently, Lucas, Griffiths, Williams, and Kalish (2015) have proposed to use Gaussian process regression as a rational model of function learning that can explain various effects within the literature of human function learning. Schulz, Tenenbaum, Reshef, Speekenbrink, and Gershman (2015) used Gaussian processes to assess participants' judgements of the predictability of functions in dependency of the smoothness of the underlying kernel. As many different kernels can be used to model function learning, Wilson, Dann, Lucas, and Xing (2015) tried to infer backwards what the human kernel might look like by using a non-parametric kernel approach to Gaussian process regression. As explained above, kernels can also be added together and multiplied to build more expressive kernels, which led Schulz, Tenenbaum, Duvenaud, Speekenbrink, and Gershman (2016d) to assess if participants' functional inductive biases can be described as made up of compositional building blocks. In a slightly different context, Gershman, Malmaud, and Tenenbaum (2016) modeled participants' utility of combinations of different objects by a Gaussian process parametrized by a tree-like kernel. Gershman, Vul, and Tenenbaum (2012) used Gaussian process regression to describe how people perceive motion.

Within an exploration-exploitation context, Borji and Itti (2013) showed that Gaussian process-based optimization can explain how participants actively search for the best output when trying to optimize one-dimensional

functions. Schulz, Konstantinidis, and Speekenbrink (2016b) used Gaussian process exploration-exploitation algorithms to model behaviour in tasks that combine function learning and decision making (contextual multi-armed bandit tasks). Lastly, Schulz, Huys, Bach, Speekenbrink, and Krause (2016a) applied the safe optimization algorithm described here to scenarios in which participants had to cautiously optimize functions while never sampling below a given threshold.

9. Discussion

This tutorial has introduced Gaussian process regression as a general purpose inference engine to model, explore and exploit unknown functions. We have mainly focused on Gaussian process regression with a radial basis function kernel, but many other kernels and kernel combinations are possible and –as we have indicated above– many standard Bayesian regression approaches can be re-parametrized to be equivalent to Gaussian process regression, given specific assumptions about the kernel (Duvenaud et al., 2013).

Of course a tutorial like this can never be fully comprehensive. For example, many other acquisition functions than the ones introduced here (uncertainty sampling and UCB) exist. For pure exploration, another commonly used acquisition function attempts to minimize the expected variance over the whole input space (Gramacy and Apley, 2014). This method tends to sample less on the bounds of the input space, but can be hard to compute, especially if the input space is large. There also exist many different acquisition functions in the exploration-exploitation context, that are mostly discussed under the umbrella term Bayesian optimization (de Fre-

itas, Smola, and Zoghi, 2012). Two other common acquisition functions that are frequently applied here are the *probability of improvement* and the *expected improvement* (Močkus, 1975), which choose inputs that have a high probability to produce a better output than the input that is currently estimated to be best, or that produce an output which is expected to surpass the expected outcome of the input currently thought best.

Another situation in which Gaussian processes are frequently applied is called "global optimization", in which the goal is finding the overall maximum of a function as quickly as possible, but without worrying about the outputs that were produced in the search process. Parameter estimation is an example of such a problem and again different algorithms have been proposed, in particular the proposal by Hennig and Schuler (2012) to maximize the information gain about the location of the maximum. There is also a growing community of researchers who apply Gaussian process-based algorithms to return uncertainty estimates of traditional computational methods such as optimization, quadrature, or solving differential equations under the umbrella term "probabilistic numerics" (Hennig, Osborne, and Girolami, 2015).

Of course Gaussian process regression does not come without drawbacks. One such drawback, as compared to traditional regression models, is that parameter-based interpretations such as "if x increases by 1, y increases by 2" are not directly possible. However, as different kernels encode different assumptions about the underlying functions, assessing which kernel describes the underlying function best can be used as a basis to interpret the modeled function (Lloyd et al., 2014). Choosing the appropriate kernel is a difficult problem. General solutions to this are to construct more complicated kernels from a set of relatively simple base kernels (as shown above)

and to search the kernel space by proposing and checking new kernel combinations (Duvenaud et al., 2013), or to define the kernel in a non-parametric manner by using a non-parametric approach towards estimating the kernel itself (Wilson and Adams, 2013). Possibly the biggest drawback of Gaussian process regression is its poor scaling. As inferring the posterior involves inverting the matrix $[K(\mathbf{X}_t, \mathbf{X}_t) + \sigma_\epsilon^2 \mathbf{I}]$, inference scales cubically with the number of observations. Speeding up inference for Gaussian process regression therefore is a vivid topic of ongoing research. Some methods that have been proposed are to sparsely approximate inputs (Lawrence, Seeger, and Herbrich, 2003) or to bound the computational cost of the matrix inversion by projecting into a pre-defined finite basis of functions drawn from the eigen-spectrum of the kernel (Rahimi and Recht, 2007).

We hope to have shown some interesting examples of Gaussian process regression as a powerful tool for many applied situations, specifically exploration-exploitation scenarios, and hope that this tutorial will inspire more scientists to apply these methods in the near future. Currently available software that can assist in this is listed in Table 4.

Table 4: Gaussian process packages

Name	**Algorithm**	**Language**	**Author**
GPML	GP Toolbox	Matlab	Rasmussen and Nickisch (2010)
SFO	Submodular Optimization	Matlab	Krause (2010)
GPy	GP Toolbox	Python	Sheffield ML group (since 2012)
tgp	Tree GPs, GP regression	R	Gramacy et al. (2007)

References

Berkenkamp, F., Krause, A., Schoellig, A. P., 2016. Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics. arXiv preprint arXiv:1602.04450.

Borji, A., Itti, L., 2013. Bayesian optimization explains human active search. In: Advances in Neural Information Processing Systems. pp. 55–63.

Cavagnaro, D. R., Aranovich, G. J., McClure, S. M., Pitt, M. A., Myung, J. I., 2014. On the functional form of temporal discounting: An optimized adaptive test.

de Freitas, N., Smola, A., Zoghi, M., 2012. Regret Bounds for Deterministic Gaussian Process Bandits. arXiv preprint arXiv:1203.2177.

Desautels, T. A., Choe, J., Gad, P., Nandra, M. S., Roy, R. R., Zhong, H., Tai, Y.-C., Edgerton, V. R., Burdick, J. W., 2015. An active learning algorithm for control of epidural electrostimulation. IEEE Transactions on Biomedical Engineering 62 (10), 2443–2455.

Duvenaud, D., Lloyd, J. R., Grosse, R., Tenenbaum, J. B., Ghahramani, Z., 2013. Structure discovery in nonparametric regression through compositional kernel search. arXiv preprint arXiv:1302.4922.

Gershman, S. J., Blei, D. M., 2012. A tutorial on Bayesian nonparametric models. Journal of Mathematical Psychology 56 (1), 1–12.

Gershman, S. J., Malmaud, J., Tenenbaum, J. B., 2016. Structured representations of utility in combinatorial domains.

Gershman, S. J., Vul, E., Tenenbaum, J. B., 2012. Multistability and perceptual inference. Neural computation 24 (1), 1–24.

Gramacy, R. B., Apley, D. W., 2014. Local Gaussian process approximation for large computer experiments. Journal of Computational and Graphical Statistics (2), 1–28.

Gramacy, R. B., Lee, H. K., 2008. Bayesian treed Gaussian process models with an application to computer modeling. Journal of the American Statistical Association 103 (483).

Gramacy, R. B., et al., 2007. tgp: an R package for Bayesian nonstationary, semiparametric nonlinear regression and design by treed Gaussian process models.

Hennig, P., Osborne, M. A., Girolami, M., 2015. Probabilistic numerics and uncertainty in computations. Proceedings of the Royal Society 471 (2179), 20150142.

Hennig, P., Schuler, C. J., 2012. Entropy search for information-efficient global optimization. Journal of Machine Learning Research 13 (Jun), 1809–1837.

Jäkel, F., Schölkopf, B., Wichmann, F. A., 2007. A tutorial on kernel methods for categorization. Journal of Mathematical Psychology 51 (6), 343–358.

Kac, M., Siegert, A., 1947. An explicit representation of a stationary Gaussian process. The Annals of Mathematical Statistics, 438–442.

Krause, A., 2010. Sfo: A toolbox for submodular function optimization. Journal of Machine Learning Research 11 (Mar), 1141–1144.

Krause, A., Golovin, D., 2012. Submodular function maximization. Tractability: Practical Approaches to Hard Problems 3, 19.

Krause, A., Singh, A., Guestrin, C., 2008. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. The Journal of Machine Learning Research 9, 235–284.

Lawrence, N., Seeger, M., Herbrich, R., 2003. Fast sparse Gaussian process methods: The informative vector machine. In: Proceedings of the 16th Annual Conference on Neural Information Processing Systems. No. EPFL-CONF-161319. pp. 609–616.

Lloyd, J. R., Duvenaud, D., Grosse, R., Tenenbaum, J. B., Ghahramani, Z., 2014. Automatic construction and natural-language description of non-parametric regression models. arXiv preprint arXiv:1402.4304.

Lucas, C. G., Griffiths, T. L., Williams, J. J., Kalish, M. L., 2015. A rational model of function learning. Psychonomic Bulletin & Review, 1–23.

Meder, B., Nelson, J. D., 2012. Information search with situation-specific reward functions. Judgment and Decision Making 7 (2), 119–148.

Močkus, J., 1975. On Bayesian methods for seeking the extremum. In: Optimization Techniques IFIP Technical Conference. Springer, pp. 400–404.

Mockus, J., 2010. Bayesian Heuristic approach to discrete and global optimization: Algorithms, visualization, software, and applications. Springer-Verlag.

Myung, J. I., Cavagnaro, D. R., Pitt, M. A., 2013. A tutorial on adaptive design optimization. Journal of Mathematical Psychology 57 (3), 53–67.

Myung, J. I., Pitt, M. A., 2009. Optimal experimental design for model discrimination. Psychological Review 116 (3), 499.

Rahimi, A., Recht, B., 2007. Random features for large-scale kernel machines. In: Advances in Neural Information Processing Systems. pp. 1177–1184.

Rasmussen, C. E., Nickisch, H., 2010. Gaussian processes for machine learning (gpml) toolbox. Journal of Machine Learning Research 11 (Nov), 3011–3015.

Schulz, E., Huys, Q. J., Bach, D. R., Speekenbrink, M., Krause, A., 2016a. Better safe than sorry: Risky function exploitation through safe optimization. arXiv preprint arXiv:1602.01052.

Schulz, E., Konstantinidis, E., Speekenbrink, M., 2016b. Putting bandits into context: How function learning supports decision making. bioRxiv, 081091.

Schulz, E., Speekenbrink, M., Hernández-Lobato, J. M., Ghahramani, Z., Gershman, S. J., 2016c. Quantifying mismatch in bayesian optimization. In: NIPS Workshop on Bayesian Optimization: Black-box Optimization and beyond.

Schulz, E., Tenenbaum, J. B., Duvenaud, D., Speekenbrink, M., Gershman, S. J., 2016d. Probing the compositionality of intuitive functions. Tech. rep., Center for Brains, Minds and Machines (CBMM).

Schulz, E., Tenenbaum, J. B., Reshef, D. N., Speekenbrink, M., Gershman, S. J., 2015. Assessing the perceived predictability of functions. In:

Proceedings of the Thirty-Seventh Annual Conference of the Cognitive Science Society.

Sheffield ML group, since 2012. GPy: A Gaussian process framework in python. `http://github.com/SheffieldML/GPy`.

Srinivas, N., Krause, A., Kakade, S. M., Seeger, M., 2009. Gaussian process optimization in the bandit setting: No regret and experimental design. arXiv preprint arXiv:0912.3995.

Steyvers, M., Lee, M. D., Wagenmakers, E.-J., 2009. A Bayesian analysis of human decision-making on bandit problems. Journal of Mathematical Psychology 53 (3), 168–179.

Sui, Y., Gotovos, A., Burdick, J., Krause, A., 2015. Safe Exploration for Optimization with Gaussian Processes. In: Proceedings of the 32nd International Conference on Machine Learning (ICML-15). pp. 997–1005.

Wetzels, R., Vandekerckhove, J., Tuerlinckx, F., Wagenmakers, E.-J., 2010. Bayesian parameter estimation in the Expectancy Valence model of the Iowa gambling task. Journal of Mathematical Psychology 54 (1), 14–27.

Williams, C. K., 1998. Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In: Learning in graphical models. Springer, pp. 599–621.

Williams, C. K., Rasmussen, C. E., 2006. Gaussian processes for machine learning. the MIT Press 2 (3), 4.

Wilson, A. G., Adams, R. P., 2013. Gaussian process kernels for pattern discovery and extrapolation. In: ICML (3). pp. 1067–1075.

Wilson, A. G., Dann, C., Lucas, C., Xing, E. P., 2015. The human kernel. In: Advances in Neural Information Processing Systems. pp. 2854–2862.