# Chapter 2
# System Identification with GP Models

In this chapter, the framework for system identification with GP models is explained. After the description of the identification problem, the explanation follows the system identification framework that consists of roughly six stages:

1. defining the purpose of the model,
2. selection of the set of models,
3. design of the experiment,
4. realisation of the experiment and the data processing,
5. training of the model and
6. validation of the model.

The model identification is always an iterative process. Returning to some previous procedure step is possible at any step in the identification process, and this is usually necessary.

The listed stages are given in the sections describing the model's purpose (Sect. 2.1), the experiment design and data processing (Sect. 2.2), the model setup (Sect. 2.3), the model selection (Sect. 2.4), the model validation (Sect. 2.6), the dynamic model simulation (Sect. 2.7) and ends with an illustrative example of nonlinear system identification with a GP model (Sect. 2.8).

The *identification problem* [1, 2] is as follows: For a given set of past observations, i.e. delayed samples of input and output signals that form a regression vector, we would like to find a relationship with future output values. As already mentioned in the previous chapter, this relation can be presented as the concatenation of a mapping from the measured data to a regression vector, followed by a nonlinear mapping from the regression vector to the output space

$$y(k) = f(\mathbf{z}(k), \boldsymbol{\theta}) + \nu(k), \tag{2.1}$$



J. Kocijan, *Modelling and Control of Dynamic Systems Using Gaussian Process Models*, Advances in Industrial Control,
DOI 10.1007/978-3-319-21021-6_2

where

$$\mathbf{z}(k) = \boldsymbol{\varphi}(y(k-1), y(k-2), \ldots, u(k-1), u(k-2), \ldots), \tag{2.2}$$

$k$ is the sampling instant,
$f$ is a nonlinear mapping from the regression vector $\mathbf{z}$ to the output space,
$\boldsymbol{\theta}$ is the finite-dimensional parameter vector,
$\nu(k)$ represents the noise and accounts for the fact that the next output value $y(k)$ will not be an exact function of past data,
$\boldsymbol{\varphi}$ is a nonlinear mapping from the finite-dimensional vector of the measurements to the regression vector $\mathbf{z}$, and its components are referred to as regressors,
$y(k-i)$, $i = 1, 2, \ldots, k-1$ are the delayed samples of the measured output signal and
$u(k-i)$, $i = 1, 2, \ldots, k-1$ are the delayed samples of the measured input signal.

The temporal or time component is inevitably present when dealing with dynamic systems. Instead of considering time as an extra input variable to the model, the time is embedded into regressors in the form of delayed samples of input and output signals. In our notation the time, usually denoted with $t$, has been substituted for $k$, where $k$ represents the $k$th subsequent time-equidistant instant sampled with the sample period $T_s$.

The identification problem has thus to be decomposed into two tasks: (a) the selection of the regression vector $\mathbf{z}(k)$ and (b) the selection of the mapping $f$ from the space of the regressors to the output space.

When the mapping $f$ is presumed linear, we talk about the identification of linear dynamic systems. The more general case is when the mapping is nonlinear. While there are numerous methods for the identification of linear dynamic systems from measured data, the nonlinear systems identification requires more sophisticated approaches.

In general, the identification methods for nonlinear systems can be grouped into those for parametric and those for nonparametric system identification. While *parametric system identification* deals with the estimation of parameters for a known structure of the mathematical model, nonparametric system identification identifies the model of an unknown system without structural information.

*Nonparametric system identification* can be divided further [3]. The first group of methods is that where the system is approximated by a linear or nonlinear combination of some basis functions $\phi_i$ with $l$ coefficients $\mathbf{w} = [w_1, w_2, \ldots, w_l]^{\mathrm{T}}$ to be optimised

$$f(\mathbf{z}, \mathbf{w}) = F(\phi_i(\mathbf{z}), \mathbf{w}), \tag{2.3}$$

where $F$ is a function representing the nonlinear combination of basis functions. The most commonly seen choices in identification practice include artificial neural networks (ANNs), fuzzy models and Volterra-series models, which can be seen as universal approximators.

Let us briefly discuss the use of methods where the nonlinear system is approximated by the combination of basis functions. The problem of nonparametric system

identification is translated into the problem of a suitable basis function selection and of the coefficients' estimation, which can be considered further as a parameter estimation problem. The approach is sensitive to the choice of basis function. Depending on the nonlinearity, a fixed basis function approach could need a relatively large number of terms to approximate the unknown nonlinear system. The increase in the number of terms with the increase in the unknown system's complexity is called the '*curse of dimensionality*' [4]—the exponential growth of the modelled volume with the input space dimension [5]—leading to (a) a model with a large number of basis functions with corresponding parameters and (b) a lot of data needed for a system description. An example of such models is an ANN. The local model network (LMN) [4], a form of the fuzzy model, which we address in Sect. 3.3, reduces this problem, but has problems with a description of the off-equilibrium regions of the dynamic system [4, 6].

The second possibility is to estimate the unknown nonlinear system locally, point by point. Representatives of these methods are kernel methods like least-square support vector machines [7]. These methods circumvent the curse of dimensionality in the sense that they do not contain basis functions. These methods are considered local because any of them is actually a weighted average based on measurements in the neighbourhood of the point where the system is estimated [3].

Nevertheless, both possibilities, the one with basis functions and the one that is point-by-point, are based on the amount of measured data in the neighbourhood of the point where the system is identified. If the dimension of the problem is high, the amount of data necessary for training increases.

The curse of dimensionality is not an issue for linear and parametric nonlinear system identification. In such cases, it is not important whether the measured data used for the identification is distributed locally or far away from the point where the unknown system is identified.

For the nonparametric identification methods that estimate the unknown system locally, only local data is useful. For kernel methods, this depends on the kernel selection. Data points that are far away provide little value for these methods [3]. This means that local approaches would not perform well when modelling large regions with only a limited amount of data available.

As an alternative to methods for the identification of nonlinear dynamic systems that are strongly affected by the curse of dimensionality, the GP model was proposed in [6]. In this context, the unknown system to be identified at a given point and the data obtained at other points are assumed to be a joint GP with a mean and a covariance matrix that has some hyperparameters.

The idea of using GP models for system identification differs from both mentioned possibilities of nonlinear system identification that can be described as the local average approaches [3] because it is a probabilistic method. GP models provide a posteriori distribution. Using this distribution, a probabilistic estimate at a point of interest can be made based on the training data that can be close or far away from this point. This prediction is presumed to be Gaussian, characterised by a predictive mean and a predictive variance. As we have seen in the previous chapter, the predictive variance can be interpreted as a level of confidence in the predictive mean. This is

important, especially in the case when the predictive mean is not sufficiently close to the ground truth.

In other words, due to its probabilistic nature the GP model provides information about its estimate over the entire space defined by the regressors. The GP model is, therefore, not constrained to the space where the measured data is available.

Furthermore, the GP approach to modelling alleviates any model bias by not focusing on a single dynamics model, but by using a *probabilistic dynamics model*, a distribution over all plausible dynamics models that could have generated the observed experience. The probabilistic model is used to faithfully express and *represents the uncertainty* about the learned dynamics. We use a probabilistic model for the deterministic system. The probabilistic model does not necessarily imply that we assume a stochastic system. In the case of modelling, the deterministic system the probabilistic model is solely used to describe the uncertainty about the model itself. In the extreme case of a test input data $\mathbf{z}^*$ at the exact location $\mathbf{z}_i$ of a training input data, the prediction of the probabilistic model will be absolutely certain about the corresponding function value $p(f(\mathbf{z}^*)) = \delta(f(\mathbf{z}_i))$.

When using GP models for identification, it is important that we are also aware of the disadvantages [3] in this context.

The first one is the computational complexity due to the inverse of a high-dimensional covariance matrix during the training. The computational complexity measured with the number of computer operations rises with the third power of the number of identification points $N$ and is denoted with $\mathcal{O}(N^3)$, while the number of computer operations for the prediction mean is $\mathcal{O}(N)$ and for the prediction variance $\mathcal{O}(N^2)$. The issue of overcoming the computational complexity is addressed in Sect. 2.5.

The second is that the noise that corrupts the measurement used for the system identification does not always have a Gaussian distribution. The hyperparameters of the GP model are frequently optimised to maximise the marginal likelihood conditioned on the measured data where the assumption is that the marginal likelihood is Gaussian. In the case of non-Gaussian noise, this assumption is not correct and it is not known whether the maximum likelihood is achieved. However, this is exactly the same issue with any other known parametric or nonparametric method for system identification. In the case of non-Gaussian noise, the data can be transformed in the form that will be better modelled by the GPs. More details about transformations, called also GP warping (Sect. 2.3.3), can be found in [8, 9].

The third disadvantage is that the performance of GP models for system identification depends on the selection of training data and the covariance function with hyperparameters, which is system dependent. Again, this disadvantage can, in one form or another, be met with any method for system identification where the selection of the basis function or the system structure and the corresponding parameter estimation is system dependent.

## 2.1 The Model Purpose

The model purpose specifies the intended use of the model and has a major impact on the level of detail of the model. The decision for the use of a specific model derives, besides the model purpose, also from the limitations met during the identification process.

In general, dynamic system models can be used for [10]

- prediction,
- simulation,
- optimisation,
- analysis,
- control and
- fault detection.

*Prediction* means that on the basis of previous samples of the process input signal $u(k-i)$ and the process output signal $y(k-i)$ the model predicts one or several steps into the future. There are two possibilities: the model is built to directly predict $h$ steps into the future or the same model is used to predict a further step ahead by replacing the data at instant $k$ with the data at instant $k+1$ and using the prediction $\hat{y}(k)$ from the previous prediction step instead of the measured $y(k)$. This is then repeated indefinitely. The latter possibility is equivalent to *simulation*. Simulation therefore means that only on the basis of previous samples of the process input signal $u(k-i)$, and initial conditions for a certain number of samples of output signals, the model simulates future output values. Name prediction will, in our case, mostly mean a one-step-ahead prediction.

Both of these sorts of models can be used for the optimisation of systems, systems analysis, control and fault detection [10]. When a model is used for *optimisation*, the issues of optimisation duration and the disturbance of the process's normal operation, which frequently occur when an optimisation is carried out in the real world, are circumvented.

The procedure of identification can also be considered as a form of system *analysis*. Based on the model's form some process properties, like input–output behaviour, stationary properties, etc., can be inferred from the identified model.

*Control* design relies on a model of the process to be controlled. The form of the model depends on the control design method, but it has to be kept in mind that it is the closed-loop system performance containing the designed controller that is evaluated on the performance and not the process model itself. Nevertheless, there are control algorithms for which the closed-loop performance is strongly dependent on the model quality. Usually, the models used for control design have to be accurate in terms of some particular properties, e.g. range around the crossover frequency of the closed-loop system.

The idea of *fault detection* is to compare the process behaviour for the time being with its nominal behaviour, which is commonly represented with a process model.

Fault detection is a model-based strategy and the focus of the fault detection also determines the model properties.

Later in the text, some of these modelling applications will be highlighted in the context of GP applications.

## 2.2 Obtaining Data—Design of the Experiment, the Experiment Itself and Data Processing

The data describing an unknown system is very important in any black-box identification. The data can be obtained from an experiment and is processed afterwards. It is usually collected with measurements on the physical system of interest or, in special cases, from computer simulations. An example of such a special case is when identification is used for complexity reduction of a theoretically obtained model.

Unless the system to be identified does not allow controlled experimentation, the experiment needs to be carefully designed. In the case when experimentation is not possible, the safety of the system or its environment might be jeopardised by the experimentation and, consequently, the data needs to be collected from the daily operation.

The design of the experiment and the experiment itself are important parts of the identification procedure. The quality of the model depends on the system information contained in the measurement data, regardless of the identification method. The design of the experiment for the nonlinear system identification is described in more details in [10–13]. Only the main issues are highlighted in this section.

As already mentioned, the Gaussian process modelling approach relies on the relations among the input–output data and not on an approximation with basis functions. Consequently, this means that the distribution of the identification data within the process operating region is crucial for the quality of the model. Model predictions can only be highly confident if the input data to the model lies in the regions where the training data is available. The GP model provides good predictions when used for interpolation, but these are not necessarily good enough when used for extrapolation, which is indicated by the large variances of the model predictions.

Consequently, the data for the model training should be chosen reasonably, which can be obstructed by the nature of the process, e.g. limitations in the experimental design in industrial processes and the physical limitations of the system.

The experiment has to be pursued in such a way that the experiments provide sets of data that describe how the system behaves over its entire range of operation.

Here we list a number of issues that need to be addressed when the experiment is designed for the acquisition of data and subsequent data processing. The list is not exhaustive. Instead, it is given as a reminder of the important issues that engineers need to address, but the reader is, again, referred to exploit the details in [10–12].

Let us highlight the most important points.

**Nonlinearity test**. A nonlinearity test should be pursued to see whether a linear or nonlinear system is to be modelled. This is important to know, not only for GP models identification, but also for other methods as well. This can be done by a test of the superposition and homogeneity [12] or by checking the frequency response. The test of the superposition can be done with step changes of the input signal in different operating regions and checking the responses, while the frequency response test can be done by checking the output frequency response for sub-harmonic components when a harmonic signal excites the input.

**Sampling frequency**. For a good description of the process, the influential variables and a suitable sample time must be chosen. A rule of thumb is that the sampling frequency should be high enough to capture all the interesting dynamics of the system to be modelled. However, in the case that it is too high, problems with numerical ill-conditioning occur in the process of identification. The sampling frequency is therefore a compromise that is usually achieved iteratively.

**Selection of excitation signals**. The main issues that need to be addressed with the selection of the excitation signals are as follows.

*Purpose*. The selection of the excitation signals [10, 13] needs to be done based on the purpose of the modelling. For example, if the purpose of the model identification is control design, then good data information content is needed around the regions determined with the envisaged closed-loop set-point signals.

*Size of the training set*. The maximum amount of training data should be carefully selected due to the trade-off between the complexity of the computation and the information content.

*Range of input signals*. The input signals should be selected to excite the nonlinear dynamic system across the entire operating region. It is important to realise that it is the range of the input space that is important, i.e. the space determined with the input regressors, and not just the range of the input and output signals and their rates.

*Data distribution*. The input signals should be selected to populate the region with data homogenously and with sufficient density. A uniform data distribution over the region of interest is the ultimate goal.

*Prior knowledge*. The 'design of experiment' requires some prior knowledge about the process. An iterative procedure is necessary, in which the design of experiment and model identification are interlaced.

*Examples of signals*. Signals that can be used for the system identification are the amplitude-modulated PRBS (Pseudo-random Binary Sequence) signals. There are different ways to obtain these kinds of signals [10, 12], but it is important to homogenously cover the input space with samples of these signals.

If the system is unstable in any way or poorly damped, it might be necessary to conduct the experiment in a closed loop [12].

**Data preprocessing**. Some of the procedures that are important for the quality of the model before the data is used for the modelling are as follows.

*Filtering*. Aliasing, the effect that results when the signal reconstructed from the samples is different from the original continuous signal, which can be avoided with analogue pre-filtering and digital filtering, can be used to remove some low-frequency disturbances [12].

*Removing data redundancy*. A large amount of data in certain regions, most frequently the regions around the equilibria of a nonlinear system, dominates the data in other regions and may lead to the poor performance of the model in these regions. A possible solution to this problem is addressed in Sect. 2.5.2.

*Removing outliers*. Data outliers are not very problematic with GP models due to the smoothing nature of GP models [14].

*Scaling*. To cancel the influence of different measuring scales, the preprocessing of the measured data can be pursued, e.g. centering and scaling, here referred to as *normalisation*. Normalisation of the input and output signals helps with the convergence of the parameter optimisation that is part of the identification procedure.

## 2.3 Model Setup

In our case the selection of the GP model is presumed. This approach can be beneficial when the information about the system exists in the form of input–output data, when the data is corrupted by noise and measurement errors, when some information about the confidence in what we take as the model prediction is required and when there is a relatively small amount of data with respect to the selected number of regressors.

After the type of model is selected, the model has to be set up. In the case of the GP model, this means selecting the model regressors, the mean function and the covariance function.

### *2.3.1 Model Structure*

This and the following subsection deal with the choice of suitable regressors. In the first subsection different model structures are discussed, while in the second, methods for the selection of regressors are discussed. The selection of the covariance function is described in the third subsection.

It is mainly nonlinear models that are discussed in the context of GP modelling in general. There are various model structures for nonlinear, black-box systems that can also be used with GP models. An overview of the structures for nonlinear models is given in [1, 2]. The true noise properties that cause uncertainties in the identified model are usually unknown in the black-box identification and therefore different

model structures should be seen as reasonable candidate models and not as model structures reflecting the true noise descriptions.

The subsequently used nomenclature for nonlinear models is taken from [1]. The single-input single-output case is described in the continuation, but models can be easily extended to the multiple-input multiple-output case. The nonlinear, black-box models are divided into a number of different groups depending on the choice of regressors. Based on this choice we can divide the models into input–output models and state-space models. The prefix N for nonlinear is added to the names for different model structures. Input–output models that can be utilised for GP models are as follows:

**NFIR** (nonlinear finite impulse response) models, which use only the input values $u(k-i)$ as the regressors and are usually considered as deterministic input values in GP models. Since the regressors are only input values, the NFIR model is always stable, which is particularly important in the nonlinear case where the stability issue is a complex one. NFIR models are well suited for applications like [15] control, dynamic system identification, noise cancellation, nonstationary time-series modelling, adaptive equalisation of a communication channel and other signal processing applications. An example of the GP-NFIR model structure can be found in [16]. A block diagram of the GP-NFIR model is shown in Fig. 2.1.

**NARX** (nonlinear autoregressive model with exogenous input) models, which use the input values $u(k-i)$ and the measured output values $y(k-i)$ as the regressors and are usually considered as deterministic input values in GP models. The NARX model, also known as the equation-error or series-parallel model, is a prediction model.

$$\begin{aligned}\hat{y}(k) = f(&y(k-1), y(k-2), \ldots, y(k-n), u(k-1),\\ &u(k-2), \ldots, u(k-m)) + \nu,\end{aligned} \tag{2.4}$$

where $n$ is the maximum lag in the output values, $m$ is the maximum lag in the input values and $\nu$ is the white Gaussian noise. The NAR model is the special case of NARX model without the exogenous input and uses only the measured output values $y(k-i)$. The GP-NARX model was introduced in [17] and it is schematically shown in Fig. 2.2.

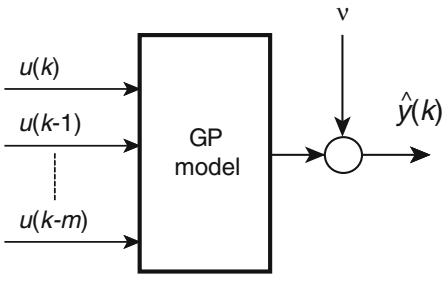


**Fig. 2.1** GP-NFIR model, where the output predictions are functions of previous measurements of input signals

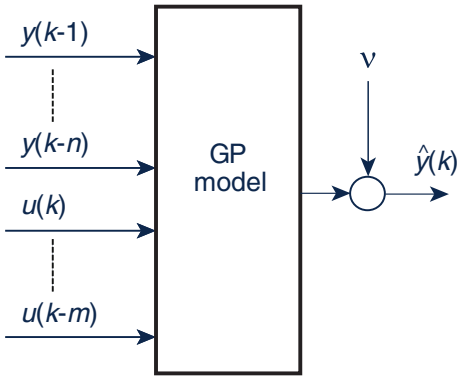


**Fig. 2.2** GP series-parallel or equation-error or NARX model, where the output predictions are functions of previous measurements of the input and output signals

**NOE** (nonlinear output error) models, which use the input values $u(k-i)$ and the output estimates $\hat{y}(k-i)$ as the regressors. The NOE model, also known as the parallel model, is a simulation model.

$$\hat{y}(k) = f(\hat{y}(k-1), \hat{y}(k-2), \ldots, \hat{y}(k-n), u(k-1), u(k-2), \ldots, u(k-m)) + \nu. \tag{2.5}$$

In the case of GP-NOE models $\hat{y}(k)$ is a normal distribution. When the normal distribution and its delayed versions are used as the regressors, the output of a nonlinear model is not a normal distribution anymore, and therefore output predictions are only approximations. The GP-NOE model is discussed in [18] and it is schematically shown in Fig. 2.3.

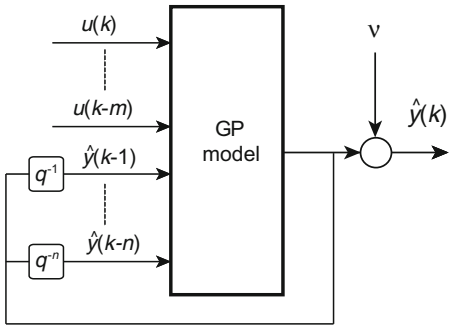


**Fig. 2.3** GP parallel or output-error or NOE model, where the output predictions are functions of previous measurements of the input signal only and delayed predictive distributions $\hat{y}$, or their approximations, are fed back to the input. $q^{-1}$ denotes the backshift operator. The time shift operator $q$ influences the instant in the following way: $q^{-1}y(k) = y(k-1)$

In general nonlinear system identification, there are other input–output model structures that have not yet been explored within the context of GP models like **NARMAX** (nonlinear autoregressive and moving average model with exogenous input) and **NBJ** (nonlinear Box–Jenkins) [1].

**State-space models**: State-space models are frequently used in dynamic systems modelling. Their main feature is the vector of internal variables called the states, which are regressors for these kinds of models. State-space regressors are less restricted in their internal structure. This implies that in general it might be possible to obtain a more efficient model with a smaller number of regressors using a state-space model [1].

The following model, described by a state-space equation, is considered:

$$\mathbf{x}(k+1) = f(\mathbf{x}(k), u(k)) + \boldsymbol{\nu}_1(k) \tag{2.6}$$

$$y(k) = g(\mathbf{x}(k), u(k)) + \nu_2(k) \tag{2.7}$$

where $\mathbf{x} \in \mathbb{R}^n$ is a vector of states, $y \in \mathbb{R}$ is a measurement output, $u \in \mathbb{R}$ is an input, $\boldsymbol{\nu}_1 \in \mathbb{R}^n$ and $\nu_2 \in \mathbb{R}$ are some white Gaussian noise sequences. The noise enters the system at two places. $\boldsymbol{\nu}_1$ is called the process noise and $\nu_2$ is called the measurement noise, $f$ is the transition or system function and $g$ is called the measurement function. In our case both functions can be modelled with GP models, so $f \sim \mathcal{GP}_f$ and $g \sim \mathcal{GP}_g$. Figure 2.4 shows a state-space model based on GP models.

The system identification task for the GP state-space model is concerned with $f$ in particular and can be described as finding the state-transition probability conditioned on the observed input and output values [19]

$$p(\mathbf{x}(k+1)|\mathbf{x}(k), \mathbf{u}(k), \mathbf{y}(k)); \mathbf{u}(k) = [u(k), \ldots, u(1)]^{\mathrm{T}}, \mathbf{y}(k) = [y(k), \ldots, y(1)]^{\mathrm{T}}. \tag{2.8}$$

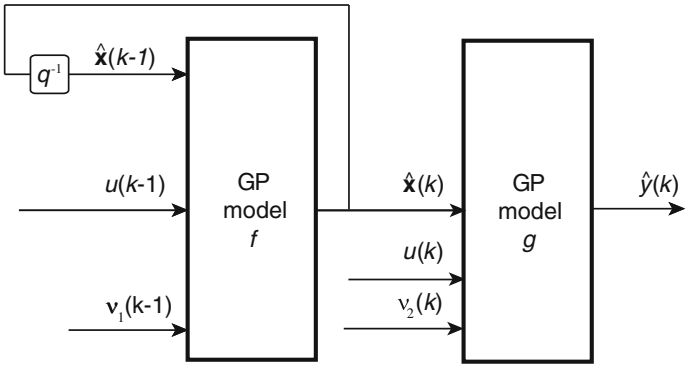


**Fig. 2.4** State-space model. $q^{-1}$ denotes the backshift operator

The system function $g$ can often be assumed to be known. Such an example is the situation where $g$ corresponds to a sensor model, where we know which of the states the sensor is measuring. This then simplifies the identification task. It is mentioned in [20] that using models that are too flexible for both $f$ and $g$ can result in problems of non-identifiability.

While using the state-space structure is common, the modelling with GP state-space models [21] is still a field of active research, and so the state-of-the-art is briefly reviewed here. At present, a state-space model of a dynamic system that involves GP models appears in the literature mainly in the context of an unknown state estimation from noisy measurements, i.e. filtering and smoothing. More results, also in the context of modelling for prediction, are to be expected in the future.

In the context of filtering and smoothing, the states $\hat{\mathbf{x}}(k)$ are estimated from the measurements $\mathbf{y}(k)$. More precisely, the posterior distribution of the states $p(\mathbf{x}(k)|\mathbf{y}(k))$ is to be found from measurements of $\mathbf{y}(k)$. If the used data history is up to and including $y(k)$, i.e. $\mathbf{y}(k) = [y(k), \ldots, y(1)]^{\mathrm{T}}$, this is called *filtering*. If we process the entire set of measurement data, then *smoothing* is taking place. Such an inference is usually made with Bayesian filters, like a Kalman filter [22] in the case of linear functions $f$ and $g$ or its nonlinear versions when at least one of the functions $f$ and $g$ is nonlinear.

GP models are used in the context of Bayesian filters, i.e. unscented and extended Kalman filters in [23] and in the context of an assumed density filter in [24]. A GP model with a state-space structure and without external input $u$ appears in [24, 25], where GP models of the functions $f$ and $g$ are obtained from presumed available observations of the states as the identification or training data. These two references describe how the inference of states can be made for dynamic systems after the functions $f$ and $g$ are identified.

We have to keep in mind that variables with a normal distribution at the input do not keep a normal distribution at the output of a nonlinear function. In the case of nonlinear functions, we deal with approximations of posterior distributions. Nevertheless, good results for the inference of posterior distributions $p(\mathbf{x}(k)|\mathbf{y}(k))$ are reported in [24, 25].

Other methods for the inference of the states $\mathbf{x}(k)$ using GP models have been reported, like particle filters, e.g. [26–29], or variational inference, e.g. [30].

Authors of [21] learn a state-space model without input $u$ with GPs by finding a posterior estimate of the latent variables and hyperparameters. The model in [21] is used for motion recognition.

The other situation is when the observations of states are not available for training. Filtering and smoothing in this case is described in [31, 32]. To learn the GP models for the functions $f$ and $g$ in such a case, they are parametrised by *pseudo training sets*, which are similar to the pseudo training sets used in so-called sparse GP approximations [33], which are described in Sect. 2.5.2. The system identification determines the appropriate hyperparameters for both GP models, such that the target time series $\mathbf{y}(k)$ can be explained. The expectation–maximisation algorithm used to determine the parameters is iterated between two steps. In the first a posterior distribution $p(\mathbf{Z}_{\mathbf{x}}|\mathbf{y}(k), \boldsymbol{\theta})$ on the hidden states $\mathbf{Z}_{\mathbf{x}} = [\mathbf{x}(k), \ldots, \mathbf{x}(1)]$ for a fixed parameter

setting $\boldsymbol{\theta}$ is determined, and in the second the parameters $\boldsymbol{\theta}^*$ of the GP state-space model that maximise the expected log likelihood $E(\ln p(\mathbf{Z}_\mathbf{x}|\mathbf{y}(k), \boldsymbol{\theta}))$ are searched, where the expectation is taken with respect to $p(\mathbf{Z}_\mathbf{x}|\mathbf{y}(k), \boldsymbol{\theta})$ from the first step. The log likelihood is decomposed into [32]

$$E(\ln p(\mathbf{Z}_\mathbf{x}|\mathbf{y}(k), \boldsymbol{\theta})) = E(\ln p(\mathbf{x}(1)|\boldsymbol{\theta}) + \sum_{k=2}^{N} \underbrace{\ln p(\mathbf{x}(k)|\mathbf{x}(k-1), \boldsymbol{\theta})}_{\text{Transition}} + \sum_{k=1}^{N} \underbrace{\ln p(y(k)|\mathbf{x}(k), \boldsymbol{\theta}))}_{\text{Measurement}} \tag{2.9}$$

In contrast to the listed methods that tend to model the GP using a finite set of identification data points and identifying functions $f$ and $g$, the method reported in [20] marginalises the transition function $f$ using sampling with the particle Markov chain Monte Carlo (MCMC) method. Consequently, the identification of $f$ is avoided for the states' estimation.

State-space models utilising GP models can also be found for continuous-time models. References [34–37] describe methods of the states' inference in the continuous-time domain. They deal mainly with linear transition functions to retain the normal distributions on states or combine them in hybrid models with first-principle models. The identification of a discrete-time linear GP state-space model is described in [38]. The signal processing aspect of GP modelling in the spatio-temporal domain is reviewed in [37].

The interested reader who wants more information about states estimation using GP models will find it in the references listed in this section.

### 2.3.2 Selection of Regressors

A very important step in the model setup is the selection of the model order. The problem of order determination in the case of input–output models is equivalent to the selection of the relevant input variables for the mapping function $f(\cdot)$ in Eq. (2.1). Therefore, the problem is actually the problem of selecting the regressors or input variables in statistics and system theory terminology, or features in the machine-learning terminology. As pointed out in [10], it is important to understand that in the case of the identification, the previous input values $u(k-i)$ and the previous output values $y(k-i)$ are considered as separate regressors. Nevertheless, subsequent input values, e.g. $y(k-1)$ and $y(k-2)$, are typically correlated. This correlation indicates redundancy, but these subsequent input values may both be relevant, so we cannot dismiss any of them. This complicates the order selection problem.

Most of the methods for system identification start with the assumption that the regressors or the input variables in general are known in advance. Nevertheless, it is crucial to obtain a good model that the appropriate regressors are selected. Here we review the selection of regressors from the viewpoint of selection-independent and relevant input variables to the mapping function.

From the identification point of view, the regressors should be independent of each other and must carry as much information as possible for the output value prediction. If the input data is not independent, the information about the output data is doubled, which results in larger model dimensions and a more difficult search for the optimal model. The same is true if the regressors do not carry information about the output data and so represent redundant regressors.

Our goal is to select only as many regressors as are really necessary. Every additional regressor increases the complexity of the model and makes the model optimisation more demanding. Nevertheless, optimal input selection is often an intractable task, and an efficient input selection algorithm is always an important element in many modelling applications.

A quick look at the literature reveals plenty of methods and algorithms for regressor selection. A thorough overview of these methods and algorithms would take up too much space, so only a general overview is presented and a survey of the literature is listed.

Various authors divide the methods differently. We adopt the division of the regressors' selection into three major groups [39–42]: wrappers or *wrapper methods*, *embedded methods* and *filter methods*.

**Wrapper methods** are the so-called brute-force methods for regressor selection. The basic idea behind these methods is that they form a kind of wrapper around the system model, which is considered as a black box. No prior knowledge or internal state of the model is considered. The search for the optimal vector of regressors is initiated from some basic set of regressors. After the model optimisation and cross-validation, the regressors are added to or taken from the model. Successful models, according to selected performance criteria, are kept, while poorly performing models are rejected.

The wrapper methods are very general and easy to implement. Nevertheless, in the case of a large number of regressor candidates the methods require lots of computational effort, so various search or optimisation methods are used. Some of these methods or groups of methods are [42] forward selection, backward elimination, nested subset, exhaustive global search, heuristic global search, single-variable ranking and other ranking methods. The wrapper methods are also known by the names validation-based regressor selection or exhaustive search for best regressors [43].

**Embedded methods** have the regressor selection built into the model optimisation procedure. For example, if a certain sort of model has a property that the values of model's parameters correspond to the importance of the used regressors, then properly selected regressors with lower importance can be eliminated. This property is called automatic relevance determination—ARD [44]. GP models possess this property for certain covariance functions, e.g. squared exponential covariance with hyperparameters describing the scales for each regressor. The ARD property assumes that the global minimum of the parameter optimisation cost function is achieved. Some other embedded methods are coupled with model optimisation, e.g. the direct optimisation method, or are weight-based, e.g. stepwise regression, recursive feature elimination.

**Filter methods** do not rely on the model structure we identify like the other two groups of methods. The measure of relevance for the regressors or combinations of regressors is extracted directly from the identification data. The relevant regressors are selected based on this measure. The relevance measure is usually computed based on the statistical properties of identification data, e.g. correlations for linear systems, conditional probabilities, or based on measures from information theory, e.g. information entropy, mutual information, or based on other properties, e.g. mapping function smoothness. These methods are attractive because they are computationally efficient in comparison with the wrapper and embedded methods. Computational efficiency comes from the fact that multiple optimisation runs are not necessary and from the relatively straightforward computation of the filter method measures.

In the case when the signals from which regressors are to be selected are known, the problem is downsized to the problem of lag or model-order selection. If we eliminate the dead time from the responses, it is often assumed that regressors that have a smaller lag are more relevant for the prediction. This can be a substantial aid for the regressor selection algorithm. The assumption about the smoothness of the regressors of the mapping function is also frequently met in filter methods. Filter methods are the method of He and Asada [12, 45], and the false nearest neighbours method [46].

When it comes to regressor selection, it is also important to touch on embedding theory [47], which describes the theory of mapping continuous-time dynamic systems to discrete-time dynamic models. In the case that prior knowledge about the order of the continuous-time dynamic model is available, or presumed, the non-minimal realisation of the discrete-time dynamic model might be required to capture the dynamic of the nonlinear system. The non-minimal realisation means the order of the discrete-time model is higher than the order known by prior. This is in accordance with Taken's embedding theorem [48], which determines the necessary order of the model obtained from sampled input–output data.

### *2.3.3 Covariance Functions*

The choice of kernel function, which in the context of GP modelling is called the covariance function, is of fundamental importance for successful modelling with kernels. The covariance function reflects the correlations between different training data observations. The parameters of the covariance functions must be determined in order to obtain accurate model predictions. More information on the topic of covariance functions' selection and its use in GP models can be found in [49, 50]. Here we provide an overview of this topic.

The prior over mapping functions that is the Gaussian process is specified by its mean and covariance matrix and the covariance function is used to generate this covariance matrix. The covariance matrix reflects the relationship between the data and the prior knowledge or the assumptions of the system to be modelled. The elements of the covariance matrix are obtained with covariance functions, i.e. kernel

functions. The hyperparameters of the covariance functions must be sought to match the behaviour of the model with the original system. However, before the hyperparameters are determined, a suitable covariance function must be selected. When selecting the covariance function, we need to be aware of the inherent assumptions relating to the regression problem that we discuss [50]. The first one is that the collected training data must represent the characteristics of the function to be modelled. The second one is that measurements of a real system contain noise, and therefore a noise model needs to be incorporated into the model. The third assumption is that two data points close together in the input space are likely to have a greater correlation than two points that are distant. We assume that similar input values are likely to result in similar target values and, consequently, the training points near to a test point should be informative about the desired prediction [50]. It is the covariance function that defines the nearness of the individual data points.

We are not so much interested in the covariance between the input and output values or the covariance between pairs of different input values. We are interested in the covariance of pairs of the output values, which is presumed through the relationship between pairs of the input values, as described with Eq. (1.10). Two data points that are close together in the input space are to be informative about each other's respective targets and expose a high covariance between these targets. Consequently, two data points that are distant have a low covariance between two corresponding targets.

The covariance function and its hyperparameters can also be selected to reflect the prior knowledge about the lengthscale property. This means that we can select the covariance function, where the similarity between nearby input data decays more quickly or slowly with their distance.

The covariance function must generate a positive, semi-definite, covariance matrix [49], which limits the choice of functions to be covariance functions. A positive-definite covariance function will ensure a positive-definite covariance matrix, which guarantees the existence of a valid Gaussian process.

A number of valid covariance functions have been defined in the literature, see [49] for an overview. Most commonly, they are divided into stationary and nonstationary covariance functions. The stationary covariance functions are those that are functions of the distance between the input data and therefore invariant to translations in the input space.

Stationary covariance functions are more commonly used for implementation and interpretability reasons. Nevertheless, there are some cases, e.g. the system changes its behaviour during operation, when a nonstationary covariance function might be a better choice.

Another property that is of importance only for stationary covariance functions is a smoothness property of the Gaussian process prior determined by the covariance function. The selection of the covariance function influences the mean-square continuity [49] and differentiability [49] of the functions describing the system to be modelled.

For systems modelling, the covariance function is usually composed of two main parts:

$$C(\mathbf{z}_i, \mathbf{z}_j) = C_f(\mathbf{z}_i, \mathbf{z}_j) + C_n(\mathbf{z}_i, \mathbf{z}_j), \tag{2.10}$$

where $C_f$ represents the functional part and describes the unknown system we are modelling and $C_n$ represents the noise part and describes the model of the noise.

It is often assumed that the noise $\nu$ has an independent, identically distributed Gaussian distribution with zero mean and variance $\sigma_n^2$; $\nu \sim \mathcal{N}(0, \sigma_n^2)$. This kind of noise is called white Gaussian noise. This means that there is no cross-covariance between the noise and the system input data and it affects only the diagonal elements of the covariance matrix

$$C_n(\mathbf{z}_i, \mathbf{z}_j) = \sigma_n^2 \delta_{ij} \tag{2.11}$$

and $\delta_{ij} = 1$ if $i = j$ and 0 otherwise, which essentially encodes that the measurement noise is independent.

The functions $C_f(\mathbf{z}_i, \mathbf{z}_j)$ and $C_n(\mathbf{z}_i, \mathbf{z}_j)$ can be selected separately, because the sum of two nonnegative definite functions gives a nonnegative definite function. An alternative for noise is that it is encoded in the likelihood [51], as we will see later in the text (Eq. (2.26)).

When the noise is not modelled as white noise, it can be modelled differently, e.g. as an input-dependent noise model like ARMA noise [52, 53].

Let us list the most common stationary and nonstationary covariance functions that can be used with GP modelling.

**Stationary Covariance Functions**

*Constant covariance function*

The simplest covariance function is the one that has the same value over the whole domain. This is the constant covariance function given by

$$C_f(\mathbf{z}_i, \mathbf{z}_j) \;=\; \sigma_f^2. \tag{2.12}$$

The only hyperparameter $\sigma_f^2$ represents the scaling factor of the possible variations of the function. The function is illustrated in Fig. 2.5. Due to the simplicity of the constant covariance function, it is normally used in combination with other covariance functions.

*Squared exponential covariance function*

This is one of the most commonly used covariance functions in Gaussian process modelling when the function to be modelled is assumed to exhibit smooth and continuous behaviour with a high correlation between the output data and the input data in close proximity.

A squared exponential covariance function is also called a Gaussian covariance function. It is defined by

$$C_f(\mathbf{z}_i, \mathbf{z}_j) = \sigma_f^2 \exp\left(-\frac{r^2}{2l^2}\right). \tag{2.13}$$

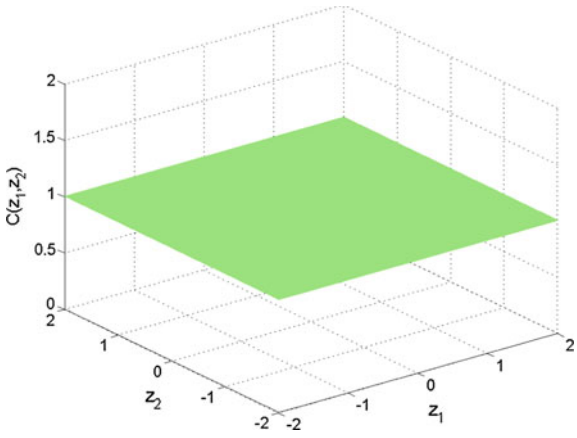


**Fig. 2.5** Constant covariance function of two one-dimensional variables with the hyperparameter $\sigma_f = 1$

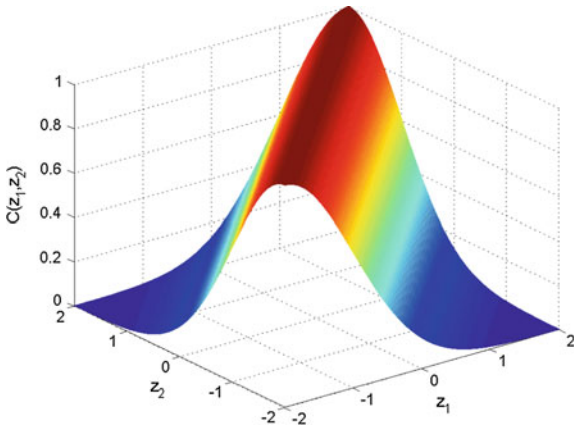


**Fig. 2.6** Isotropic squared exponential covariance function of two one-dimensional variables with the hyperparameters $\sigma_f = 1$ and $l = 1$

The hyperparameter $\sigma_f^2$ represents the scaling factor of the possible variations of the function or the vertical scaling factor and the hyperparameter $l$ is called the horizontal scaling factor and determines the relative weight on distance for the input variable $\mathbf{z}$. The variable $r$ is the input distance measure and is $r = |\mathbf{z}_i - \mathbf{z}_j|$. The function is illustrated in Fig. 2.6.

Samples of functions with different hyperparameters are given in Fig. 2.7.

The covariance function, as represented with Eq. (2.13), decays monotonically with $r$. These kinds of functions are called isotropic covariance functions [49].

Anisotropic versions of covariance functions can be created by setting $r^2(\mathbf{z}_i, \mathbf{z}_j) = (\mathbf{z}_i - \mathbf{z}_j)^{\mathrm{T}} \boldsymbol{\Lambda}^{-1} (\mathbf{z}_i - \mathbf{z}_j)$ for some positive, semi-definite matrix $\boldsymbol{\Lambda}^{-1}$:

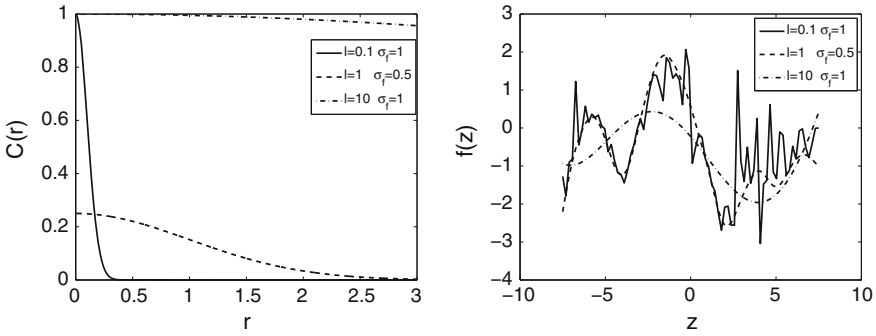


**Fig. 2.7** Squared exponential covariance function with different hyperparameters (*left figure*) and random sample functions from the GP model with squared exponential covariance function (*right figure*)

$$C_f(\mathbf{z}_i, \mathbf{z}_j) = \sigma_f^2 \exp\left[-\frac{1}{2}(\mathbf{z}_i - \mathbf{z}_j)^{\mathrm{T}} \boldsymbol{\Lambda}^{-1} (\mathbf{z}_i - \mathbf{z}_j)\right]$$
$$= \sigma_f^2 \exp\left[-\frac{1}{2}\sum_{d=1}^{D} w_d (z_{di} - z_{dj})^2\right], \tag{2.14}$$

where $w_d = \frac{1}{l_d^2}; d = 1, \ldots, D$.

If $\boldsymbol{\Lambda}^{-1}$ is diagonal, $\boldsymbol{\Lambda}^{-1} = \mathrm{diag}([l_1^{-2}, \ldots, l_D^{-2}])$ this implements the use of different length scales on different regressors and can be used to assess the relative importance of the contributions made by each regressor through comparison of their lengthscale hyperparameters. This is a property called *Automatic Relevance Determination* (ARD).

The ARD property was first introduced in [54, 55] in the context of a Bayesian neural network implementation. The ARD property can be used to optimise the structure, i.e. the regressor selection, of the GP model.

See [49] for a discussion on the use of the non-diagonal matrix $\boldsymbol{\Lambda}^{-1}$.

It must be kept in mind that in the case of dynamic systems identification the input dimension is high and this makes setting the assumptions on the mapping function to be modelled somehow difficult. A squared exponential covariance function is therefore frequently used because smooth and continuous input–output characteristics are expected, commonly from lots of dynamic systems, even though such assumptions are sometimes unrealistic, as argued in [56].

It is shown in [24, 51] that a GP model with a squared exponential covariance function corresponds to a universal function approximator.

*Exponential covariance function*

An exponential covariance function is used when the function to be modelled is assumed to be continuous, but not smooth and non-differentiable in the mean-square sense [49].

It is defined by

$$C_f(\mathbf{z}_i, \mathbf{z}_j) = \sigma_f^2 \exp\left(-(\frac{r}{l})^d\right) \quad \text{for} \quad 0 < d \leq 2. \tag{2.15}$$

The hyperparameter $\sigma_f^2$ represents the scaling factor of the possible variations of the function or the vertical scaling factor, the hyperparameter $l$ or the horizontal scaling factor determines the relative weight on distance for the input variable $\mathbf{z}$ and the hyperparameter $d$ determines the exponent. The variable $r$ is the input distance measure and is $r = |\mathbf{z}_i - \mathbf{z}_j|$. The function is illustrated in Fig. 2.8. Samples of functions with different hyperparameters are given in Fig. 2.9.

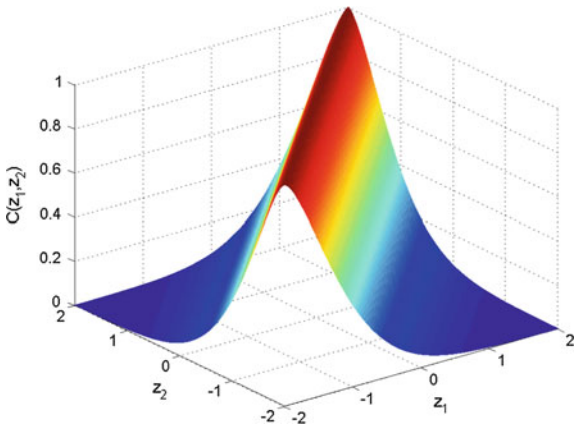


**Fig. 2.8** Exponential covariance function of two one-dimensional variables with the hyperparameters $\sigma_f^2 = 1$, $l = 1$ and $d = 1$

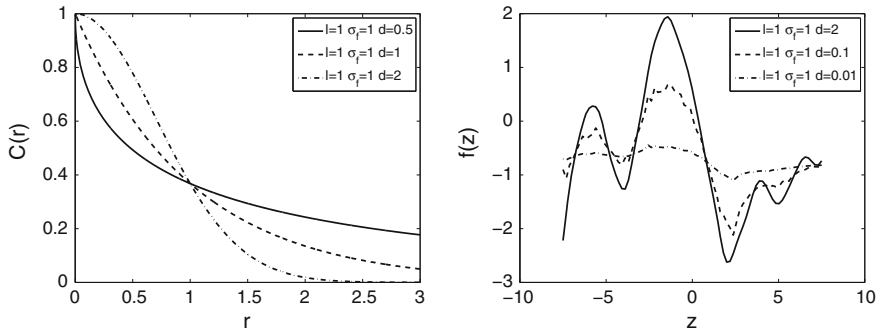


**Fig. 2.9** Exponential covariance function with different hyperparameters (*left figure*) and random sample functions from the GP model with an exponential covariance function (*right figure*)

*Rational quadratic covariance function*

The rational quadratic covariance function is used when the function to be modelled is assumed to be continuous and differentiable in the mean-square sense [49]. Therefore, it is not the appropriate choice when the function to be modelled contains a discontinuity or is discontinuous in its first few derivatives. The rational quadratic covariance function can be seen as a scale mixture or an infinite sum of squared exponential covariance functions with different characteristic length scales.

It is defined by

$$C_f(\mathbf{z}_i, \mathbf{z}_j) = \sigma_f^2 \left(1 + \frac{r^2}{2\alpha l^2}\right)^{-\alpha}. \tag{2.16}$$

The hyperparameter $\sigma_f^2$ represents the scaling factor of the possible variations of the function or the vertical scaling factor, the hyperparameter $l$ or the horizontal scaling factor determines the relative weight on distance for the input variable $\mathbf{z}$ and $\alpha$ is a positive hyperparameter. The variable $r$ is the input distance measure and is $r = |\mathbf{z}_i - \mathbf{z}_j|$. The function is illustrated in Fig. 2.10. Samples of functions with different hyperparameters are given in Fig. 2.11.

The ARD version of the rational quadratic covariance function is

$$C_f(\mathbf{z}_i, \mathbf{z}_j) = \sigma_f^2 \left(1 + \frac{1}{2}(\mathbf{z}_i - \mathbf{z}_j)^{\mathrm{T}} \boldsymbol{\Lambda}^{-1} (\mathbf{z}_i - \mathbf{z}_j)\right)^{-\alpha}. \tag{2.17}$$

*Matérn covariance functions*

The Matérn covariance function is used when assumptions about the function to be modelled are less stringent regarding the smoothness or differentiability in the mean-square sense.

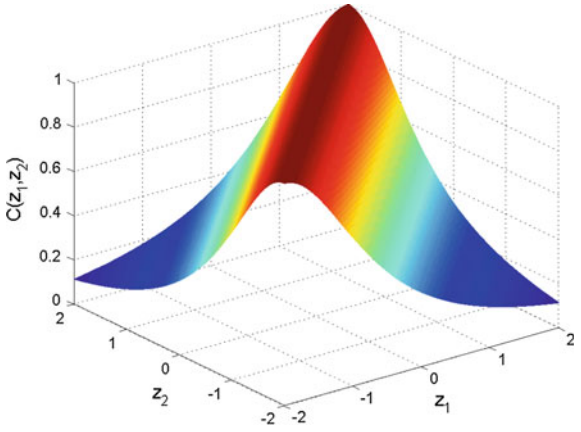


**Fig. 2.10** Isotropic rational quadratic covariance function of two one-dimensional variables with the hyperparameters $\sigma_f = 1, l = 1$ and $\alpha = 1$

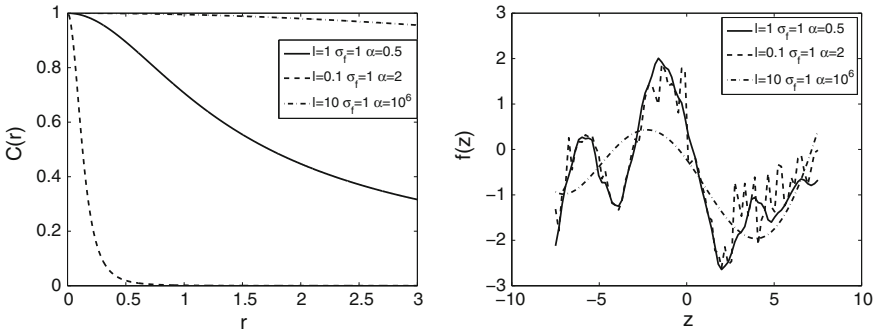


**Fig. 2.11** Rational quadratic covariance function with different hyperparameters (*left figure*) and random sample functions from the GP model with rational quadratic covariance function (*right figure*)

It is defined by

$$C_f(\mathbf{z}_i, \mathbf{z}_j) = \sigma_f^2 \left(\frac{2^{1-d}}{\Gamma(d)}\right)\left(\frac{\sqrt{2d}r}{l}\right)^d K_d\left(\frac{\sqrt{2d}r}{l}\right). \tag{2.18}$$

The hyperparameter $\sigma_f^2$ represents the scaling factor of the possible variations of the function or the vertical scaling factor, the hyperparameter $l$ or the horizontal scaling factor determines the relative weight on distance for the input variable $\mathbf{z}$, $K_d$ is a modified Bessel function and the hyperparameter $d$ can be seen to control the differentiability of the modelled mapping function. The variable $r$ is the input distance measure and is $r = |\mathbf{z}_i - \mathbf{z}_j|$. Often, $d$ is fixed to be $d = \frac{3}{2}$ or $d = \frac{5}{2}$. Increasing the value of $d$ makes the sample function smoother. In [49] it is stated that in the cases where $d > \frac{5}{2}$ it is probably difficult to distinguish between the properties of the sample functions in the case of noisy training data.

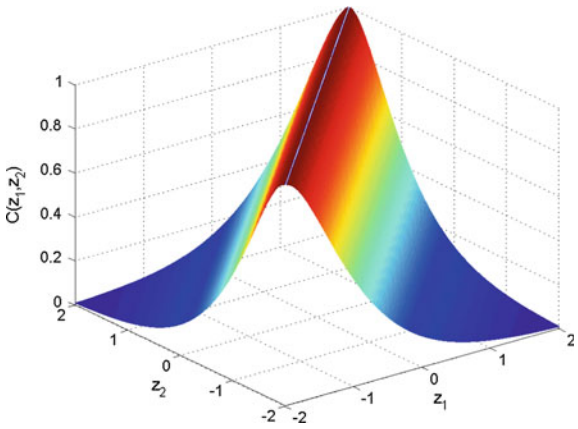


**Fig. 2.12** Matérn covariance function of two one-dimensional variables with the hyperparameters $\sigma_f = 1$, $l = 1$ and $d = 3/2$

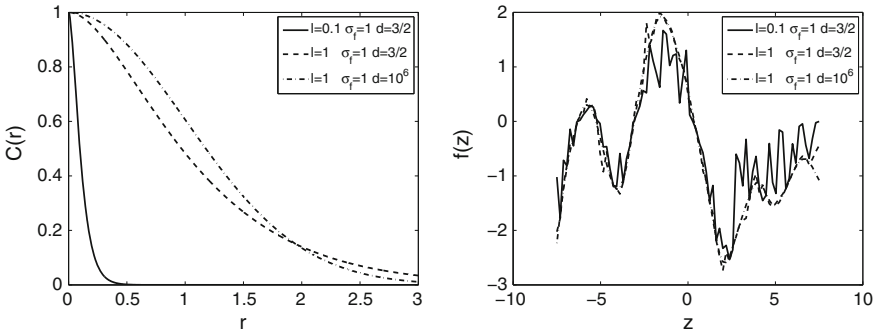


**Fig. 2.13** Matérn covariance function with different hyperparameters (*left figure*) and random sample functions from the GP model with the Matérn covariance function (*right figure*)

The function is illustrated in Fig. 2.12. Samples of functions with different hyperparameters are given in Fig. 2.13.

*Periodic covariance functions*

A periodic covariance function does not have a large value only between two data points that are close together, but also between data points that are on a fixed distance, i.e. period. There exist many periodic covariance functions. A representative one is defined by

$$C_f(\mathbf{z}_i, \mathbf{z}_j) = \sigma_f^2 \exp\left(-\frac{2}{l^2}\sin^2\left(\frac{\pi}{T_p}r\right)\right) \tag{2.19}$$

It can be used for the modelling of functions that repeat themselves exactly. The hyperparameter $\sigma_f^2$ represents the scaling factor of the possible variations of the function or the vertical scaling factor, the hyperparameter $l$ or the horizontal scaling factor determines the relative weight on distance for the input variable $\mathbf{z}$ and the hyperparameter $T_p$ defines the period. The variable $r$ is the input distance measure and is $r = |\mathbf{z}_i - \mathbf{z}_j|$.

The function is illustrated in Fig. 2.14. Samples of functions with different hyperparameters are given in Fig. 2.15.

**Nonstationary Covariance Function**

*Linear Covariance Function*

A linear covariance function is used when the function to be modelled is assumed to be linear.

It is defined by

$$C_f(\mathbf{z}_i, \mathbf{z}_j) = \sigma_f^2(\mathbf{z}_i \cdot \mathbf{z}_j + 1), \tag{2.20}$$

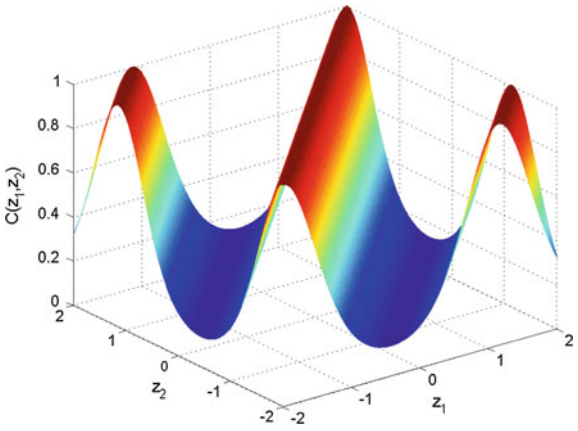


**Fig. 2.14** Periodic covariance function of two one-dimensional variables with the hyperparameters $\sigma_f = 1$, $l = 1$ and $T_p = \pi$

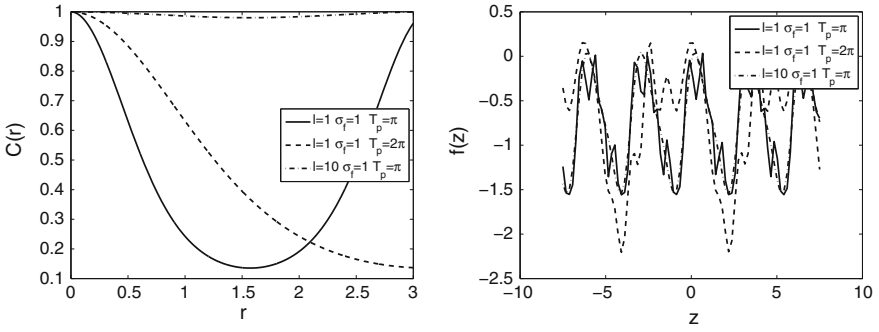


**Fig. 2.15** Periodic covariance function with different hyperparameters (*left figure*) and random sample functions from the GP model with a periodic covariance function (*right figure*)

or without a bias term by

$$C_f(\mathbf{z}_i, \mathbf{z}_j) = \sigma_f^2(\mathbf{z}_i \cdot \mathbf{z}_j), \tag{2.21}$$

The hyperparameter $\sigma_f^2$ represents the scaling factor of the possible variations of the function or the vertical scaling factor. The function in Eq. (2.20) is illustrated in Fig. 2.16 together with samples of functions with different hyperparameters. The linear covariance function without a bias term can be generalised to

$$C_f(\mathbf{z}_i, \mathbf{z}_j) = \mathbf{z}_i^{\mathrm{T}} \boldsymbol{\Lambda}^{-1} \mathbf{z}_j. \tag{2.22}$$

where $\boldsymbol{\Lambda}^{-1}$ is a general positive semi-definite matrix used on the components of $\mathbf{z}$. If $\boldsymbol{\Lambda}^{-1}$ is diagonal, we obtain a linear covariance function with the ARD property.

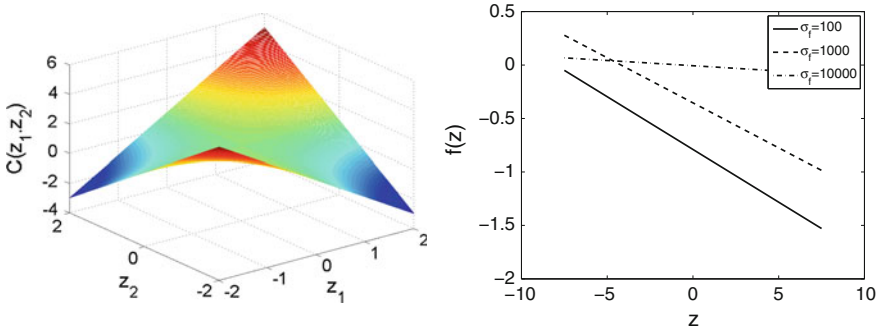


**Fig. 2.16** Linear covariance function of two one-dimensional variables with the hyperparameters $\sigma_f^2 = 1$ (*left figure*) and random sample functions from the Gaussian process with a linear covariance function (*right figure*)

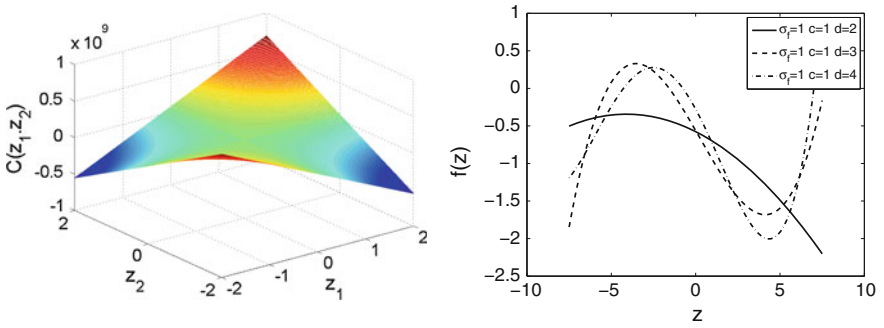


**Fig. 2.17** Polynomial covariance function of two one-dimensional variables with the hyperparameters $\sigma_f = 1$, $c = 1$ and $d = 10$ (*left figure*) and random sample functions from the GP model with a polynomial covariance function (*right figure*)

*Polynomial covariance function*

The polynomial covariance function is defined by

$$C_f(\mathbf{z}_i, \mathbf{z}_j) = (\sigma_f^2 \mathbf{z}_i \cdot \mathbf{z}_j + c)^d. \tag{2.23}$$

The hyperparameter $\sigma_f^2$ represents the scaling factor of the possible variations of the function or the vertical scaling factor, the hyperparameter $c$ determines the vertical bias and the hyperparameter $d$ determines the exponent.

The polynomial covariance function is not thought to be very useful for regression problems, as the prior variance will become very large with $|\mathbf{z}|$ as $|\mathbf{z}| > 1$.

The function is illustrated in Fig. 2.17 together with samples of functions with different hyperparameters.

*Neural network covariance function*

The neural network covariance function [49] is defined by

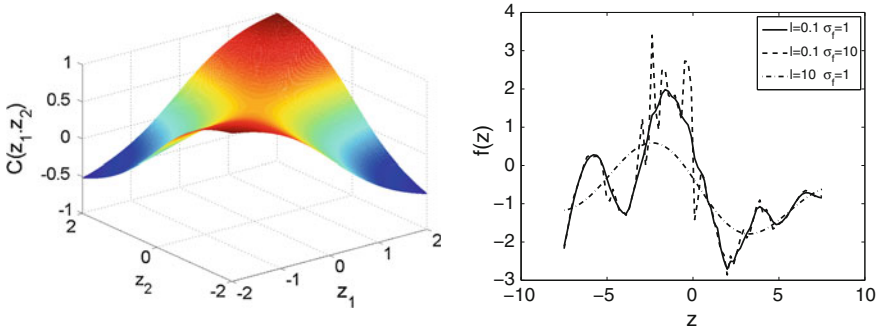


**Fig. 2.18** Neural network covariance function of two one-dimensional variables with the hyperparameter $\sigma_f = 1$ (*left figure*) and random sample functions from the GP model with a neural network covariance function

$$C_f(\mathbf{z}_i, \mathbf{z}_j) = \sigma_f^2 \frac{2}{\pi} \sin^{-1}\left( \frac{2\tilde{\mathbf{z}}_i^{\mathrm{T}} \boldsymbol{\Lambda}^{-1} \tilde{\mathbf{z}}_j}{\sqrt{1 + 2\tilde{\mathbf{z}}_i^{\mathrm{T}} \boldsymbol{\Lambda}^{-1} \tilde{\mathbf{z}}_i} \sqrt{1 + 2\tilde{\mathbf{z}}_j^{\mathrm{T}} \boldsymbol{\Lambda}^{-1} \tilde{\mathbf{z}}_j}} \right), \tag{2.24}$$

where $\tilde{\mathbf{z}}_i = [1, \mathbf{z}_i] = [1, z_1, \ldots, z_D]^{\mathrm{T}}$, $\boldsymbol{\Lambda}^{-1}$ is the covariance matrix on components of $\mathbf{z}$ and is often set as a unity matrix multiplied by $l^{-2}$.

For neural networks, a more common sigmoid function, e.g. $\tanh(\cdot)$, is not positive definite so it cannot be used as a covariance function [49].

The covariance function in Eq. (2.24) has been successfully applied to modelling the input–output mapping function of the step form in [49].

This function is illustrated in Fig. 2.18 together with samples of functions with different values of the hyperparameters.

The covariance functions can be manipulated in different ways to form a new composite covariance function:

**Sum of covariance functions**

A sum of two kernels is also a kernel. Therefore, a sum of two covariance functions $C_1(\mathbf{z}_i, \mathbf{z}_j)$ and $C_2(\mathbf{z}_i, \mathbf{z}_j)$ is a nonnegative definite function $C(\mathbf{z}_i, \mathbf{z}_j) = C_1(\mathbf{z}_i, \mathbf{z}_j) + C_2(\mathbf{z}_i, \mathbf{z}_j)$ and consequently a covariance function. This property is, for example, useful where a number of different characteristic length scales can be observed. If you sum together two kernels, then the resulting kernel will have a high value if either of the two summed kernels has a high value. An example is the summation of a linear and a periodic covariance function for obtaining a kernel that can be used for modelling functions that are periodic with an increasing mean, as we move away from the origin.

**Product of covariance functions**

Similarly, a product of two kernels is also a kernel. Therefore, a product of two covariance functions $C_1(\mathbf{z}_i, \mathbf{z}_j)$ and $C_2(\mathbf{z}_i, \mathbf{z}_j)$ is a nonnegative definite function $C(\mathbf{z}_i, \mathbf{z}_j) = C_1(\mathbf{z}_i, \mathbf{z}_j) \cdot C_2(\mathbf{z}_i, \mathbf{z}_j)$ and consequently a covariance function. If you

multiply together two kernels, then the resulting kernel will have a high value only if both of the two used kernels have a high value. An example is the multiplication of a periodic and a squared exponential covariance function for obtaining a kernel that can be used for modelling functions that do not repeat themselves exactly.

**Vertical rescaling**

This is the operation where a stationary covariance function is transformed into a nonstationary one. Let it be $g(\mathbf{z}_i) = a(\mathbf{z}_i)f(\mathbf{z}_i)$, where $a(\mathbf{z}_i)$ is a deterministic function, $f(\mathbf{z}_i)$ is a random process and $C_1(\mathbf{z}_i, \mathbf{z}_j) = \text{cov}(f(\mathbf{z}_i), f(\mathbf{z}_j))$. The new covariance function is $C(\mathbf{z}_i, \mathbf{z}_j) = \text{cov}(g(\mathbf{z}_i), g(\mathbf{z}_j)) = a(\mathbf{z}_i)C_1(\mathbf{z}_i, \mathbf{z}_j)a(\mathbf{z}_j)$. This method can be used to normalise kernels.

**Convolution of covariance functions**

A new covariance function can also be obtained with convolution. This operation also converts a stationary covariance function into a nonstationary one. If $C_2(\mathbf{z}_i, \mathbf{y}_i)$ is an arbitrary fixed covariance function, $C_1(\mathbf{y}_i, \mathbf{y}_j) = \text{cov}(f(\mathbf{y}_i), f(\mathbf{y}_j))$ and the transformation is $g(z) = \int C_2(\mathbf{z}_i, \mathbf{y}_i)f(\mathbf{y}_i)d\mathbf{y}_i$ then the transformed new covariance function is $C(\mathbf{z}_i, \mathbf{z}_j) = \text{cov}(g(\mathbf{z}_i), g(\mathbf{z}_j)) = \int C_2(\mathbf{z}_i, \mathbf{y}_i)C_1(\mathbf{y}_i, \mathbf{y}_j)C_2(\mathbf{z}_j, \mathbf{y}_j)d\mathbf{y}_j d\mathbf{y}_i$.

**Warping—nonlinear mapping of covariance function**

Another possibility is to employ an arbitrary nonlinear mapping, also known as warping of the input to handle the nonstationary nonlinearity of the function in tandem with a stationary covariance function. More on warping using the parametric functions can be found in [8] and using the nonparametric functions in [9]. This method can be also used to transform data when the noise that corrupts measurement data does not have Gaussian distribution.

Composite covariance functions that are suited to the problem at hand can be developed using listed operations. An overview [57] gives further directions for the selection and combination of covariance functions. Examples of customary covariance functions can be found in [58–62].

## 2.4 GP Model Selection

This section explains how we get from the Bayesian model selection to the optimisation of the hyperparameters for the selected covariance function explained in Sect. 2.3 so that the GP model is a model of an unknown system.

Due to the probabilistic nature of the GP model, the model optimisation approach where the model parameters and the structure are optimised by the minimisation of a loss function defined in terms of model error only is not really applicable. Since the GP is a Bayesian probabilistic method, a probabilistic approach to the model selection is appropriate. The probabilistic approach to model selection is described in the following subsection. According to [49], the term model selection covers the selection of a covariance function for a particular model, the selection of the

mean function, the choice of the hyperparameters and a comparison across different families of models.

Several possibilities for hyperparameter determination exist. A very rare possibility is that the hyperparameters are known in advance as prior knowledge. Almost always, however, they must be determined from the training data.

In the following subsections, the background for hyperparameter selection is described. First, the evidence or marginal likelihood maximisation is described. The next subsection deals with the mathematical and computational implementations, where the methods can be divided into direct and approximate implementations.

The alternative to the case where the set of identification data is available in advance is when the identification data is collected online. This alternative is practical when the unknown system is modelled as a time-varying model or as a method to circumvent the computational burden due to a large amount of data being used for the identification. In both cases, only the data with sufficient information content is selected for the training process.

### 2.4.1 Bayesian Model Inference

Bayesian inference is a process in which a prior on the unknown quantity, i.e. an input–output mapping function $f$, has to be specified. Next, the identification data is observed. Afterwards, a posterior distribution over $f$ is computed that refines the prior by incorporating the observations, i.e. the identification data.

For example, the identification data for the GP-NARX model is

$$\mathcal{D} = \{(\mathbf{Z}, \mathbf{y})\},$$
$$\mathbf{y} = \begin{bmatrix} y(k) \\ \vdots \\ y(k+i) \\ \vdots \\ y(k+N) \end{bmatrix}, \mathbf{Z} = \begin{bmatrix} \mathbf{z}_1, \ \ldots, \ \mathbf{z}_i, \ \ldots, \mathbf{z}_N \end{bmatrix},$$
$$\mathbf{z}_i = [y(k+i-1, \ldots, y(k+1-n), u(k+i-1), \ldots, u(k+i-m)]^{\mathrm{T}}.$$

We set a GP prior for a function $f$, i.e. a presumption on the distribution of the model candidates. We specify the prior mean function and the prior covariance function. The prior mean function is often set as $m_f = 0$.

Inference of the GP posterior [51] can be pursued in more levels based on the unknowns to be inferred. There are at least two levels above the data $\mathcal{D} = \{(\mathbf{Z}, \mathbf{y})\}$: one for inferring the distribution over function $f$ and the second, on the top of it, to determine the hyperparameters $\boldsymbol{\theta}$ that specify the distribution over the function values $f$. A further model inference level can be the level of different covariance functions.

We start with the GP posterior of the function presuming that the data and the hyperparameters are given:

$$p(f|\mathbf{Z}, \mathbf{y}, \boldsymbol{\theta}) = \frac{p(\mathbf{y}|f, \mathbf{Z}, \boldsymbol{\theta})p(f|\boldsymbol{\theta})}{p(\mathbf{y}|\mathbf{Z}, \boldsymbol{\theta})}, \tag{2.25}$$

where $p(\mathbf{y}|f, \mathbf{Z}, \boldsymbol{\theta})$ is the likelihood of the function $f$ and $p(f|\boldsymbol{\theta})$ is the GP prior on $f$.

The likelihood of the function $f$, with the assumption that the observations, i.e. measurements, $y_i$ are conditionally independent given $\mathbf{Z}$, is [51]

$$\begin{aligned} p(\mathbf{y}|f, \mathbf{Z}, \boldsymbol{\theta}) &= p(y_1, y_2, \ldots, y_N|f(\mathbf{Z}), \mathbf{Z}, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i|f(\mathbf{z}_i), \boldsymbol{\theta}) \\ &= \prod_{i=1}^{N} \mathcal{N}(y_i|f(\mathbf{z}_i), \sigma_n^2) = \mathcal{N}(\mathbf{y}|f(\mathbf{Z}), \sigma_n^2\mathbf{I}), \end{aligned} \tag{2.26}$$

where $\sigma_n^2$ is a variance of additive noise on the output identification data. The likelihood in Eq. (2.26) encodes the assumed noise model.

The normalising constant in Eq. (2.25)

$$p(\mathbf{y}|\mathbf{Z}, \boldsymbol{\theta}) = \int p(\mathbf{y}|\mathbf{Z}, f, \boldsymbol{\theta})p(f|\boldsymbol{\theta})df \tag{2.27}$$

is the evidence or marginal likelihood. The evidence is the likelihood of the hyperparameters given the data after having marginalised out the function $f$.

Because we do not have the hyperparameters given, we have to infer them. The next level is, therefore, to infer a posterior probability distribution over the hyperparameters $\boldsymbol{\theta}$ that is conditional on the data $\mathcal{D}$:

$$p(\boldsymbol{\theta}|\mathbf{Z}, \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{Z}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{y}|\mathbf{Z})}, \tag{2.28}$$

where $p(\boldsymbol{\theta})$ is the prior on the hyperparameters.

The evidence of Eq. (2.28) is

$$p(\mathbf{y}|\mathbf{Z}) = \int p(\mathbf{y}|\mathbf{Z}, \boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{2.29}$$

where we marginalise out the hyperparameters $\boldsymbol{\theta}$.

Note that in Bayesian inference there is no model overfitting to the identification data common to other methods mentioned at the beginning of this chapter. This is because in essence there is no data fitting in the Bayesian inference.

However, the integral in Eq. (2.29) is analytically intractable in most interesting cases. A possible solution is to use numerical approximation methods, such as the Markov chain Monte Carlo method, to obtain the posterior. Unfortunately, significant computational efforts may be required to achieve a sufficiently accurate approximation.

In addition to the numerical approximation methods, another standard and general practice for estimating the hyperparameters is the evidence maximisation with regards to the hyperparameters. This approximation means that instead of a posterior distribution over the hyperparameters $p(\boldsymbol{\theta}|\mathbf{Z}, \mathbf{y})$ we look for a point estimate $\hat{\boldsymbol{\theta}}$. This method has several names, among others it is called empirical Bayes or type-2 maximum likelihood [44]. Nevertheless, with this approximation we are not completely in line with the coherent Bayesian inference anymore.

Hyperparameters estimation via evidence maximisation is described in the next section.

### 2.4.2 Marginal Likelihood—Evidence Maximisation

Using evidence maximisation is based on the presumption that the most likely values of the hyperparameters are noticeably more likely than other values. This means that the posterior distribution of hyperparameters is unimodal and is described with a narrow Gaussian function. This is usually valid for a larger number of identification datapoints relative to the number of hyperparameters. For the smaller number of datapoints relative to the number of hyperparameters, the posterior distribution is usually not unimodal.

The values of the hyperparameters depend on the data-at-hand and it is difficult to select their prior distribution. If a uniform prior distribution is selected, which means that any values for the hyperparameters are equally possible a priori, then the hyperparameters' posterior distribution is proportional to the marginal likelihood in Eq. (2.25), that is,

$$p(\boldsymbol{\theta}|\mathbf{Z}, \mathbf{y}) \propto p(\mathbf{y}|\mathbf{Z}, \boldsymbol{\theta}). \tag{2.30}$$

This means that the maximum a posteriori (MAP) estimate of the hyperparameters $\boldsymbol{\theta}$ equals the maximum marginal likelihood estimate of Eq. (2.27) [51].

The hyperparameters $\boldsymbol{\theta}$ are therefore obtained with the maximisation of evidence or marginal likelihood in Eq. (2.25) on the first level of inference with respect to the hyperparameters:

$$p(\mathbf{y}|\mathbf{Z}, \boldsymbol{\theta}) = \frac{1}{(2\pi)^{\frac{N}{2}} |\mathbf{K}|^{\frac{1}{2}}} e^{-\frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{y}}. \tag{2.31}$$

with the $N \times N$ covariance matrix $\mathbf{K}$ of the identification or training data.

Due to the mathematical properties of the logarithm function for the numerical scaling purposes, the logarithm of the evidence is used as the objective function for the optimisation:

$$\ln p(\mathbf{y}|\mathbf{Z}, \boldsymbol{\theta}) = \ell(\boldsymbol{\theta}) = -\overbrace{\frac{1}{2}\ln(|\mathbf{K}|)}^{\text{complexity term}} - \overbrace{\frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{y}}^{\text{data-fit term}} - \overbrace{\frac{N}{2}\ln(2\pi)}^{\text{normalisation const.}}, \tag{2.32}$$

where $\ell$ is the value of the logarithm of the evidence or marginal likelihood. Equation (2.32) also shows interpretations of its three terms.

The posterior model we obtain with evidence maximisation trades off data fit and model complexity. Hence, it avoids overfitting [49, 51] by implementing *Occam's razor*, which tells us to use the simplest model that explains the data.

Maximising the evidence using Eq. (2.32) is a nonlinear, non-convex *optimisation* problem. Nevertheless, even though a global optimum of this optimisation problem has not necessarily been found, the GP model can always explain the data [51].

The fact that the posterior GP model is obtained with optimisation can be, on one hand, considered as a disadvantage, because Bayesian inference is not pursued strictly all the way to the end. On the other hand, however, a lot of people involved in system identification are very familiar with optimisation-based methods for experimental modelling, which brings GP modelling closer to their attention. This might not be the case with full Bayesian inference.

The following sections describe some possibilities for the optimisation implementation.

#### 2.4.2.1 Deterministic Optimisation Methods

Deterministic optimisation methods are well known and are widely used in system identification. Here, we are not going into a detailed description. An overview of these methods can be found in many references, e.g. [10].

In the case of using one of the gradient optimisation methods, the computation of the partial derivatives of marginal likelihood with respect to each of the hyperparameters is required:

$$\nabla(\ell(\boldsymbol{\theta})) = \frac{\partial \ln p(\mathbf{y}|\mathbf{Z}, \boldsymbol{\theta})}{\partial \theta_i} = -\frac{1}{2}\text{trace}\left(\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta_i}\right) + \frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta_i}\mathbf{K}^{-1}\mathbf{y}. \tag{2.33}$$

The computation of the partial derivatives involves the computation of the inverse of the $N \times N$ covariance matrix $\mathbf{K}$ during every iteration. This means that the computational complexity is $\mathcal{O}(N^3)$. However, there are alternative approaches, and some of them will be touched upon in Sect. 2.5.2.

A frequently used method for optimising the cost function is a conjugate gradient method—a Polack–Ribiere version utilised in tandem with the Wolfe-Powell stopping conditions [49]. Compared with numerical approximation methods of the Monte Carlo type used for obtaining the hyperparameters' posterior, this conjugate gradient approach can find a reasonable approximation to a local maximum after a relatively acceptable number of evaluations.

The trust-region optimisation method is proposed as an alternative by [63], where the Hessian matrix is simplified and then the trust-region algorithm is used for the GP hyperparameters' optimisation.

A property of deterministic optimisation methods is that their result heavily depends on the initial values of the hyperparameters, especially for complex

multidimensional systems, where the objective function has many local optima. Therefore, whatever optimisation method is used it should be run repeatedly with various initial values of the hyperparameters. While the space of possible values is huge, the initial values are often chosen randomly. Therefore, stochastic optimisation methods can be considered as an alternative approach. Three stochastic algorithms are described in the next subsection: genetic algorithms, differential evolution and particle swarm optimisation.

#### 2.4.2.2 Stochastic Optimisation Methods

Stochastic optimisation methods are used as an alternative to deterministic optimisation methods when local extremes are expected in the optimisation of hyperparameters. The following description is adopted from [64].

Evolutionary algorithms (EAs) are generic, population-based, stochastic optimisation algorithms inspired by biological evolution. They use mechanisms similar to those known from the evolution of species: reproduction, mutation, recombination and selection. Candidate solutions to the optimisation problem play the role of individuals in a population, and the objective function determines the environment within which the solutions 'live'. Simulated evolution of the population then takes place after the repeated application of the above operators.

Evolutionary algorithms [65] perform well at approximating solutions to diverse types of problems because they make no assumptions about the underlying fitness landscape; this generality is shown by their success in fields as diverse as science, engineering, economics, social sciences and art.

In most real-world applications of evolutionary algorithms, computational complexity is a prohibiting factor. In fact, this computational complicity is due to a objective function evaluation. In our case, as was shown in the previous section, an evaluation of the objective function contains an inversion of the covariance matrix, of which the computational time rises with the third power of the amount of data. However, this 'inconvenience' is unfortunately inescapable without an approximation.

The most commonly used evolutionary algorithms for numerical optimisation, such as maximisation of the logarithmic marginal likelihood, are genetic algorithm with real numbers representation, differential evolution and particle swarm optimisation.

**Genetic algorithm (GA)** is a flexible search technique [65] used in computing to find exact or approximate solutions to optimisation and search problems in many areas. Traditionally, the solutions are represented as binary strings of 0s and 1s, but other encodings are also possible. The simulated evolution usually starts from a population of randomly generated individuals and proceeds in generations. In each generation, the fitness of every individual in the population is evaluated, and multiple individuals are stochastically selected from the current population (based on their fitness) and modified (recombined and possibly randomly mutated)

to form a new population. The new population is then used in the next generation of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations have been iterated, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached. Examples of GP model optimisation with a GA can be found in [66, 67].

**Differential evolution (DE)** is a method for numerical optimisation without explicit knowledge of the gradients. It was presented by Storn and Price [68] and works on multidimensional real-valued functions that are not necessarily continuous or differentiable. DE searches for a solution to a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae of vector crossover and mutation, and then keeping whichever candidate solution has the best score or fitness on the optimisation problem at hand. In this way, the optimisation problem is treated as a black box that merely provides a measure of quality given a candidate solution and the gradient is therefore not needed. More details about DE can be found in [69].

Example of GP model optimisation with DE can be found in [64].

**Particle swarm optimisation (PSO)** is a method proposed by Kennedy and Eberhart [70] that is motivated by the social behaviour of organisms such as bird flocking and fish schooling. Like DE it is used for numerical optimisation without explicit knowledge of the gradients. PSO provides a population-based search procedure in which individuals called particles change their position (state) with time. In a PSO system, particles 'fly' around in a multidimensional search space. During flight, each particle adjusts its position according to its own experience and the experience of a neighbouring particle, making use of the best position encountered by itself and its neighbour. Thus, a PSO system combines local search with global search, attempting to balance exploration and exploitation. Further details about PSO can be found in [71].

Examples of GP model optimisation with a PSO can be found in [64, 72].

*Example 2.1* (*$CO_2$ concentration modelling*) This example compares different stochastic optimisation methods and is adopted from [64].

To assess the potential of evolutionary algorithms in the optimisation of GP model hyperparameters, a problem concerning the concentration of $CO_2$ in the atmosphere from [49] was chosen. The data consists of monthly average atmospheric $CO_2$ concentrations derived from in situ air samples collected at the Mauna Loa Observatory, Hawaii, between 1959 and 2009 (with some missing data).[1] The goal is to model the $CO_2$ concentration as a function of time.

Although the data is one-dimensional, and therefore easy to visualise, a complex covariance function is used. It is derived by combining several kinds of simple covariance functions. First, a squared exponential covariance function in Eq. (2.13) is used to model the long-term smooth rising trend. With the product of a periodic

[1] The data is available from http://cdiac.esd.ornl.gov/ftp/trends/co2/maunaloa.co2.

of Eq. (2.19) and the squared exponential covariance function of Eq. (2.13), a seasonal component is modelled. To model the medium-term irregularities, a rational quadratic covariance function in Eq. (2.16) is used. Finally, the noise is modelled as the sum of a squared exponential and a constant covariance function of Eq. (2.12).

$$
\begin{aligned}
C(\mathbf{z}_i, \mathbf{z}_j) &= C_1(\mathbf{z}_i, \mathbf{z}_j) + C_2(\mathbf{z}_i, \mathbf{z}_j) + C_3(\mathbf{z}_i, \mathbf{z}_j) + C_4(\mathbf{z}_i, \mathbf{z}_j), \\
C_1(\mathbf{z}_i, \mathbf{z}_j) &= \theta_{12}^2 \exp\left(-\frac{r^2}{2\theta_{11}^2}\right), \\
C_2(\mathbf{z}_i, \mathbf{z}_j) &= \theta_{22}^2 \exp\left(-\frac{r^2}{2\theta_{21}^2}\right) \theta_{24}^2 \exp\left(-\frac{2}{\theta_{23}^2} \sin^2\left(\frac{\pi}{\theta_{25}} r\right)\right), \\
C_3(\mathbf{z}_i, \mathbf{z}_j) &= \theta_{32}^2 \exp\left(1 + \frac{r^2}{2\theta_{33}\theta_{31}^2}\right)^{-\theta_{33}}, \\
C_4(\mathbf{z}_i, \mathbf{z}_j) &= \theta_{42}^2 \exp\left(-\frac{r^2}{2\theta_{41}^2}\right) + \theta_{43}^2 \delta_{ij}.
\end{aligned}
\tag{2.34}
$$

This complex covariance function involves 13 hyperparameters. Note that in [49] the covariance function involves only 11 hyperparameters due to the fixed period of the periodic covariance function to one year.

For differential evolution an implementation from [69], for particle swarm optimisation an implementation from [73], and for genetic algorithms an implementation from [74] were used. These methods were compared to a commonly used deterministic conjugate gradient method (CG) as a frequently used deterministic optimisation method for GP modelling.

All the stochastic algorithms had the same population size, number of generations and number of solution evaluations. The population size was set to 50 individuals, the number of generations to 2000 and the number of iterations to 10. Although the tested algorithms have various properties, the fairness of experimental evaluation was tried to be guaranteed. The parameters of these algorithms were tuned in preliminary experiments. For a comparison of the tested stochastic methods with the conjugate gradient method, the same number of evaluations was used with it as well. Thus the conjugate gradient method was executed 10 times with 100,000 solution evaluations available. That means, in one iteration, the conjugate gradient methods are repeatedly executed with random initial values and possibly restarted until all the available evaluations are spent.

For each algorithm the following statistics were calculated: minimum, maximum, average and standard deviation. They are given in Table 2.1.

Figure 2.19 shows the performance traces of each algorithm averaged over 10 runs. At first sight, it appears that the DE and PSO perform similarly and a lot better than the GA and CG. The DE and PSO reached a very similar maximum value that was very close to the result from [49]. However, the PSO reached a lower minimum value, which means it has a larger variance than the DE. In other words, using the DE will more probably find an optimal value or at least a value near to it.

**Table 2.1** Statistical values—maximum, minimum, average and standard deviation—of 10 iterations for each algorithm: CG, GA, DE, PSO

| | CG | GA | DE | PSO |
|---|---|---|---|---|
| Maximum | −598.8627 | −639.0114 | −142.2658 | −142.4529 |
| Minimum | −638.9760 | −703.9570 | −176.2658 | −216.9546 |
| Average | −630.9088 | −645.8582 | −154.5382 | −177.6854 |
| Standard deviation | 12.5953 | 20.4178 | 11.3083 | 26.9605 |

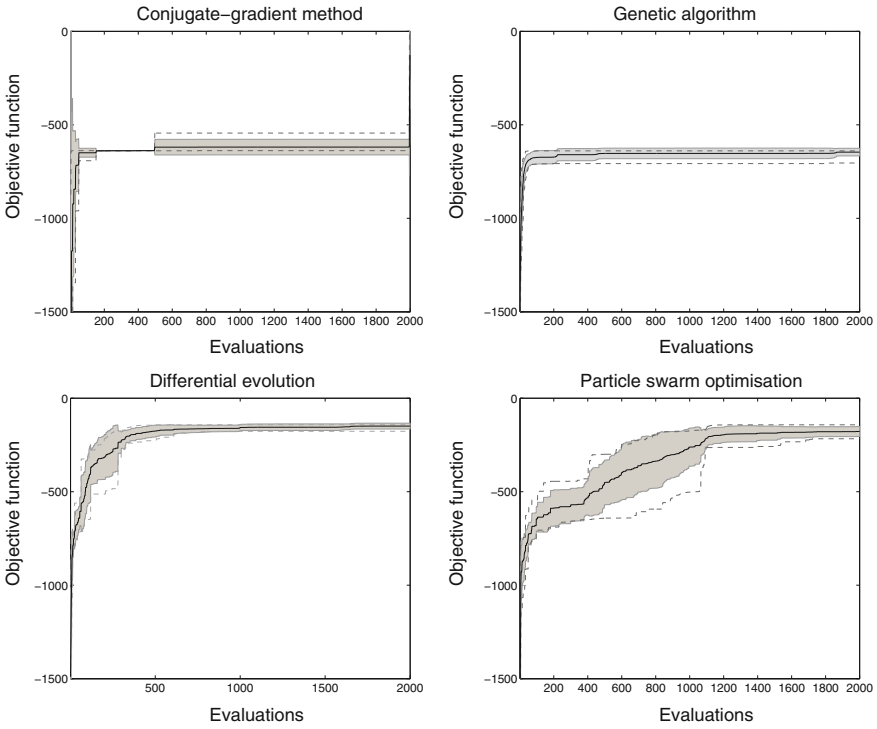


**Fig. 2.19** Performance traces for the tested algorithms. *Solid lines* denote mean values, *dashed lines* denote maximum and minimum values, *grey areas* present standard deviations

Unsatisfactory results obtained by the CG imply the difficulty of the chosen problem for this traditional method.

However, Fig. 2.20, which shows the predictions of $CO_2$ for the next 20 years based on the model obtained from the best hyperparameter values found with DE and shown in Table 2.2, confirms the superiority of the DE. While the best hyperparameter values obtained by the PSO differ from the DE's at most by $10^{-5}$, consequently the

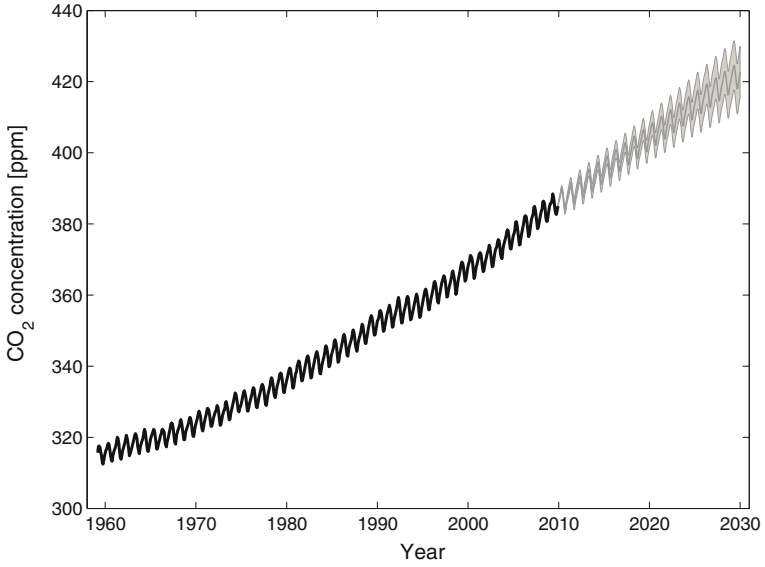


**Fig. 2.20** Concentration of $CO_2$ in the atmosphere and its predictions based on a GP model with hyperparameters optimised using differential evolution

**Table 2.2** Best hyperparameters of our experiment, which were obtained with the DE

| $i$ | $\theta_{i1}$ | $\theta_{i2}$ | $\theta_{i3}$ | $\theta_{i4}$ | $\theta_{i5}$ |
|---|---|---|---|---|---|
| 1 | 5.0000 | 5.0000 | | | |
| 2 | 5.0000 | −1.5668 | 0.2406 | 2.1831 | 2.4843 |
| 3 | 1.3454 | −0.7789 | 5.0000 | | |
| 4 | 0.4580 | −1.7163 | −1.6161 | | |

log-marginal likelihood and the predictions of the PSO are very similar to those of the DE.

Please note that our predictions are almost identical to the originals from [49], but the log-marginal likelihood is smaller, due to the wider range of measurements for training and the wider range of predictions used in our experiment.

### *2.4.3 Estimation and Model Structure*

Up until now, we have not been taking into account the different model structures, as listed in Sect. 2.3.1.

Let us explore the differences between the NARX and NOE general kinds of models, not only GP models in particular, with an emphasis on modelling for simulation,

and therefore training in a parallel configuration as described in [18]. This is of particular interest in dynamic systems identification. The NFIR model can in this context be taken as a special case of the NARX model.

Consider again the system with added white Gaussian noise

$$y(k) = f(\mathbf{z}(k)) + \nu \tag{2.35}$$

where $\mathbf{z}$ is a vector of regressors at the time instant $k$ and $\nu$ is the measurement noise, also the observation noise, at output $y$ at the time instant $k$. If there is no noise in the output measurements, then the NARX and NOE models are the same and the differences between them do not matter. Nevertheless, noise always exists in real-life measurements. In the NARX model, these measurements are contained within the regressors that are delayed samples of the output signal. This means that when the NARX model is identified the *errors-in-variables* problem occurs. There are structures with other noise models that are more complex.

The problem of learning GP models with noisy input variables is investigated in [75]. Another possible solution to this problem is the automated filtering of input and output signals. This solution is equivalent to the optimisation of a noise filter, i.e. a filter that filters white noise to the one that corresponds to the measurement noise. In system identification this method is known as the *instrumental variables* method [2]. A similar principle in the context of GP models is described in [19].

If, on the other hand, the model is trained without using delayed measurements as the regressors, i.e. an output-error model, then it is assumed that the noise is entering the output signal after the system.

For a dynamic system of order $n$, the one-step-ahead prediction is calculated with the previous process output values as described with Eq. (2.4)

$$\begin{aligned}\hat{y}(k) = f(&y(k-1), y(k-2), \ldots, y(k-n), u(k-1),\\ &u(k-2), \ldots, u(k-m)) + \nu,\end{aligned}$$

while the simulation is evaluated with the previous model's output estimates as described with Eq. (2.5)

$$\begin{aligned}\hat{y}(k) = f(&\hat{y}(k-1), \hat{y}(k-2), \ldots, \hat{y}(k-n), u(k-1),\\ &u(k-2), \ldots, u(k-m)) + \nu.\end{aligned}$$

In the first case, we have a feedforward prediction, while in the second case we have a recurrent one and, in the case of nonlinear system's modelling with GP models, an approximation of the prediction distribution. A simulation is required whenever the process output cannot be measured during the operation, which is always when the system is simulated independently from the real system. This is frequently the case for the design of fault detection and control systems.

Nevertheless, the NARX model is by far the most applied model for dynamic systems, not because it is realistic, but because it is easier to train than the NOE model.

In the NARX case, the model is trained based on loss functions that are dependent on the prediction error, while in the NOE case the model is trained based on loss functions that are dependent on the simulation error. The hyperplane of the loss function is much more complicated in the second case. However, the model that is used in the parallel configuration does not necessarily have to be trained in the parallel configuration as well, if the noise assumptions are relaxed, which is often the case in engineering practice. The disadvantage of using NARX models for the simulation is the error accumulation, which does not happen with the prediction. In extreme situations, the NARX model used for the simulation can become unstable, regardless of a satisfactory one-step prediction performance.

The trade-offs between the advantages and disadvantages of the NARX and NOE models need to be evaluated when the model is developed for a model simulation.

The case when GP models are used is specific. Since the output of the GP model $\hat{y}(k)$ is a normal distribution in the case of the NOE model, this output represents only an approximation of the distribution that should appear on the output of a nonlinear model. The level of the approximation can be very different, as will be discussed in Sect. 2.6. It can range from the simplest case, where only expectations of the outputs are fed back, to more informative analytical or numerical approximations.

General GP optimisation algorithms for parallel-series or the GP-NARX model described with Eq. (2.4) and for parallel or GP-NOE model in Eq. (2.5) are as follows.

*Optimisation algorithm for GP-NARX model*:

---

**Algorithm:** OPTIMISENARXMODEL(*Inputs*)

---

set input data, target data, covariance function, initial hyperparameters
**repeat**
  calculate $-\ell(\boldsymbol{\theta})$ and its derivative based on input data
  $\{y(k-1), y(k-2), \ldots, y(k-n), u(k-1), u(k-2), \ldots, u(k-m)\}$
  change hyperparameters
**until** $-\ell(\boldsymbol{\theta})$ is minimal

---

*Optimisation algorithm for GP-NOE model*:

---

**Algorithm:** OPTIMISENOEMODEL(*Inputs*)

---

set input data, target data, covariance function, initial hyperparameters
**repeat**
  calculate the model simulation response
  calculate $-\ell(\boldsymbol{\theta})$ and its derivative based on input data
  $\{\hat{y}(k-1), \hat{y}(k-2), \ldots, \hat{y}(k-n), u(k-1), u(k-2), \ldots, u(k-m)\}$
  change hyperparameters
**until** $-\ell(\boldsymbol{\theta})$ is minimal

---

The difference between considering the prediction and simulation error has a major impact on the complexity of the optimisation. This difference is illustrated in the following example.

*Example 2.2* (*Difference in GP-NARX and GP-NOE optimisation*) The difference in the optimisation for the GP-NARX and GP-NOE models is illustrated with an example where two loss functions are calculated. The input and output data is obtained from the selected dynamic GP model of the first order. The regressors are delayed samples of the output and input signals, $y(k-i)$ and $u(k-i)$, respectively, in the case of the GP-NARX model; and delayed samples of expectations of the output predictions $E(\hat{y}(k))$ and delayed samples of the input signals $u(k-i)$ in the case of the GP-NOE model. This means that the simplest possible approximation has been used in this example where no prediction distribution is propagated through the model. Even though this is the simplest approximation, the main differences between the model structures are noticeable.

Since there are two regressors and the GP model uses squared exponential and constant covariance functions, there are four hyperparameters in this process.

We fix the two hyperparameters that are parameters that control the variances $\sigma_f^2$ and $\sigma_n^2$ and calculate the surfaces of the loss functions for the GP-NARX and GP-NOE models, when the two remaining hyperparameters controlling the two regressors are varied. The loss function is a negative logarithm of the marginal likelihood in both cases, but in the case of the GP-NARX model the output observations are used for the calculation of the loss function and in the case of the GP-NOE model the mean values of the output predictions are used for the calculation of the loss function. The results can be seen in Fig. 2.21.

Figure 2.21 shows that not only the loss functions surfaces, but also the optimal hyperparameters for the GP-NARX and GP-NOE models are significantly different. It is apparent that the optimisation of the model parameters for the prediction is a much easier task than the optimisation of the model parameters for simulation.

### 2.4.4 Selection of Mean Function

We have mentioned in several places that the zero mean function is often presumed as the prior mean function of the GP model. This is quite common with system identification where the identification and validation data is preprocessed to ensure, among others, a zero mean. Nevertheless, this is not necessary and in some cases an explicit modelling of the mean function is required. The mean function can be specified using explicit basis functions. The following text is summarised from [49], but the topic is also elaborated in [76].

There are three main ways that the mean function can be used together with GP models:

1. The mean function is fixed and deterministic, which is usually the case when it is determined before the identification. An example would be when the data

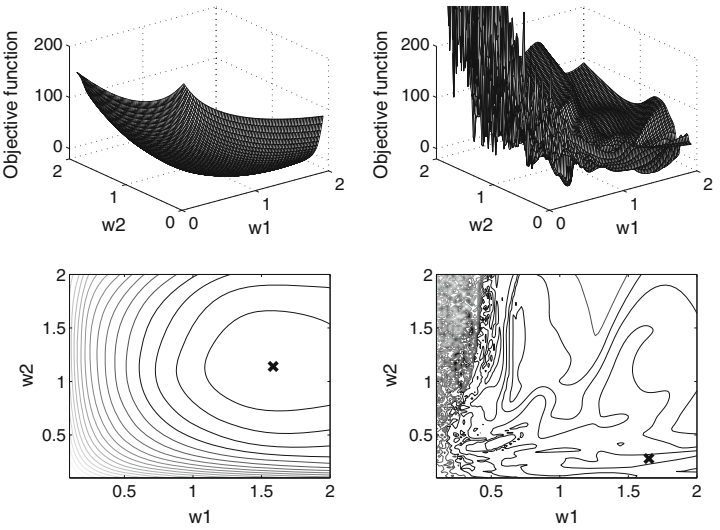


**Fig. 2.21** NARX (*left*) and NOE (*right*) model loss functions (*top*) and their contour plots (*bottom*) with *black crosses* at the positions of the minima

is detrended during the preprocessing. In this case, the usual zero mean GP is applied to the difference between the measurements and the fixed mean function. Using

$$f(\mathbf{z}) \sim \mathcal{GP}(m_f(\mathbf{z}), C(\mathbf{z}_i, \mathbf{z}_j)), \tag{2.36}$$

the predictive mean becomes

$$E(y_f^*) = m_f(\mathbf{z}^*) + \mathbf{k}(\mathbf{z}^*)\mathbf{K}^{-1}(\mathbf{y} - m_f(\mathbf{z})), \tag{2.37}$$

where $\mathbf{K} = \boldsymbol{\Sigma}_f + \sigma_n^2\mathbf{I}$ and the predictive variance is according to Eq. (1.25).

2. The mean function is identified as a deterministic function that is a combination of a few preselected fixed basis functions with the coefficients collected in vector $\boldsymbol{\beta}$. We can write

$$g(\mathbf{z}) = f(\mathbf{z}) + \boldsymbol{\phi}(\mathbf{z})^{\mathrm{T}}\boldsymbol{\beta}, \quad \text{where } f(\mathbf{z}) \sim \mathcal{GP}(0, C(\mathbf{z}_i, \mathbf{z}_j)), \tag{2.38}$$

where $f(\mathbf{z})$ is a zero mean GP, $\boldsymbol{\phi}(\mathbf{z})$ is a vector of fixed basis functions and $\boldsymbol{\beta}$ are the additional coefficients to be identified together with the hyperparameters.

3. The mean function is identified as a stochastic function with a Gaussian prior on $\boldsymbol{\beta}$ so that $p(\boldsymbol{\beta}) = \mathcal{N}(\mathbf{b}, \mathbf{B})$. We get another GP with the covariance function expanded with uncertainty in the parameters of the mean:

$$g(\mathbf{z}) \sim \mathcal{GP}(\boldsymbol{\phi}(\mathbf{z})^{\mathrm{T}}\mathbf{b}, C(\mathbf{z}_i, \mathbf{z}_j) + \boldsymbol{\phi}(\mathbf{z}_i)^{\mathrm{T}}\mathbf{B}\boldsymbol{\phi}(\mathbf{z}_j)^{\mathrm{T}}). \tag{2.39}$$

The objective function that is used for the identification of the hyperparameters and the mean function parameters is expanded from Eq. (2.32) with $\mathbf{B}$ set to 0 in the case of deterministic $\beta$:

$$\begin{aligned}&\ln p(\mathbf{y}|\mathbf{Z}, \mathbf{b}, \mathbf{B})\\ &= -\frac{1}{2}\ln(|\mathbf{K} + \mathbf{\Phi}^{\mathrm{T}}\mathbf{B}\mathbf{\Phi}|) - \frac{1}{2}(\mathbf{\Phi}^{\mathrm{T}}\mathbf{b} - \mathbf{y})^{\mathrm{T}}(\mathbf{K} + \mathbf{\Phi}^{\mathrm{T}}\mathbf{B}\mathbf{\Phi})^{-1}(\mathbf{\Phi}^{\mathrm{T}}\mathbf{b} - \mathbf{y})\\ &\quad - \frac{N}{2}\ln(2\pi),\end{aligned} \tag{2.40}$$

where the matrix $\mathbf{\Phi}$ collects the vectors $\phi(\mathbf{z})$ for all the identification regression vectors.
The predictions are obtained as

$$\begin{aligned}E(\mathbf{y}_g^*) &= E(\mathbf{y}_f^*) + \mathbf{R}^{\mathrm{T}}E(\beta),\\ \mathrm{cov}(\mathbf{y}_g^*) &= \mathrm{cov}(\mathbf{y}_f^*) + \mathbf{R}^{\mathrm{T}}(\mathbf{B}^{-1} + \mathbf{\Phi}(\mathbf{Z})\mathbf{K}^{-1}\mathbf{\Phi}(\mathbf{Z})^{\mathrm{T}})\mathbf{R},\end{aligned} \tag{2.41}$$

where $\mathbf{R} = (\mathbf{\Phi}(\mathbf{Z}^*) - \mathbf{\Phi}(\mathbf{Z})\mathbf{K}^{-1}\mathbf{k}(\mathbf{Z}^*, \mathbf{Z}))$ and $E(\beta) = (\mathbf{B}^{-1} + \mathbf{\Phi}(\mathbf{Z})\mathbf{K}^{-1}\mathbf{\Phi}(\mathbf{Z})^{\mathrm{T}})^{-1}(\mathbf{\Phi}(\mathbf{Z})\mathbf{K}^{-1}\mathbf{y} + \mathbf{B}^{-1}\mathbf{b})$.

In cases when the prior knowledge about $\mathbf{B}$ is vague, which means that $\mathbf{B}^{-1}$ approaches to the matrix of zeros, which is elaborated in [49].

### 2.4.5 Asymptotic Properties of GP Models

In model identification, it is important that the model is consistent. This means that the posterior distribution concentrates around the true distribution of the parameters as more and more data is observed or the sample size increases. The theory on consistency of GP models with sufficient conditions for the *posterior consistency* of GP regression is explained in [76] and reviewed in [49], with further references describing studies in various general settings in both listed references.

As stated in [76], these sufficient conditions are difficult to validate and may not be intuitive when applied to concrete models. In [77], the alternative concept of the so-called *information consistency* for GP regression models is described. The consistency of the information is checked with the information criterion that is the Césaro average of the sequence of prediction errors. See [76, 77] for more details and the background theory.

## 2.5 Computational Implementation

A noticeable drawback of the system identification with GP models is the computation time necessary for the modelling. GP regression, on which system identification is based, involves several matrix computations. This increases the number of operations with the third power of the number of input data, i.e. $\mathcal{O}(N^3)$, such as matrix inversion and the calculation of the log determinant of the used covariance matrix. This computational greed restricts the number of training data to at most a few thousand cases on modern workstations.

A common approach to computing the objective function from Eq. (2.32) and its gradient makes use of the Cholesky decomposition [49] of $\mathbf{K}$ to compute $\boldsymbol{\alpha} = \mathbf{K}^{-1}\mathbf{y}$.

The training algorithm in pseudocode is as follows:

---

**Algorithm:** GP TRAINING($\mathbf{Z}$, $y$, $C$, initial $\boldsymbol{\theta}$)

---

**repeat**
  change hyperparameters $\boldsymbol{\theta}$
  compute $\mathbf{K}(\boldsymbol{\theta}) = [C(\mathbf{z}_p, \mathbf{z}_q)]_{N\times N}$
  compute Cholesky decomposition $\mathbf{L} = \text{chol}(\mathbf{K})$
  solve $\mathbf{L}\boldsymbol{\gamma} = \mathbf{y}$ for $\boldsymbol{\gamma}$ and $\mathbf{L}^T\boldsymbol{\alpha} = \boldsymbol{\gamma}$ for $\boldsymbol{\alpha}$ to get $\boldsymbol{\alpha} = \mathbf{K}^{-1}\mathbf{y}$
  compute $\ell(\boldsymbol{\theta})$ and $\nabla\ell(\boldsymbol{\theta})$ using $\boldsymbol{\alpha}$
**until** $-\ell(\boldsymbol{\theta})$ is minimal

---

### *2.5.1 Direct Implementation*

One option to deal with the computational implementation is to approach the computation problem from the utilised hardware technology point of view. Since hardware capabilities are increasing every day, this approach might seem inefficient when looking over the longer term, but it is undoubtedly effective in the short term.

Parallel processing [78] is a popular way to deal with a large amount of data and a large number of computations. The authors of [79] envision the future computing systems as being hybrid computers, where the two major types of integrated components will be *multi-core central processing units (CPUs)* and *massively parallel accelerators*. While the 'standard' CPUs will continue to provide users with more and more computing power [80], many computer scientists will migrate towards general-processing graphics processing unit (GPGPU) applications [81], using graphics-card processors as the parallel accelerators for memory-dense, floating-point-intensive applications. The graphics processing unit (GPU) is, therefore, currently a low-cost, high-performance computing alternative to larger, stand-alone, parallel computer systems. An accelerated linear algebra package exploiting the hybrid computation paradigm is currently under development [82].

The concept of a GPGPU processor evolved from the needs of 3D-graphics-intensive applications. This need dictated the design of a processor such that more transistors were dedicated to the data processing than to the control and data caching, as in a regular CPU. Next, the processor was designed to be able to execute a data-parallel algorithm on a stream of data, which is the reason why GPGPU processors are sometimes called 'stream processors'. The currently dominant architectures for GPGPU computing are the nVidia CUDA [83] and the AMD APP (formerly ATI Stream) [84].

Some computation acceleration results for kernel methods are reported in [85, 86], while some preliminary results for GP modelling computation with a GPU are reported in [87].

The intrinsic parallel structure of a GPU allows a significant speed-up in comparison to the single-processor architecture. Although it is relatively easy to set up and perform basic operations, it quickly becomes more complex when dealing with more demanding numerical problems. Additionally, special care must be taken when performing memory operations:

- due to the relatively slow memory transfer, data transfers between the host system and the GPU device shall be as few as possible, and shall be asynchronous if possible;
- improper kernel code design with respect to the operation on different memory types (main GPU memory, shared memory, constant memory, texture memory) and ignoring memory access coalescing on the GPU device can cause a significant performance loss; and
- shared memory is organised into banks and accessing elements not consecutively will cause a bank conflict.

The modern NVIDIA GPUs come with an application programming interface (API), which had to be integrated into the implemented code, for example, Matlab's MEX functions. Additionally, several other programming libraries had to be used in order to implement the time-critical operations, such as the matrix factorisation and the inverse. The code makes use of CUBLAS, which is NVIDIA's implementation of the popular BLAS library [83, 88, 89] and of the CULA premium libraries [90], which provide a CUDA-accelerated implementation of linear algebra routines from the LAPACK library [91], including the time-critical Cholesky decomposition. Additionally, several other matrix operations had to be implemented as custom CUDA kernels.

As hardware capabilities are improving constantly and research on efficient algorithms is on-going, the presented hardware solutions might not be of long-term value. However, they offer a state-of-the-art, affordable, hardware configuration that might help to circumvent the computational issue in an intermediate time frame before more efficient algorithms or better technology arrive.

### 2.5.2 Indirect Implementation

To overcome the computational limitation issues and make use of the method also for large-scale dataset applications, a lot of authors have suggested various approximations, e.g. [92–94]. However, the research in this field is still active. A unified view of approximations of GP with a comparison of the methods is provided in [95] and some newer developments are compared in [96, 97].

Section 2.5 mentions that the computational demand of the GP regression direct implementation increases with the third power of the size of training set—$\mathcal{O}(n^3)$. This is due to the calculation of the covariance matrix inverse in

$$\alpha = \mathbf{K}^{-1}\mathbf{y} \tag{2.42}$$

or better finding the solution to the linear system of equations

$$\mathbf{K}\alpha = \mathbf{y} \tag{2.43}$$

for $\alpha$. This becomes an issue when we work on problems that involve large quantities of training data, e.g. more than a couple of thousand for the presently available average computation power. Consequently, the developments of methods designed to reduce the computational demands of the GP modelling approach have received a great deal of attention.

These so-called approximation methods can, in general, be divided as follows:

- fast matrix-vector multiplication (MVM) methods, which use efficient computational methods for solving the system of linear equations. An example would be the use of iterative methods such as conjugate gradients;
- sparse matrix methods, which approximate the covariance matrix. The idea of the sparse matrix methods is to reduce the rank of the covariance matrix, i.e. the number of linearly independent rows, and to keep as much information contained in the training set as possible.

Figure 2.22 shows a schematic representation of an overview of the approximation methods.

#### *Fast Matrix-Vector Multiplication*

MVM methods treat all the identification data and build a full covariance matrix $\mathbf{K}$ of size $N \times N$, but use various, more efficient computation methods or their approximations for the calculation of computationally demanding expressions, i.e. the inverse of the covariance matrix and the logarithm of the covariance matrix determinant. An overview of the MVM methods is given in [98].

An example of such a method for solving the problem described with Eq. (2.43) is the conjugate gradient optimisation method, which reduces the computational demand of one iteration down to the order of $\mathcal{O}(N^2)$ [49], but an approximate solution can be obtained if the algorithm is terminated after $k$ iterations, which results in an overall computational demand of $\mathcal{O}(kN^2)$.

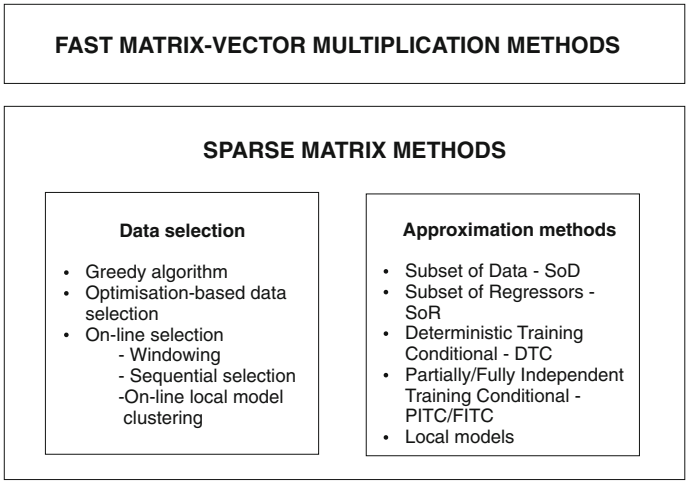


**Fig. 2.22** Schematic representation of an overview of the approximation methods for GP modelling

Some other methods that reduce the number of computations of the covariance matrix inverse and the logarithm of the covariance determinant can be found in [99–101].

The MVM methods are the most efficient when the number of data is relatively small. However, these methods are also necessary with large amounts of data in order to decrease the computer memory demand during optimisation.

### *Sparse Matrix Methods*

The fundamental property of the sparse matrix methods is that only a subset of the variables is treated exactly, with the remaining variables given some approximate, but computationally cheaper, approach. This is the most straightforward method for reducing the computational burden.

The first step in constructing a reduced rank or a sparse version of the covariance matrix $\mathbf{K}$ is the selection of a subset of datapoints. The selection of this 'active' subset of data, or active dataset, also called the induction variables, is something in common to all sparse matrix methods.

The second step is the approximation or treatment of the used latent stochastic variables that are to be treated exactly, by the GP model framework and of the remaining variables that are to be approximated by a less computationally demanding method. The used subset is of size $M \ll N$, where $N$ is the size of the overall training dataset.

Only the active dataset is used for GP modelling. The rest of the $N - M$ data is treated differently using different approximation methods.

First, we list the methods for the selection of data to be included in the active subset and, second, we list the approximation methods. However, it needs to be emphasised

that a number of other sparse matrix methods have appeared in recent years as this is a field of active research.

**Methods for Data Selection**

As only the active dataset is to be treated fully in the sparse model, the process of determining which data points are to be included is critical to the success of the approximation [50]. One possible strategy is to build the subset of data through the manual selection of data points based on a preliminary knowledge of the system to be modelled. However, this is difficult where the preliminary knowledge about the system to be modelled is limited and when the system contains complex, multidimensional nonlinearities, which is a common situation with systems identification.

It makes sense to search the active dataset based on a selected criterion. The criterion-based evaluation of all the possible combinations of an active dataset of dimension $M$ is not viable, because of the $\frac{N!}{M!(N-M)!}$ combinations. There have been a lot of methods developed based on different criteria to overcome the computational issue.

**Greedy algorithm** is one of the more popular methods. Apart from the random selection of data points, with limited applicability, *greedy approximation* methods [102] have been shown to have great potential. The idea of these methods is that the active dataset is selected and updated according to some criterion. Such an algorithm would initiate with an empty active set $\mathcal{I}$, with the remaining set $\mathcal{R}$ containing all the indexed training observations. Then, using an iterative method, each indexed training example is added to the active set in turn and the selection criterion is evaluated. If the criterion is met, the training example under review will be included in the active set.

Various authors, e.g. [102–104], have suggested different criteria for the selection of data.

This method can be used with most of the approximation methods. The question that arises is what kind of selection criteria should be used to determine the active subset of data. Some of the selection criteria are those used in the following methods: informative vector machine [102], informative gain [105], sparse spectral sampling [106], iterative sparse [103] and matching pursuit [104].

**Optimisation-based data selection** Greedy methods select the active dataset out of the available identification data. A method is proposed in [33] called *Sparse Pseudo-input Gaussian Processes (SPGP)*, which does not select the dataset $\mathcal{I}$, but optimises it. This means that the active dataset $\mathcal{I}$ can at the end of the optimisation contain arbitrary data. Optimisation of this arbitrary data, called *pseudo-inputs*, is pursued simultaneously with the optimisation of the hyperparameters based on the log-marginal likelihood loss function. This is perceived as an advantage of this method.

Based on the same idea, the authors of [107] have proposed a new sparse matrix method called *Sparse Spectrum Gaussian Process Regression (SSGP)*, which is based on a spectral representation of GP.

Because of the increased number of optimisation parameters, both methods are susceptible to overfitting, particularly when there is no available information on the initial values of the hyperparameters and the pseudo-inputs. To overcome this issue, a variational method is proposed in [108]. This method maximises the lower boundary of the log-marginal likelihood instead of the exact value when it optimises the active dataset $\mathcal{I}$ and the hyperparameters.

**Online data selection** In the case that the GP model is identified based on the incoming data stream that provides new information about the modelled system online, online identification, or training, methods are in place.
Online data selection can be roughly divided as follows.

*Windowing* is a method where the most recent $m$ data is used for the identification. The active dataset $\mathcal{I}$ is therefore composed of the most recent $m$ data, which are all weighted equally for the optimisation. The windowing method is called also the time-stamp method. Examples of using windowing in the GP modelling context can be found in [109–111].
The situation is different with the method of *forgetting*, where the significance of the data is decreasing, usually exponentially, with age. The forgetting method is also called the weight-decay method [110]. When the data point significance is below the threshold, the data point is discarded. The forgetting method is still closely related to the pure windowing.

*Sequential selection* follows the idea that the size of the active dataset $\mathcal{I}$ has to be constrained, which is implemented with a selected criterion for the inclusion of data. This criterion is similar to that used by the greedy method, but in this case each data point is evaluated separately with already-selected data in the active dataset $\mathcal{I}$ and not with those in the remaining dataset $\mathcal{R}$, as is the case with the greedy method. The criterion that has been proposed in [103], on the other hand, is based on a change of the mean value of the posterior distribution when a new data point is included in the model. Following this idea, the authors of [112] proposed the method for online training, which also enables the online optimisation of the hyperparameters.

*Online model clustering* Slightly different from the other methods for online data selection is the method described in [113]. This method is based on the online clustering of data, where these clusters of data are used for the identification of local GP models. The advantage of this method is its computational speed, because online clustering is computationally much less demanding than online GP model training. Nevertheless, the quality of the prediction could be slightly worse in comparison with using data selection methods.

### Methods for Approximation

The idea behind sparse matrix methods is to change the joint prior distribution of the identification and validation data $p(f^*; f)$ from Eq. (2.44) so that the computation of the predictive distribution based on the validation data from Eq. (2.45) is less time consuming.

$$p(f^*; f) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k}(\mathbf{z}^*) \\ \mathbf{k}^{\mathrm{T}}(\mathbf{z}^*) & \kappa(\mathbf{z}^*) \end{bmatrix}\right), \tag{2.44}$$

where $^*$ denotes the validation data.

$$p(f^*|y) = \mathcal{N}\left(\mathbf{k}(\mathbf{z}^*)\mathbf{K}^{-1}\mathbf{y}, \kappa(\mathbf{z}^*) - \mathbf{k}^{\mathrm{T}}(\mathbf{z}^*)\mathbf{K}^{-1}\mathbf{k}(\mathbf{z}^*)\right) \tag{2.45}$$

We divide the approximation methods into five groups.

**Subset of data** (SoD) method appears to be the most straightforward method for a sparse matrix approximation, where the active set of the data $M$ is selected from the whole training data set of size $N$. The equations for the calculation of the predictions and the optimisation remain unchanged by the method. The calculation method for such a method depends only on the dataset of size $M$ and is therefore $\mathcal{O}(M^3)$, where $M < N$. The efficiency of this method depends very much on the selection of the subset of data, while the remaining dataset is discarded rather than approximated. Nevertheless, with carefully selected data the method may still provide a good approximation to the GP model with complete dataset and is consequently frequently used in comparison with the more sophisticated sparse methods.

**Subset of regressors** (SoR) method takes advantage of the equivalence between the GP model's mean predictor and that of a finite-dimensional, generalised, linear regression model. Consequently, the SoR model is a finite, linear-in-the-parameters model with a particular prior knowledge put on the weights. In contrast to the SoD method, the SoR method is to employ all $N$ datapoints of the training set in the approximation. The major disadvantage of the SoR method is that, due to its basis upon a linear-in-the-parameters model, the GP model becomes degenerate and restricts the variety of possible functions that will be plausible under the posterior [96]. The main disadvantage of this degeneracy is that the predictive distributions of the GP model can become unreasonable. The computation demand of the SoR method is $\mathcal{O}(M^2N)$ for the initial matrix computations, and $\mathcal{O}(M)$ and $\mathcal{O}(M^2)$ for the calculation of the predictive mean and variance, respectively.

**Deterministic training conditional** (DTC) method applies the model of the entire $N$ size dataset and therefore does not use a degenerated model like the SoR method does. The obtained GP model represents only $M < N$ function values, but uses the entire dataset so that it projects $M$ input values to $N$ dimensions. The computational method for the DTC method is, like in the case of SoR method, $\mathcal{O}(M^2N)$, for the initial matrix computations, and $\mathcal{O}(M)$ and $\mathcal{O}(M^2)$ for the calculation of the predictive mean and variance, respectively. The method was initially named projected latent variables [105], but is also known as projected processes [49]. The predictive mean value when using the DTC method is identical to that of using the SoR method. The predictive variance is never less than the variance when using the SoR method. The computation demand of the DTC method is the same as that of the SoR method.

**Partially and fully independent training conditional** (PITC, FITC) The previously described methods use exclusively identification data for the input–output mapping function modelling, so the covariance matrix for the training conditional distribution $p(f|\mathbf{u})$ is zero. Another possibility is that the covariance matrix has

a block-diagonal form, with data obtained from the covariance matrix of the complete dataset. This means that we assume conditional independence, which is assumed in previously described methods for only a part of the modelled function values. Such a method is called *Partially Independent Training Conditional (PITC)*.

The PITC method's computational demand depends on the used block-diagonal form. In the case of $l = \frac{N}{M}$ blocks of size $M \times M$, as in [114] the computational demand is $\mathcal{O}(NM^2)$.

A special form of the PITC method is the approximation method called *Fully Independent Training Conditional (FITC)*, which was proposed in [33], where it was named the SPGP method. The name FITC comes from the fact that the training set function observations are presumed to be completely independent. The FITC method is almost identical to the PITC method, except for the fact that the covariances on a diagonal of the covariance matrix are exact. This means that instead of approximated prior variances the exact prior variances are on the covariance matrix diagonal.

The predictive distribution obtained with the FITC method is identical to the predictive distribution obtained with the PITC method. The computational demand is equal to those of the SoR, DTC and PITC methods.

**Local models** An alternative approach to the listed methods for the approximation of the covariance matrix or the model likelihood is the method that replaces one GP model with *local models*. The obtained model is known as a *mixture of GP models* or *mixture of GP experts*. This method does not match exactly to the listed ones, because the local models replace the entire GP model. Some research about using local models for an approximation in the context of dynamic systems modelling can be found in [113, 115, 116].

The idea of the method is based on the *divide-and-conquer* principle. The method divides the dataset of $N$ identification data into $l = \frac{N}{M}$ subsets or *clusters* of size $M$. The data in these clusters is used as the identification data for separate, locally valid GP models. The various local-model-based methods differ, among others, in the way in which the various existing clustering methods are pursued.

The hyperparameters can be optimised for each of the local models separately or for all the local GP models together. The latter is an interesting option only when $M$ is small enough. Combination of posterior distributions of local models into one distribution that represents the model prediction is another difference between the various methods.

All the listed approximation methods have comparable computational demands. The overview in [96] shows that the obtained predictive mean values are very similar, while the predictive variances are quite different. Therefore, it is sensible to decide

before selecting an approximation method, whether the information about the exact model's predictive variance is important, or we are focused more on the model's predictive mean value.

The SoR method is appropriate in the case that the predictive variance is not very important, because the predictive variance approximation is computationally the simplest one.

In the case, however, that the accurate information about the predictive variance is important, it is sensible to use one of the more advanced approximation methods, i.e. DTC, PITC or FITC.

In the case of computationally simpler identification problems, which are those with the smaller number of identification data, it is sensible to use the SoD method. It competes well with the more advanced approximation methods.

Another aspect interesting for the approximation-method selection is the implementation of the method, the computational demand for the model identification and the demand necessary for the obtained model prediction.

It is very much worth considering a combination of methods, like the one proposed in [97]. This approach combines the SoD method for the model training and the FITC method for the model prediction. It uses the selected active dataset $\mathcal{I}$ and the hyperparameters obtained with the SoD method to save computational time for the hyperparameters' optimisation, which for the SoD method is $\mathcal{O}(M^3)$, instead of $\mathcal{O}(NM^2)$ for the FITC method.

### 2.5.3 Evolving GP Models

In this section, an approach to the online training of GP models is described. Such an approach is needed when the dynamic system to be identified is represented as a time-varying one or when the training data is not available for the whole region of interest, and so not all the dynamics of the system can be trained at once. In these cases, the model needs to be constantly adapted in accordance with the new operating region and/or the changing dynamics. Such an approach can be used in environments that are constantly changing. For this purpose, a method for the online adaptation of GP models is proposed in [117], and the models obtained with this method are named *Evolving GP models*.

Evolving systems [118] are self-developing systems inspired by the idea of system model evolution in a dynamically changing and evolving environment. They differ from other traditional adaptive systems known from control theory [119, 120] in that they online adapt both the structure and parameter values of the model using the incoming data [118].

The term evolving is used in the sense of the iterative adaptation of the structure of the GP model and the hyperparameter values. This term was introduced in the

1990s for neural networks [121], in 2001 for fuzzy rule-based systems [122] and in 2002 for neuro-fuzzy systems [123].

The GP models depend on the data and the covariance function. More precisely, the data is defined with various regressor variables, in short regressors, and corresponding observations grouped in the so-called basis vectors. The covariance function is defined with the type and hyperparameter values.

Therefore, there are at least four parts that can evolve

- the regressors,
- the basis vectors,
- the type of covariance function and
- the hyperparameter values.

The ideas of the online adaptation of GP models can be found in literature implemented mainly as online data selection, e.g. [111, 124], online data selection and hyperparameters determination, e.g. [125], and active learning for control, e.g. [117, 126]. A method that does not select an active dataset for GP model identification, but only iteratively corrects the identified model's predictive mean and variance with new data, is proposed in, e.g. [127, 128]. The criterion for the correction is based on the prediction error.

As already stated in Sect. 2.5, the training of GP models for a large amount of data is very time consuming. The condition under which the evolving GP model does not 'decay in time' with in-streaming data is the so-called *persistent excitation* condition, which means that the in-streaming signal has enough information content. This can be achieved so that only a subset of the most informative data, the so-called **basis vectors** set, is used. With a type or a combination of various types of covariance functions a prior knowledge of the system is included in the model. Nevertheless, by optimising **the hyperparameter values** the model response evolves closer to the response of the real system. However, in dynamic, nonlinear systems, where the nonlinear mapping between the input and output data cannot be easily formulated, the squared exponential covariance function is frequently used, assuming the smoothness and stationarity of the system. This means that the covariance function may be kept fixed and does not need to evolve. Nevertheless, the structure-discovery algorithm described in [129] can be used for the online **covariance function** selection. Furthermore, the squared exponential covariance function can be used with ARD, which is able to find the influential **regressors** [49]. With the optimisation of the hyperparameter values, the noninfluential regressors have smaller values and, as a consequence, they have a smaller influence on the result. Therefore, all the available regressors can be used and, as a result, only the set of basis vectors and the hyperparameter values remain to be evolved.

The general concept of evolving GP models, presented in [117], with a fixed covariance function and regression vector, contains the following steps:

**Algorithm:** MODELEVOLVING($\mathbf{z}^*, y^*$)

**comment:** Add new input data $\mathbf{z}^*$ to the informative dataset $\mathcal{I}$

1. $\mathcal{I} \leftarrow$ ADD($\mathcal{I}, \mathbf{z}^*$)

**if** LENGTH($\mathcal{I}$) $>$ *maxLength*

**then**
- **comment:** Calculate the information gain for each data point in $\mathcal{I}$
- 2. *iGains* $\leftarrow$ INFORMATIONGAIN($\mathcal{I}$)
- **comment:** Remove worst data from $\mathcal{I}$
- 3. $\mathcal{I} \leftarrow$ REMOVEWORST($\mathcal{I}$, *iGains*)

**comment:** Calculate hyperparameter values $\boldsymbol{\theta}$

4. $\boldsymbol{\theta} \leftarrow$ CALCHYPVAL($\mathcal{I}, \boldsymbol{\theta}$)

**comment:** Update covariance matrix

5. $\mathbf{K} \leftarrow$ UPDATECOVMAT($\mathcal{I}, \boldsymbol{\theta}$)

These basic steps are repeated for every incoming sample of data until there is no more available data or until a requirement to stop the process is received.

To keep the subset of the most informative data small enough to process all the steps before new data arrives, the maximum length of the subset should be set with the parameter *maxLength*. This means that the parameter *maxLength* is a design parameter.

Operations in the pseudocode can be implemented in various ways. There are two critical operations: the calculation of the information gain and the calculation of the hyperparameter values. Both of these operations can be implemented using various well-known methods. For the information gain calculation, any data selection criterion from online learning methods for GP models can be used, e.g. [103, 105], etc. Also, for the hyperparameter value calculation any suitable optimisation method or even some heuristic method can be used. Our implementation of the operations in the concept is described below.

First, the basic elements and some operations will be described. The core of the concept is a set of the most informative data. Actually, it is a subset of the data that was already considered. It is denoted as $\mathcal{I}$ and defined as

$$\mathcal{I} \subset \mathcal{D}. \tag{2.46}$$

To operate with the set $\mathcal{I}$ two basic operations are needed: adding elements and removing elements. Both operations are very straightforward. Adding the new element $\zeta^+$ to the set $\mathcal{I}$ is defined as

$$\mathcal{I}^+ = \left\{\mathcal{I}, \zeta^+\right\}, \tag{2.47}$$

where $\mathcal{I}^+$ is a new, extended set of the most informative data. Removing the $i$th element $\zeta_i$ from the set $\mathcal{I}$ is defined as

$$\mathcal{I}^{-} = \{\zeta_1, \ldots, \zeta_{i-1}, \zeta_{i+1}, \ldots, \zeta_M\}, \tag{2.48}$$

where $M$ is the number of elements in the set $\mathcal{I}$.

The main operations of the algorithm are implemented as follows:

1. ADD($\mathcal{I}, \mathbf{z}^*$): Adds new data to the set of the most informative data $\mathcal{I}$. It is implemented in such a way that it adds new data $\mathbf{z}^*$ to $\mathcal{I}$ only when the data contributes new information to the current model. This improves the robustness of the algorithm. The contribution of the data, according to the current model, is scored on the prediction distribution for $\mathbf{z}^*$. If the absolute difference between the prediction mean value $\mu(\mathbf{z}^*)$ and the measured value $y(\mathbf{z}^*)$ is greater than the pre-set threshold, this means the current model cannot predict the prediction based on $\mathbf{z}^*$ accurately enough. Therefore, $\mathbf{z}^*$ should be added to $\mathcal{I}$.
If the absolute difference $|y(\mathbf{z}^*) - \mu(\mathbf{z}^*)|$ is small enough, the prediction variance is also taken into consideration. If the prediction variance $\sigma^2(\mathbf{z}^*)$ is smaller than the pre-set threshold, it can be considered that the model is also confident enough in its prediction; therefore, there is no need to include the new data. If the prediction variance is high, the model is not confident in the prediction. This means that the model does not have enough information in that region; therefore, $\mathbf{z}^*$ should be added to $\mathcal{I}$.
To summarise, $\mathbf{z}^*$ is added only when the absolute difference between the prediction mean value $\mu(\mathbf{z}^*)$ and the measured value $y(\mathbf{z}^*)$ or the prediction variance $\sigma^2(\mathbf{z}^*)$ is greater than the pre-set thresholds for the mean value and variance, respectively.
Thresholds can be set heuristically and are, therefore, design parameters. If the condition is fulfilled, $\mathbf{z}^*$ is added to $\mathcal{I}$ using Eq. (2.47). This operation as such might not be necessary, but it avoids any unnecessary updates of $\mathcal{I}$, which improves the computational efficiency of the algorithm.

---

**Procedure 1:** ADD($\mathcal{I}, \mathbf{z}^*, y(\mathbf{z}^*)$)

---

**comment:** Prediction of the GP model is calculated

**if** $|y(\mathbf{z}^*) - \mu(\mathbf{z}^*)| >$ threshold$_\mu$ AND $\sigma^2(\mathbf{z}^*) >$ threshold$_{\sigma^2}$

**then** {
**comment:** Adding $\mathbf{z}^*$ to $\mathcal{I}$ using Eq. (2.47)
$\mathcal{I} \leftarrow \{\mathcal{I}, \mathbf{z}^*\}$

**return** ($\mathcal{I}$)

---

2. INFORMATIONGAIN($\mathcal{I}$): Calculates the information gain for each element in $\mathcal{I}$. Actually, it calculates the log-marginal likelihood for each subset $\mathcal{I}^{-}$ of size $M - 1$, where $M$ is the number of elements in $\mathcal{I}$. It should be noted that this operation is performed only when the subset $\mathcal{I}$ has exceeded the pre-set size $L$. A higher log-marginal likelihood means a higher information gain for the processed element

in $\mathcal{I}$. As calculated for the subset without the data for which the information gain is to be calculated, it should be inverted. Therefore, it is multiplied by $-1$, so that a higher log-marginal likelihood means a lower information gain.
This principle is in its essence the Bayesian information criteria (BIC) [44]:

$$\begin{aligned} \text{BIC} &= -2\ln p(\mathcal{I}^-|\boldsymbol{\theta}) + (D+2)\ln(M) \\ &= -2\tilde{\ell}(\mathcal{I}^-, \boldsymbol{\theta}) + (D+2)\ln(M), \end{aligned} \tag{2.49}$$

where $D+2$ is the number of hyperparameters to be estimated.
In the case of a time series, forgetting is also used. It is implemented as exponential forgetting

$$\tilde{\ell}(\mathcal{I}^-, \boldsymbol{\theta}) = \lambda^{i-c} \cdot \ell(\mathcal{I}^-, \boldsymbol{\theta}) \tag{2.50}$$

where $\lambda \in [0, 1]$ is the forgetting factor, $i$ is the current sequence number, $c$ is the number of the sequence when the currently considered data was added to $\mathcal{I}$, $\tilde{\ell}$ is the log-marginal likelihood considering the exponential forgetting and $\mathcal{I}^-$ is the subset of $\mathcal{I}$ of size $M-1$. The forgetting can be easily turned off by setting $\lambda = 1$.

**Procedure 2:** INFORMATIONGAIN($\mathcal{I}$)

**for** $i \leftarrow 1$ **to** LENGTH($\mathcal{I}$)
**do**
- **comment:** removing $i$th element from $\mathcal{I}$ using Eq. (2.48)
- $\mathcal{I}^- \leftarrow \{\zeta_1, \ldots, \zeta_{i-1}, \zeta_{i+1}, \ldots, \zeta_M\}$
- **comment:** calculating information gain using Eq. (2.50)
- $iGains[i] \leftarrow -\tilde{\ell}(\mathcal{I}^-, \boldsymbol{\theta})$

**return** ($iGains$)

3. REMOVEWORST($\mathcal{I}$, $iGains$): Removes element with the worst information gain from the set $\mathcal{I}$. It is performed simply using Eq. (2.48).

**Procedure 3:** REMOVEWORST($\mathcal{I}$, $iGains$)

$ind \leftarrow -1$
$min \leftarrow \infty$
**for** $i \leftarrow 1$ **to** LENGTH($iGains$)
**do**
- **if** $iGains[i] <$ min
  - **then**
    - $ind \leftarrow i$
    - $min \leftarrow iGains[i]$

**comment:** removing $ind$th element from $\mathcal{I}$ using Eq. (2.48)
$\mathcal{I} \leftarrow \{\zeta_1, \ldots, \zeta_{ind-1}, \zeta_{ind+1}, \ldots, \zeta_M\}$
**return** ($\mathcal{I}$)

4. CALCHYPVAL($\mathcal{I}, \boldsymbol{\theta}$): Hyperparameter values are calculated by maximising the marginal log likelihood. This can be done with any suitable optimisation method off-line or with iterative calculations online.
In the off-line case, the evolving method downgrades to the so-called online modelling with a fixed set of input data, examples can be found in [7, 76].
5. UPDATECOVMAT($\mathcal{I}, \boldsymbol{\theta}$): Updates the covariance matrix when $\mathcal{I}$ or $\boldsymbol{\theta}$ have changed.
Updates the covariance matrix and its inversion. If the hyperparameter values have changed, both the covariance matrix $\mathbf{K}$ and its inversion $\mathbf{K}^{-1}$ must be fully recalculated. However, in cases when only $\mathcal{I}$ has changed, the covariance matrix and its inversion can be updated more efficiently. The covariance matrix is updated by appending $\mathbf{k}(\mathbf{z}^*)$ and $\mathbf{k}^{\mathrm{T}}(\mathbf{z}^*)$ as presented in Eq. (2.51) and removing the $i$th row and column if the $i$th data was removed from $\mathcal{I}$, as shown in Eq. (2.52).

$$\mathbf{K}^{+} = \left[\begin{array}{ccc|c} \mathbf{K}_{1,1} & \cdots & \mathbf{K}_{1,M} & \mathbf{k}_1 \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{K}_{M,1} & \cdots & \mathbf{K}_{M,M} & \mathbf{k}_M \\ \hline \mathbf{k}_1 & \cdots & \mathbf{k}_M & \kappa \end{array}\right], \tag{2.51}$$

where $\mathbf{k}$ is the vector of covariances between the data in the active dataset $\mathcal{I}$ and the new data $\mathbf{z}^+$ and $\kappa$ is the autocovariance of $\mathbf{z}^+$.

$$\mathbf{K}^{-} = \left[\begin{array}{ccc|ccc} \mathbf{K}_{1,1} & \cdots & \mathbf{K}_{1,i-1} & \mathbf{K}_{1,i+1} & \cdots & \mathbf{K}_{1,M} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{K}_{i-1,1} & \cdots & \mathbf{K}_{i-1,i-1} & \mathbf{K}_{i-1,i+1} & \cdots & \mathbf{K}_{i-1,M} \\ \hline \mathbf{K}_{1+1,1} & \cdots & \mathbf{K}_{i+1,i-1} & \mathbf{K}_{i+1,i+1} & \cdots & \mathbf{K}_{i+1,M} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{K}_{M,1} & \cdots & \mathbf{K}_{M,i-1} & \mathbf{K}_{M,i+1} & \cdots & \mathbf{K}_{M,M} \end{array}\right] \tag{2.52}$$

As the Cholesky decomposition is used for the calculation of the covariance matrix inversion, it can be updated in a similar way as updating the covariance matrix using the rank-1 update and downdate operations [130]. First, the Cholesky decomposition is updated by $\mathbf{k}(\mathbf{z}^*)$ and later downdated by $\mathbf{k}(\mathbf{z}_i)$ if the $i$th data was removed from $\mathcal{I}$, which is relatively efficient ($\mathcal{O}(M^2)$).

## 2.6 Validation

Validation concerns the level of agreement between the mathematical model and the system under investigation [131] and it is many times underemphasised despite its importance. The model validation in the context of identification is the phase following the model selection, where the regressors, covariance function, mean function and

hyperparameters are selected. Validation is eliminated in a full Bayesian approach where these elements are not selected by optimisation but integrated out. Nevertheless, the GP model selection, which is based on evidence or marginal likelihood maximisation, requires this phase too. In validation, we are concerned with evaluating the performance of the model based on datasets different from those used for the modelling.

The data that is used for modelling is called *identification data*, also *estimation data*, which are names common to identification literature. An identification dataset can be, depending on the model used, split into the subset of data used for the parameters' estimation, and the subset of data for the structure identification as well as the model order and regressors. The purpose of this division is to use the other subset of data for monitoring the performance during the identification for model structure reduction. The data for evaluating the performance, which is different from that used for the identification, is called *validation data*.

This division of data subsets is referred to differently in machine-learning literature, though the purpose is the same. The data for parameter estimation is called *training data*, the data for monitoring the performance is called *validation data* and the data used for the validation of the obtained final model is called *test data*. It has to be noted that this division can also be met in the literature describing system identification, e.g. [10].

The concept of splitting empirical data into separate sets for identification and validation is generally known as *cross-validation*. An analysis of importance to use separate sets for validation also in the context of GP classification is done in [132], but can be generalised also for regression.

The quality of the model that is in the focus of the validation can be represented by several features. Their overview can be found in [131, 133]. The most important are the model plausibility, model falseness and model purposiveness, explained as follows.

Model *plausibility* expresses the model's conformity with the prior process knowledge by answering two questions: whether the model 'looks logical' and whether the model 'behaves logical'. The first question addresses the model structure, which in the case of GP models means mainly the plausibility of the hyperparameters. The second one is concerned with the responses at the model's output to typical events on the input, which can be validated with a visual inspection of the responses, as is the case with other black-box models.

Model *falseness* reflects the agreement between the process and the model's output or the process input and the output of the inverse model. The comparison can be made in two ways, both applicable to GP models: qualitatively, i.e. by visual inspection of the differences in the responses between the model and the process, or quantitatively, i.e. through the evaluation of the performance measures, some of them listed later in the section.

Model *purposiveness* or usefulness tells us whether or not the model satisfies its purpose, which means the model is validated when the problem that motivated the modelling exercise can be solved using the obtained model. Here, again, the

prediction variance can be used, e.g. when the prediction confidence is too low, the model can be labelled as not purposive.

As already mentioned in Sect. 2.1, various purposes are served with two sorts of models: for prediction, i.e. one-step-ahead prediction, and for simulation. Section 2.3 described different model structures in this context. Nevertheless, it is a very common practice to train the model for prediction and to call out its purpose during the validation, where especially the dynamic system model is often validated for simulation.

The cross-validation concept requires a relatively large amount of data. This is provided by properly designed experiments, which are, commonly, repeated if the amount of data does not correspond to the modelling needs. That is why the experiment design and the experiments themselves are very important parts of system identification.

However, for cases where the amount of empirical data is limited and new experiments cannot be pursued, methods have been suggested that seek to maximise the exploitation of the available data. A lot of research to solve this problem has been done in machine learning. Here, we highlight only some more frequently used methods for the more efficient use of the identification data.

One such method is *k-fold cross-validation* where the available empirical data is partitioned into $k$ data subsets. Each subset is then used in turn as a dataset for the evaluation of the model trained on the other $k - 1$ data subsets. The overall error rate is taken as the average of these $k$ data subset evaluations.

The extreme version of the k-fold cross-validation is when only a single observation, data piece, of the overall data is to be left out and used as an evaluation example. The remaining data is used for training. The method is called *leave-one-out-validation* (LOO). It is a method that can be used for small datasets.

While GP model validation in the context of one-step-ahead prediction is elaborated already in [49], where marginal likelihood is discussed in detail, cross-validation, especially leave-one-out cross-validation (LOO-CV), is described. The expression 'validation' in the listed terms is used here in the machine-learning sense and not in the system identification sense, where the validation data is a separate and fresh dataset.

We have mentioned that model falseness can be evaluated with different performance measures. There exists an abundance of performance measures, each suited for a different purpose with the emphasis on a different property and with different expressiveness. Here we list only a few that serve our purposes.

A commonly used performance measure, especially when the model is identified using a method that is based on a square error, is the mean-squared error (MSE):

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - E(\hat{y}_i))^2, \tag{2.53}$$

where $y_i$ and $\hat{y}_i$ are the system's output measurement, i.e. observation and model's output in $i$th step, respectively. The model's output can be, based on investigation, a posterior probability distribution of a prediction or a simulation.

The measure that normalises the mean-squared error between the mean of the model's output and the measured output of the process by the variance of the output values of the validation dataset is the standardised mean-squared error (SMSE):

$$\text{SMSE} = \frac{1}{N}\frac{\sum_{i=1}^{N}(y_i - E(\hat{y}_i))^2}{\sigma_y^2}, \tag{2.54}$$

where $y_i$ and $\hat{y}_i$ are the system's output measurement, i.e. observation and the model's output in the $i$th step.

The mean relative square error (MRSE) is calculated by taking the square root of the measure MSE divided by the average of output measurements:

$$\text{MRSE} = \sqrt{\frac{\sum_{i=1}^{N}(y_i - E(\hat{y}_i))^2}{\sum_{i=1}^{N} y_i^2}}. \tag{2.55}$$

Some authors call this performance measure the relative-root-mean-square error (RRMSE).

The performance measures described with Eqs. (2.53)–(2.55) deal with the mean values of outputs and do not take the entire output distribution into account.

The performance measures such as the log predictive density error (LPD) [134, 135] can be used for evaluating GP models, taking into account not only the mean of the model prediction, but also the entire distribution:

$$\text{LPD} = \frac{1}{2}\ln(2\pi) + \frac{1}{2N}\sum_{i=1}^{N}\left(\ln(\sigma_i^2) + \frac{(y_i - E(\hat{y}_i))^2}{\sigma_i^2}\right) \tag{2.56}$$

where $\sigma_i^2$ is the model's output variance in the $i$th step. The performance measure LPD weights the output error $E(\hat{y}_i) - y_i$ more heavily when it is accompanied by a smaller output variance $\sigma_i^2$, thus penalising the overconfident model's output values more than the acknowledged bad model's output values, indicated by a higher variance.

The mean standardised log loss (MSLL) [49] is obtained by subtracting the loss of the model that predicts using a Gaussian with the mean $E(\mathbf{y})$ and the variance $\sigma_y^2$ of the measured data from the model LPD and taking the mean of the obtained result

$$\text{MSLL} = \frac{1}{2N}\sum_{i=1}^{N}\left[\ln(\sigma_i^2) + \frac{(y_i - E(\hat{y}_i))^2}{\sigma_i^2}\right] - \frac{1}{2N}\sum_{i=1}^{N}\left[\ln(\sigma_y^2) + \frac{(y_i - E(\mathbf{y}))^2}{\sigma_y^2}\right]. \tag{2.57}$$

The MSLL is approximately zero for simple models and negative for better ones.

The smaller the listed measures are, the better the model response is, irrespective of whether it is a prediction or simulation.

The variance of the GP model predictions on a validation signal can be a plausibility measure itself, as it indicates whether the model operates in the region where the identification data were available. Nevertheless, it should be used carefully and in combinations with other validation tools, as predictions with a small variance are not necessarily good, as will be shown in the example at the end of this chapter.

To avoid the computational complexity of cross-validation, alternative methods of evaluating validation errors have been developed. These include the use of various *information criteria* methods, such as the final prediction error information criterion, or Akaike's information criterion [44], where the normalised log likelihood is used as the prediction error criterion

$$\text{AIC} = -2\ln(p(\mathcal{D}|\boldsymbol{\theta}_{\text{ML}})) + 2n, \tag{2.58}$$

where $n$ is a number of adjustable parameters in the model and $\boldsymbol{\theta}_{\text{ML}}$ is the maximum likelihood solution for $\boldsymbol{\theta}$, or Akaike's Bayesian information criterion [44] that uses the marginal likelihood of the observed data $\mathcal{D}$ given the model:

$$\text{BIC} = -2\ln(p(\mathcal{D}|\boldsymbol{\theta}_{\text{MAP}})) + n\ln N, \tag{2.59}$$

where $N$ is the number of data and $\boldsymbol{\theta}_{\text{MAP}}$ is the value of $\boldsymbol{\theta}$ at the mode of the posterior distribution. Statistical hypothesis tests can also be used for the model validation. See [2] or [10] for more information on these validation strategies.

The quality of the obtained model can also be evaluated based on a residual analysis. Residuals are the differences between the most likely model's output values and the measured output values of the system to be modelled. Residual analysis [2] is evaluating statistics of residuals like correlation tests, whiteness tests and analyses of average generalisation errors. It is often used with methods that are concern mainly with the model's posterior mean values.

The authors of [12] discuss methods for the validation of prediction models in the context of neural networks with the residual analysis. They also provide a discussion on the visualisation of predictions, which is also a useful method with the validation of simulation models.

Criteria that are concerned mainly with the model's posterior mean values do not take account the entire posterior distributions as in a fully Bayesian approach [44, 49], which is explained in Sects. 1.1 and 2.4.1.

Since a more than fair portion of dynamic system models is validated for simulation purposes, the next section is devoted to the implementation of a model simulation in the context of GP models.

## 2.7 Dynamic Model Simulation

The simulation of dynamic system models can be used for the evaluation of the model behaviour or for the model validation. Simulation is a multistep-ahead prediction when the number of steps in the prediction horizon is infinite or at least as large as the time horizon of interest for the foreseen analysis of the model's behaviour.

There are two implementation options for the simulation or multistep-ahead prediction:

- a direct method, where different models are learnt for every perceived horizon $h$ or
- an iterative method, where the one-step-ahead prediction is iteratively repeated.

The problem of the direct method is that the horizon needs to be known and fixed in advance. In the case that the horizon is changed, the model, or better models, has to be learnt again. The second issue with the direct method is that highly nonlinear systems need a large horizon and, consequently, a large amount of learning data [135]. An example of using the direct method for multistep-ahead prediction is given in [66].

The iterative method for Gaussian process models of dynamic systems means that the current output estimate depends on previous model estimations and on the measured inputs.

$$\hat{y}(k) = f(\hat{y}(k-1), \hat{y}(k-2), \ldots, \hat{y}(k-n), u(k-1), u(k-2), \ldots, u(k-m)), \tag{2.60}$$

where the regression vector is composed of the previous model estimations $\hat{y}$ and measured input values $u$ up to a given lag. The model is therefore treated as a model with a NOE structure.

When only the mean values of the model predicted values are fed back, the simulation is named *naive*. However, when we want to obtain a more realistic picture of the dynamic model multistep-ahead prediction, we have to take into account the uncertainty of future predictions, which provide the 'input data' for estimating further means and uncertainties. A partial overview of the results given in [136] is given as follows.

In the case of a multistep-ahead prediction, we wish to make a prediction at $\mathbf{z}^*$, but this time the input vector $\mathbf{z}^*$ contains uncertain input values fed back from the outputs. Within a Gaussian approximation, the input values can be described by the normal distribution $\mathbf{z}^* \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{z}^*}, \Sigma_{\mathbf{z}^*})$, where $\boldsymbol{\mu}_{\mathbf{z}^*}$ and $\boldsymbol{\Sigma}_{\mathbf{z}^*}$ are the vector and the matrix of the input mean values and variances, respectively. To obtain a prediction, we need to integrate the predictive distribution $p(y^*|\mathbf{z}^*, \mathcal{D})$ over the input data distribution, that is

$$p(y^*|\boldsymbol{\mu}_{\mathbf{z}^*}, \boldsymbol{\Sigma}_{\mathbf{z}^*}, \mathcal{D}) = \int p(y^*|\mathbf{z}^*, \mathcal{D}) p(\mathbf{z}^*) d\mathbf{z}^*, \tag{2.61}$$

where

$$p(y^*|\mathbf{z}^*, \mathcal{D}) = \frac{1}{\sqrt{2\pi\sigma^2(\mathbf{z}^*)}} \exp\left[-\frac{(y^* - \mu(\mathbf{z}^*))^2}{\sigma^2(\mathbf{z}^*)}\right]. \tag{2.62}$$

Since $p(y^*|\mathbf{z}^*, \mathcal{D})$ is in general a nonlinear function of $\mathbf{z}^*$, the new predictive distribution $p(y^*|(\boldsymbol{\mu}_{y^*}, \boldsymbol{\Sigma}_{y^*}, \mathcal{D}))$ is not Gaussian and this integral cannot be solved without using an approximation. In other words, when the Gaussian distribution is propagated through a nonlinear model, it is not a Gaussian distribution at the output of the model.

Approximations can be roughly divided into numerical methods, for example Monte Carlo methods, and analytical methods.

### 2.7.1 Numerical Approximation

Eq. (2.61) can be solved by performing a numerical approximation of the integral, using a simple Monte Carlo approach:

$$p(y^*|\boldsymbol{\mu}_{\mathbf{z}^*}, \boldsymbol{\Sigma}_{\mathbf{z}^*}, \mathcal{D}) \approx \frac{1}{S} \sum_{i=1}^{S} p(y^*|\mathbf{z}^{*i}, \mathcal{D}) \tag{2.63}$$

where $S$ is a number of samples and $\mathbf{z}^{*i}$ is a sample from the input data distribution $p(\mathbf{z}^*)$. The output distribution is therefore not a Gaussian, but can be seen as a Gaussian mixture:

$$p(y^*|\boldsymbol{\mu}_{\mathbf{z}^*}, \boldsymbol{\Sigma}_{\mathbf{z}^*}, \mathcal{D}) \approx \frac{1}{S} \sum_{i=1}^{S} \mathcal{N}(\mu(\mathbf{z}^{*i}), \sigma^2(\mathbf{z}^{*i})). \tag{2.64}$$

When applying this approximation in a simulation, it means that in every following time step it can happen that we sample a more complicated Gaussian mixture, so the algorithm has to be implemented efficiently. See [135] for hints on an efficient numerical implementation for multistep-ahead prediction.

Other numerical approximations that have been used for the uncertainty propagation, mainly in the context of state-space models, are sequential Monte Carlo methods, e.g. [20, 26, 28].

### 2.7.2 Analytical Approximation of Statistical Moments with a Taylor Expansion

In order to achieve computational simplicity, an analytical approximation that consists of computing only the first two moments, namely, the mean and variance of $p(y^*|\mathbf{z}^*, \mathcal{D})$ can be used.

The mean and variance of the predictive distribution which in general is a non-Gaussian predictive distribution, are approximated with a Gaussian approximation, such that

$$p(y^*|\boldsymbol{\mu}_{\mathbf{z}^*}, \boldsymbol{\Sigma}_{\mathbf{z}^*}, \mathcal{D}) \approx \mathcal{N}(\mu^*, \sigma^{2*}). \tag{2.65}$$

The predictive mean and variance at the model's output corresponding to a noisy input value $\mathbf{z}^*$ are obtained by solving [136]

$$\mu^* = E(\mu(\boldsymbol{\mu}_{\mathbf{z}^*})), \tag{2.66}$$

$$\begin{aligned}\sigma^{2*} &= E(\sigma^2(\boldsymbol{\mu}_{\mathbf{z}^*})) + \mathrm{var}(\mu(\boldsymbol{\mu}_{\mathbf{z}^*})) \\ &= E(\sigma^2(\boldsymbol{\mu}_{\mathbf{z}^*})) + E(\mu^2(\boldsymbol{\mu}_{\mathbf{z}^*})) - (E(\mu(\boldsymbol{\mu}_{\mathbf{z}^*})))^2,\end{aligned} \tag{2.67}$$

where $\mu(\boldsymbol{\mu}_{\mathbf{z}^*})$ and $\sigma^2(\boldsymbol{\mu}_{\mathbf{z}^*})$ denote the mean and variance of the Gaussian predictive distribution in the case when there are no uncertain input values, respectively.

Instead of working with the expressions of $\mu(\boldsymbol{\mu}_{\mathbf{z}^*})$ and $\sigma^2(\boldsymbol{\mu}_{\mathbf{z}^*})$, Eqs. (2.66) and (2.67) are solved by approximating directly $\mu^*$ and $\sigma^{2*}$ using their first- and second-order Taylor expansions, respectively, around $\boldsymbol{\mu}_{\mathbf{z}^*}$. The second-order expansion is required in order to get a correction term for the new variance. This is a relatively rough approximation.

Consequently, within a Gaussian approximation and a Taylor expansion $\mu^*$ and $\sigma^{2*}$ around $\mathbf{z}^* = \boldsymbol{\mu}_{\mathbf{z}^*}$, the predictive distribution is again Gaussian with a mean and variance [136]

$$\mu^* = E(\mu(\boldsymbol{\mu}_{\mathbf{z}^*})) \approx \mathbf{k}(\boldsymbol{\mu}_{\mathbf{z}^*})^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{y}, \tag{2.68}$$

$$\begin{aligned}\sigma^{2*} &= E(\sigma^2(\boldsymbol{\mu}_{\mathbf{z}^*})) + \mathrm{var}(\mu(\boldsymbol{\mu}_{\mathbf{z}^*})) \\ &\approx \sigma^2(\boldsymbol{\mu}_{\mathbf{z}^*}) + \frac{1}{2}\mathrm{tr}\left(\left.\frac{\partial^2\sigma^2(\mathbf{z}^*)}{\partial\mathbf{z}^*\partial\mathbf{z}^{*\mathrm{T}}}\right|_{\mathbf{z}^*=\boldsymbol{\mu}_{\mathbf{z}^*}} \boldsymbol{\Sigma}_{\mathbf{z}^*}\right) \\ &\quad + \left.\frac{\partial\mu(\mathbf{z}^*)}{\partial\mathbf{z}^*}\right|^{\mathrm{T}}_{\mathbf{z}^*=\boldsymbol{\mu}_{\mathbf{z}^*}} \boldsymbol{\Sigma}_{\mathbf{z}^*} \left.\frac{\partial\mu(\mathbf{z}^*)}{\partial\mathbf{z}^*}\right|_{\mathbf{z}^*=\boldsymbol{\mu}_{\mathbf{z}^*}}.\end{aligned} \tag{2.69}$$

Equations (2.68) and (2.69) can be applied in a calculation of the multistep-ahead prediction with the propagation of uncertainty. For a more detailed derivation, see [136] and for further details see Appendix B.

### 2.7.3 Unscented Transformation

The unscented transformation also does not make assumption about the structural nature of the model. It estimates the posterior distribution applying a given nonlinear transformation to a probability distribution that is characterised only in terms of a mean value and covariance.

The unscented transformation takes a finite number of 'sigma points' with the same statistical moments as the input probability distribution, and then maps these sigma points through the mean of the probabilistic dynamics model to obtain the

transformed set of points. The mean and covariance are then set to that of the weighted statistics of the transformed dataset to give an unbiased prediction.

More details of applying the unscented transformation and GP models in the context of state-space models and Kalman filtering are in [27].

### 2.7.4 Analytical Approximation with Exact Matching of Statistical Moments

The alternative approach to approximation is that instead of an approximation of the entire mean and variance, only the integral of Eq. (2.61) is approximated. A simulation with this kind of approximation is named *exact*. In every time step, the model prediction is based on stochastic input data that has a normal distribution and the prediction is a Gaussian mixture, which is approximated with a normal distribution, as depicted in Fig. 2.23 [51] for the case of one input variable for demonstration purposes.

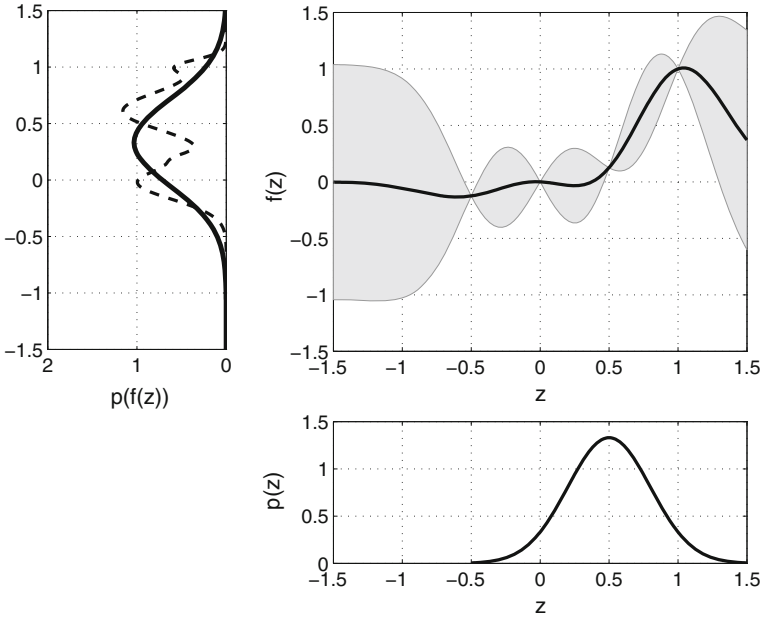


**Fig. 2.23** GP prediction at a stochastic input variable. The input distribution $p(z)$ is the figure on the *bottom right*. The figure on the *right* shows the mean function (*full line*) and the 95 % confidence interval (*shaded*) based on the training data points (points with zero confidence interval in the figure). To determine the expected function value, we average over both the input distribution (*bottom right*) and the function distribution (GP model). The *shaded* distribution represents the exact distribution over the function values. The exact predictive distribution (*dashed line* in the *left figure*) is approximated by a Gaussian (*full line* in the *left figure*) that possesses the mean and the covariance of the exact predictive distribution (known as moment matching)

The expressions for mean and variance are Eqs. (2.66) and (2.67):

$$\begin{aligned}\mu^* &= E(\mu(\boldsymbol{\mu}_{\mathbf{z}^*})),\\ \sigma^{2*} &= E(\sigma^2(\boldsymbol{\mu}_{\mathbf{z}^*})) + \mathrm{var}(\mu(\boldsymbol{\mu}_{\mathbf{z}^*}))\\ &= E(\sigma^2(\boldsymbol{\mu}_{\mathbf{z}^*})) + E(\mu^2(\boldsymbol{\mu}_{\mathbf{z}^*})) - (E(\mu(\boldsymbol{\mu}_{\mathbf{z}^*})))^2.\end{aligned}$$

which can be derived further using

$$E(\zeta(\mathbf{z}^*)) = \int \zeta(\mathbf{z}^*)p(\mathbf{z}^*)d\mathbf{z}^* \tag{2.70}$$

for each of the components in Eqs. (2.66) and (2.67), with $\zeta(\mathbf{z}^*)$ denoting a particular component of these equations.

$$\begin{aligned}\mu^* &= \int \mu(\boldsymbol{\mu}_{\mathbf{z}^*})p(\mathbf{z}^*)d\mathbf{z}^*\\ &= \int \mathbf{k}(\mathbf{z}^*)\mathbf{K}^{-1}\mathbf{y}p(\mathbf{z}^*)d\mathbf{z}^*, \qquad (2.71)\\ \sigma^{2*} &= \int (\kappa(\mathbf{z}^*) - \mathbf{k}^{\mathrm{T}}(\mathbf{z}^*)\mathbf{K}^{-1}\mathbf{k}(\mathbf{z}^*))p(\mathbf{z}^*)d\mathbf{z}^*\\ &\quad + \int \mathbf{k}^{\mathrm{T}}(\mathbf{z}^*)\mathbf{K}^{-1}\mathbf{y}\mathbf{y}^{\mathrm{T}}(\mathbf{K}^{-1})^{\mathrm{T}}\mathbf{k}(\mathbf{z}^*)p(\mathbf{z}^*)d\mathbf{z}^*\\ &\quad - (\mu^*)^2. \qquad (2.72)\end{aligned}$$

The exact derivations for particular covariance functions can be found in [135]. The final results for the case of single and multiple outputs for squared exponential and linear covariance functions can be found in Appendix B.

Predictions for a sparse GP, namely, the FITC, also named SPGP, method in the case of uncertain, i.e. stochastic, input values are introduced in [137] and for the SoR and DTC methods, together with other predictions for stochastic methods in [138].

### 2.7.5 *Propagation of Uncertainty*

The iterative, multistep-ahead prediction is made by feeding back the mean of the predictive distribution as well as the variance of the predictive distribution at each time step, thus taking the uncertainty attached to each intermediate prediction into account. In this way, each input variable for which we wish to predict becomes a normally distributed random variable. Nevertheless, this is an approximation of the Gaussian mixture at the output of the model. The illustration of such a dynamic model simulation is given in Fig. 2.24.

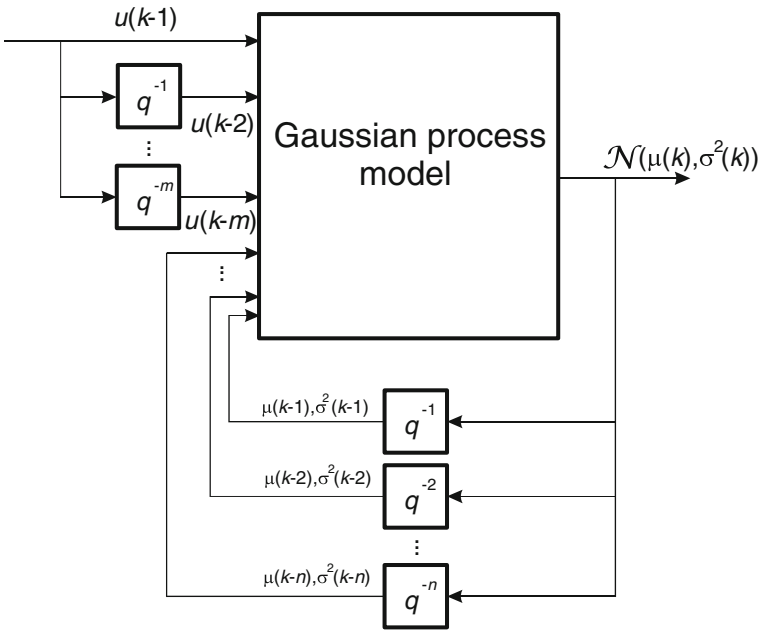


**Fig. 2.24** Block scheme of dynamic system simulation with the iterative method where variance is propagated through the model

These results presume the iterative method, where a one-step-ahead prediction is iteratively repeated. Note that when predicting ahead in time and propagating the uncertainty, the exogenous inputs $u$ are usually assumed to be known and are treated like a deterministic approach. This is also the situation shown in Fig. 2.24. However, the following explanation is general and presumes stochastic input variables.

As with [139], in the case of function observations only, we can predict $h$ steps ahead and propagate the uncertainty of the successive predictions by considering each feedback data $y(k+h-i)$ as a Gaussian random variable, resulting in an $D \times 1$; $D = n + m$ input into the model $\mathbf{z}(k+h) = [y(k+h-1), \ldots, y(k+h-n), u(k+h-1), \ldots, u(k+h-m)]^{\mathrm{T}} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{z}}, \boldsymbol{\Sigma}_{\mathbf{z}})$ at each time step with the mean

$$\boldsymbol{\mu}_{\mathbf{z}} = \begin{bmatrix} \mu_y(k+h-1) \\ \vdots \\ \mu_y(k+h-n) \\ \mu_u(k+h-1) \\ \vdots \\ \mu_u(k+h-m) \end{bmatrix} \tag{2.73}$$

and the covariance matrix

$$\boldsymbol{\Sigma}_{\mathbf{z}} = \begin{bmatrix} \operatorname{var}(y(k+h-1)) & \cdots & \operatorname{cov}(u(k+h-m), y(k+h-1)) \\ \vdots & \ddots & \vdots \\ \operatorname{cov}(y(k+h-1), u(k+h-m)) & \cdots & \operatorname{var}(u(k+h-m)) \end{bmatrix}, \tag{2.74}$$

where the mean values and variances for each entry are computed using one of the approximation methods described beforehand with the equations given in Appendix B.

In general, at time sample $k+l$, we have the random input vector $\mathbf{z}(k+l) = [y(k+l-1), \ldots, y(k+l-n), u(k+h-1), \ldots, u(k+h-m)]^{\mathrm{T}} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{z}}, \boldsymbol{\Sigma}_{\mathbf{z}})$ with the vector of means $\boldsymbol{\mu}_{\mathbf{z}}$ formed by the mean of the predictive distribution of the lagged output data and input data $y(k+l-\tau)$, $\tau = 1, \ldots, n$; $u(k+l-\tau)$, $\tau = 1, \ldots, m$ and the diagonal elements of the $D \times D$; $D = n+m$ input covariance matrix $\boldsymbol{\Sigma}_{\mathbf{z}}$ containing the corresponding predictive covariances. The cross-covariance terms $\operatorname{cov}(y(k+l-i), u(k+l-j))$, for $i, j = 1, \ldots, D$ with $i \neq j$, are obtained by computing $\operatorname{cov}(y(k+l), \mathbf{z}(k+l))$, disregarding the last, the oldest element of $\mathbf{z}(k+l)$:

$$\operatorname{cov}(y(k+l), \mathbf{z}(k+l)) = E(y(k+l)\mathbf{z}(k+l)) - E(y(k+l))E(\mathbf{z}(k+l)). \tag{2.75}$$

Again, the equations for calculating the cross-covariances can be found in Appendix B.

### 2.7.6 When to Use Uncertainty Propagation?

The uncertainty propagation seems to be an issue mainly with two applications of GP models. The first one is the simulation of dynamic system and the second one is the inference of state-vector distribution in GP state-space model.

In the case of GP state-space model of nonlinear systems, the uncertainty propagation is inevitable to get usable approximation of state-vector distribution. Therefore, the rest of this discussion is devoted to uncertainty propagation in the case of the dynamic system's simulation.

A simulation is of major importance as the validation tool for systems identification. However, as the uncertainty propagation extension to the GP modelling method adds a considerable level of complexity, it is worth discussing when uncertainty propagation is best employed.

The uncertainty propagation usually has an effect to the shape of the predictive distribution. It mainly affects, usually increases, its variance, because the predictive distribution becomes wider. Nevertheless, it also affects the mean value. The examples presented in [134, 135, 140] show that the differences between the means

of naive and non-naive cases are usually not huge. It is difficult to argue for the inclusion of uncertainty propagation purely for the sake of improving the quality of the mean prediction. Nevertheless, the computational load for uncertainty propagation is considerable when a model with a large input dataset is employed.

The trade-off between the computational load and the accuracy of the predictive distribution is certainly of importance for an engineer using GP models for dynamic systems identification. The GP modelling approach results in models with an output estimate in the form of a predictive distribution. Through the variance of this output distribution, the GP model becomes more informative. The question is, however, whether we are interested in a precise quantitative value of the predicted variance or we are more interested in qualitative information that the predicted variance carries.

This issue is closely related to the issue of purposiveness of the dynamic system model. In the case that the accurate model multistep-ahead prediction means and variances are of importance for the accuracy of the final product for which the model is developed, then the computational load needs to be taken into account. In general, implementing uncertainty propagation would increase the robustness of the model's response. This means that the increased variance, and improved mean value, obtained with the uncertainty propagation might have a greater chance of enveloping the real response. Such cases would be some cases of predictive control where the mismatch between the model predictions and the real system response makes a difference in the optimisation of future control input values.

On the other hand, when the dynamic system model's predictive variances are used to determine whether the system's output values are predicted outside the region where the identification dataset was available, the qualitative information about the predicted variance's magnitude already serves the purpose. Therefore, the concept of taking into account the uncertainty of the input values and propagating uncertainty for subsequent predictions would not seem to be sensible for applications where the focus is on predicted mean values.

A possible rule of thumb for uncertainty propagation use with the dynamic system's simulation is that the use is decided upon the importance of the exactness of the variance's magnitude. If the variance is to be actively employed in some manner, such as in the design of control systems, the uncertainty propagation may prove to be an important addition. In general, for a lot of the dynamic system's application for engineering purposes, a naive simulation will do.

## 2.8 An Example of GP Model Identification

*Example 2.3* (*Bioreactor identification*) The purpose of this example, adapted from [141], is to demonstrate the GP model identification procedure with a special emphasis on the utility of the prediction variance and other GP model-specific measures for the model validation. The example illustrates how the model is selected. The selected model is then used to demonstrate the influence of increased noise variance on the system's output, the behaviour of the model prediction in unmodelled regions

and the behaviour of the model when a new, unmodelled input is introduced to the system.

A second-order discrete bioreactor model [142, 143] is taken as the system to be identified for demonstration purposes. With the selection of discrete model, the issue of sampling time selection is avoided. In the bioreactor, the microorganisms grow by consuming the substrate.

The bioreactor is given as discrete second-order dynamic system [142] with the sampling time $T_s = 0.5$ s:

$$\begin{aligned} x_1(k+1) &= x_1(k) + 0.5\frac{x_1(k)x_2(k)}{x_1(k)+x_2(k)} - 0.5u(k)x_1(k) \\ x_2(k+1) &= x_2(k) - 0.5\frac{x_1(k)x_2(k)}{x_1(k)+x_2(k)} - 0.5u(k)x_2(k) + 0.05u(k) \\ y(k) &= x_1(k) + \nu(k) \end{aligned} \tag{2.76}$$

where $x_1$ is the concentration of the microorganisms and $x_2$ is the concentration of the substrate. The control input $u$ is the output flow rate, which is limited between $0 \leq u(k) \leq 1$. The output of the system $y$ is the concentration of microorganisms, which is corrupted by white Gaussian noise $\nu$ with a standard deviation $\sigma_\nu = 0.0005$. Our task is to model this system with the GP model and validate the acquired model. The *purpose of the model* is the simulation of the bioreactor.

In the *experiment design*, two separate identification and one validation input signals are acquired, from which the identification and validation data are sampled. Two separate identification signals are acquired so that one is used for the hyperparameter estimation and the other with structure, regressors and covariance function selection. To acquire the first set of identification data, the system described with Eq. (2.76) is excited with the signal $u$ in the form of 4-seconds-long stairs with random amplitude values between $0 \leq u(k) \leq 0.7$. The second set of identification data is obtained with the same kind of signal, but with stairs that last longer. Note that the upper limit of both input signals is chosen so that a part of the operating region remains unmodelled. Before the identification of the models, the signals are normalised around a constant mean value, so that they had a maximum value of one and a minimum value of minus one. From the normalised signals, 602 training points are composed. Later, when the identification results are presented, the data will be scaled into the original range and a constant value for the *mean function* is added.

The GP-NARX *model structure* is used for the model identification. The $i$th training point at the sample step $k$ for the $n$th-order GP model is composed from the input regressors:

$$\mathbf{z}_i = [y(k-1), \ldots, y(k-n),\ u(k-1), \ldots, u(k-m)]^{\mathrm{T}}$$

and the output value $y_i = y(k)$, where $u$ and $y$ are normalised input and output signals. The GP-NOE model structure is going to be used for validation due to the model's purpose.

The squared exponential *covariance function* with the ARD property described with Eq. (2.14) is selected as the initial covariance function that is used for the regressor selection procedure. The ARD property of the selected covariance function ensures that different length scales on different regressors can be used to assess the relative importance of the contributions made by each regressor through a comparison of their lengthscale hyperparameters.

$$C_f(\mathbf{z}_i, \mathbf{z}_j) = \sigma_f^2 \exp\left[-\frac{1}{2}(\mathbf{z}_i - \mathbf{z}_j)^{\mathrm{T}} \boldsymbol{\Lambda}^{-1}(\mathbf{z}_i - \mathbf{z}_j)\right]$$
$$= \sigma_f^2 \exp\left[-\frac{1}{2}\sum_{d=1}^{D} w_d (z_{di} - z_{dj})^2\right],$$

where $w_d = \frac{1}{l_d^2}$; $d = 1, \ldots, D$. The reason behind the selection of the squared exponential covariance function is that we do not know much about the mapping between the input and the output data. Nevertheless, the prior knowledge about most physical systems, among which is the bioreactor, is that the mapping can be modelled with stationary and smooth covariance functions. As we will see later the validation results confirm this prior knowledge.

The *regressor selection* is done with validation-based regressor selection [144] on the second set of data for identification, i.e. where a low-order model is expected based on prior knowledge of the process. The fourth-, third- and second-order models are evaluated with a simulation to obtain an appropriate set of regressors.

All three initial GP models with the same number of delays in the input and output values used for regressors, i.e. $n = m$, are evaluated with a simulation, where the second identification dataset as well as the validation dataset is obtained by simulating the system described with Eq. (2.76) using a different input signal $u$ than for obtaining the first set of identification data.

The results of the regressor selection procedure can be seen in Table 2.3. The log likelihood of the identified model $\ell_1$ described with Eq. (2.32) is used as the objective function for optimisation during the identification and performance measures SMSE, Eq. (2.54) and MSLL, Eq. (2.57), are used for the validation of the simulated model on the second set of identification data and on the validation data. The model has been tested with a naive simulation and with a simulation based on the Monte Carlo numerical approximation. The same conclusions can be drawn from the results of both methods, because the obtained numerical results do not differ significantly. The figures presented in the continuation show the results of the naive simulation.

From the performance measures used on the identification results, shown in the first three rows of Table 2.3, it can be seen that the differences between the identified models in terms of the SMSE and MSLL values evaluating the model simulations on the second set of data are slightly significant. The results on the second set of data which have not been used for the hyperparameters estimation favour a simpler model. The second-order model is also favoured by the principle of Occam's razor [49],

**Table 2.3** Values of the validation performance measures and the hyperparameters of different bioreactor GP models with the best measure values in bold

| Model | Ident. data | | | Valid. data | | Hyperparameters | | | | | | | | | Noise‡ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Order | $\ell_1$ | $\text{SMSE}_2$ | $\text{MSLL}_2$ | SMSE | MSLL | $l_{y_4}=\frac{1}{\sqrt{w_{y_4}}}$ | $l_{y_3}=\frac{1}{\sqrt{w_{y_3}}}$ | $l_{y_2}=\frac{1}{\sqrt{w_{y_4}}}$ | $l_{y_1}=\frac{1}{\sqrt{w_{y_1}}}$ | $l_{u_4}=\frac{1}{\sqrt{w_{u_4}}}$ | $l_{u_3}=\frac{1}{\sqrt{w_{u_3}}}$ | $l_{u_2}=\frac{1}{\sqrt{w_{u_2}}}$ | $l_{u_1}=\frac{1}{\sqrt{w_{u_1}}}$ | $\sigma_f$ | $\sigma_n$ |
| 4 | 1628 | $3.1\times10^{-3}$ | −3.31 | $5.1\times10^{-3}$ | −3.22 | 9.2 | 6.3 | 4.8 | 977 | 406 | 7.2 | 7.0 | 8.9 | 2.5 | $5.0\times10^{-4}$ |
| 3 | 1621 | $2.3\times10^{-3}$ | −3.38 | $3.8\times10^{-3}$ | −3.32 | × | 20.0 | 3.90 | 49.3 | × | 17.4 | 17.3 | 15.6 | 4.2 | $5.1\times10^{-4}$ |
| 2 | 1612 | $1.2\times10^{-3}$ | −3.55 | $1.9\times10^{-3}$ | **−3.50** | × | × | 5.8 | 29.8 | × | × | 13.3 | 14.7 | 5.3 | $5.3\times10^{-4}$ |
| 2♠ | 1603 | $\mathbf{7.8\times10^{-4}}$ | **−3.57** | $\mathbf{1.2\times10^{-3}}$ | −3.49 | × | × | 5.2 | × | × | × | 13.1 | 14.4 | 5.6 | $5.4\times10^{-4}$ |
| 2♣ | 785 | $3.5\times10^{-3}$ | −2.36 | $3.1\times10^{-4}$ | −2.37 | × | × | 4.07 | × | × | × | 10.7 | 8.7 | 3.3 | $2.1\times10^{-3}$ |

*Notes*
The index $_1$ denotes the first set of data for identification
The index $_2$ denotes the second set of data for identification
‡the identified standard deviation of noise $\sigma_n$
♠reduced number of regressors by masking the regressor $y(k-1)$
♣identified on the output signal with the increased standard deviation of noise, $\sigma_\nu = 0.002$

**Table 2.4** Values of log-marginal likelihood $\ell_1$ for the prediction results on the first set of identification data and simulation performance measures for the second set of identification data for different covariance functions with the best measure values in bold

| 2nd order model | Ident. data | | |
|---|---|---|---|
| Covariance function | $\ell_1$ | $\text{SMSE}_2$ | $\text{MSLL}_2$ |
| Squared exponential + ARD | **1603** | $7.8 \times 10^{-4}$ | **−3.57** |
| Matérn + ARD $d = \frac{3}{2}$ | 1591 | $\mathbf{7.2 \times 10^{-4}}$ | −3.54 |
| Rational quadratic + ARD | **1603** | $4.1 \times 10^{-3}$ | −3.31 |
| Linear + ARD | 1281 | $1.7 \times 10^{-2}$ | −1.46 |
| Matérn $d = \frac{3}{2}$ | 1587 | $1.2 \times 10^{-3}$ | −2.03 |
| Matérn $d = \frac{5}{2}$ | 1597 | $1.7 \times 10^{-3}$ | −1.58 |
| Neural network | 1596 | $5.2 \times 10^{-3}$ | −1.00 |
| Squared exponential | 1600 | $1.6 \times 10^{-3}$ | −1.35 |

stating that the most simple explanation for the given problem should be used. The results obtained on the validation data confirm these results.

The hyperparameters $l_{z_i}$ reflect the relative importance of the regressors $z(k-i)$ and in all the model structures the regressor $y(k-1)$ exhibits a lower importance due to the large value of the associated hyperparameter $l_{y_1}$. The removal of this regressor from the selected second-order model results in even better results. Note that the variance of modelled noise is likely to be a small amount greater than the ground truth. This effect is related to errors-in-variables problem.

This regressor selection procedure led us to the GP model with the following regressors: $y(k-2), u(k-1), u(k-2)$.

*Covariance function selection* is illustrated with Table 2.4, where the same performance measures as for the regressors' selection are gathered for different covariance functions with and without the ARD property for the second-order model.

The squared exponential covariance function and Matérn covariance function with the ARD property have the best results from the tested covariance functions. Consequently, the squared exponential covariance function with the ARD property is kept for the model.

The *model validation* is pursued with the dynamic model simulation according to the purpose of the model. The naive simulation is selected in our case, because no special accuracy needs are expressed for the posterior variance in the model purpose and the obtained accuracy for mean values is acceptable. In this way, we are avoiding computational complexity due to the propagation of uncertainty through the model, at the cost of only a slightly lower accuracy for this case.

The simulation results on the validation data for the selected second-order model can be seen in Fig. 2.25, where the model's output and the noise-free target are depicted. It can be seen that most of the time the value of the predicted standard deviation $\sigma_n$ is around $5 \times 10^{-4}$, corresponding to the noise level present at the

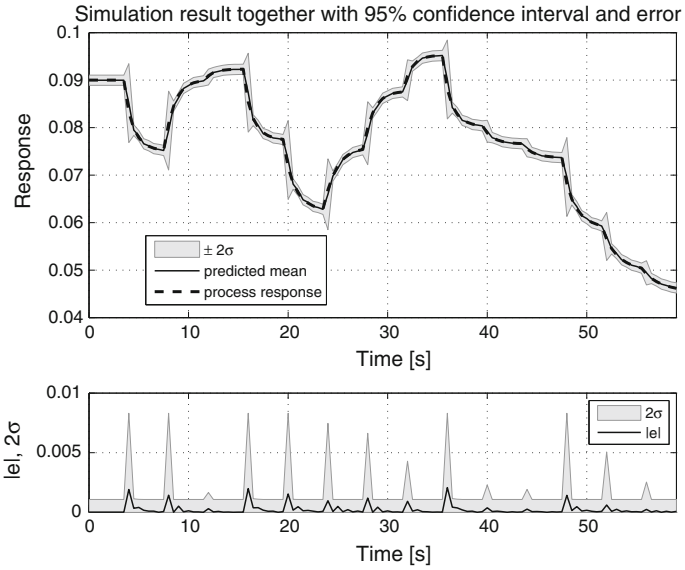


**Fig. 2.25** Validation with simulation for the bioreactor GP model, $|e|$ is the absolute value of the error between the predicted mean and the true values

system's output. The prediction variance increases at the steps where the input variable $u$ changes its value due to the small number of training points describing those regions. Note that the error of the model's predicted mean values remains inside the 95 % confidence limits, defined within $\pm 2\sigma$, indicating the level of trust that can be put in the prediction.

These model validation results will serve as the reference for the observation of how different conditions can influence the model prediction and validation.

First, it will be shown how the model prediction changes when the model reaches the unmodelled region of the system. As there is no identification data available nearby, the model must extrapolate from the data describing the neighbouring regions and the prior mean function in order to make the predictions. This worsens the prediction mean, but is also accompanied by an increase of the prediction variance, thus widening the model's confidence limits. This effect can be observed in Fig. 2.26, where the values of the control input were increased above the $u(k) > 0.7$ at time $t > 12\,\text{s}$.

Second, we would like to show how the increase in the system's output noise variance reflects in the identified model. For this purpose, the standard deviation of the system's output noise was increased to $\sigma_\nu = 2 \times 10^{-3}$. The set of control input signals, used for generating the identification and validation data, is the same as in the reference example. The second-order GP model is identified. The values of the GP model's hyperparameters can be seen in Table 2.3.

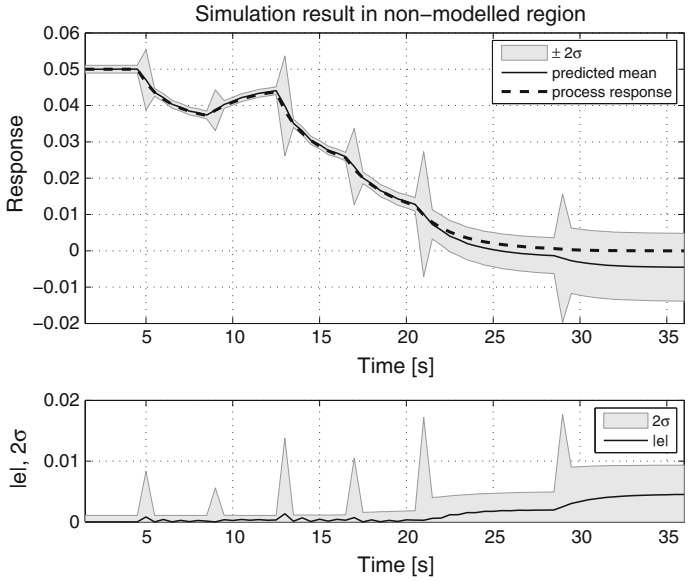


**Fig. 2.26** GP model prediction in the unmodelled region

The mean model prediction is satisfactory and the prediction variance is increased at the expense of a higher output noise variance being predicted, as can be observed from the simulation results on the validation data in Fig. 2.27. The estimation of the system's output noise is satisfactorily close to the real value, i.e. $\sigma_n = 2.1 \times 10^{-3}$, which also shows that the value of the hyperparameter $\sigma_n^2$ tends to the value of the system's output noise when enough identification data is used. The value of SMSE is slightly worse than in the reference example, as this model is identified with more noise present in the identification, i.e. training, data. Also, the value of MSLL is slightly worse as, despite the increased variance, the influence of the prediction error prevails.

Finally, we would like to show how the unmodelled regressor influences the model. For this purpose, an additional control input signal $v$, not correlated to the input signal $u$, was added to the system. The effect of this unmeasured input variable is the same as the effect of the control input signal $u$ and could represent an additional outlet or leak of the system described with Eq. (2.76), which changes the description to

$$\begin{aligned} x_1(k+1) &= x_1(k) + 0.5\frac{x_1(k)x_2(k)}{x_1(k)+x_2(k)} - 0.5u(k)x_1(k) - 0.5v(k)x_1(k) \\ x_2(k+1) &= x_2(k) - 0.5\frac{x_1(k)x_2(k)}{x_1(k)+x_2(k)} - 0.5u(k)x_2(k) \\ &\quad - 0.5v(k)x_2(k) + 0.05u(k) + 0.05v(k) \\ y(k) &= x_1(k) + \nu(k) \end{aligned} \tag{2.77}$$

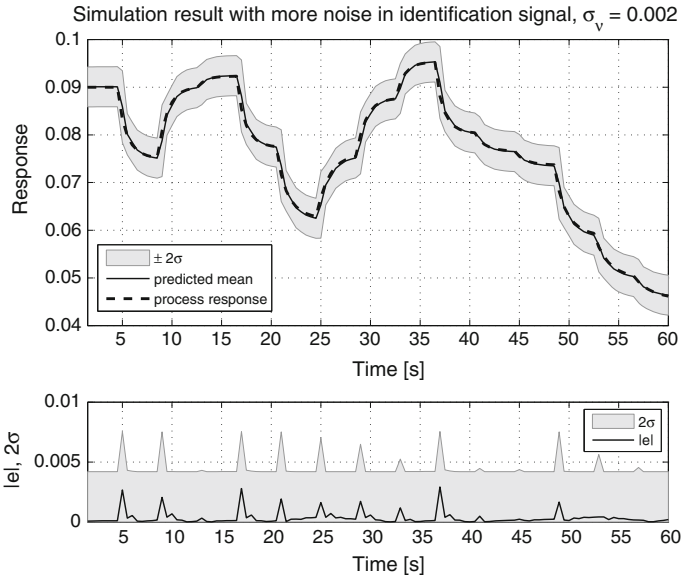


**Fig. 2.27** Influence of the increased system's output noise variance on the GP model

The reference GP model is used for the simulation, where the input signal $v$ is not present and therefore neglected for the identification of the model. The control input signal in the form of a step $v = 0.05$ is introduced into the system at the simulation time 30 s. The (non)influence of the unmodelled input signal on the prediction variance, when the model operates in the region with sufficient identification data, can be seen in Fig. 2.28. The model's simulation response from time $t = 30$ s worsens, but the 95 % confidence interval remains tight. This example shows that the variance, obtained with the model simulation, cannot be informative with respect to the unaccounted influences on the system in the identification data. Note that the results are different if the model prediction is pursued instead of the model simulation.

With the bioreactor example, the following properties of the GP model have been illustrated:

1. The hyperparameters' ARD property can be effectively used to reduce the number of regressors of the identified model.
2. There are two possible causes for the increase of the prediction variance:
   - the particular region of the system, where the model makes predictions, is described with insufficient identification data; and
   - the data describing particular regions contains more noise. In the example, this has been shown for the whole region, but the same applies when the noise is increased only in part of the system's operating region.

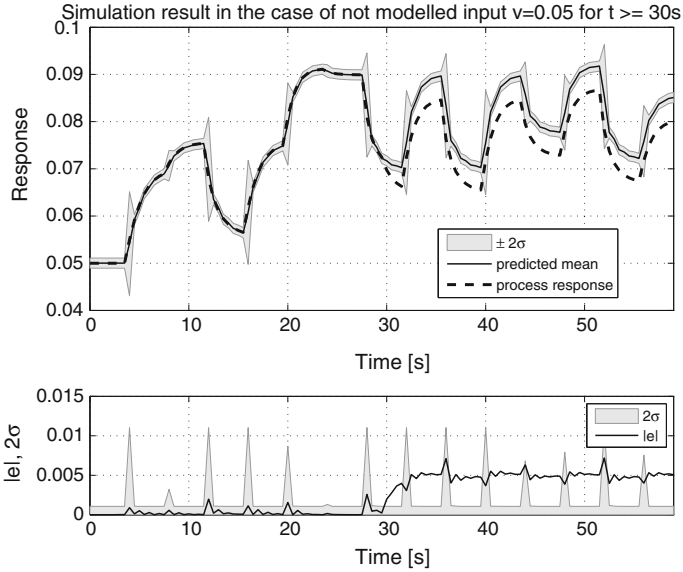


**Fig. 2.28** Influence of the unmodelled input signal on the GP model prediction

These two causes cannot be easily distinguished without prior knowledge about the identified system.

3. When the unmodelled influence is introduced to the system, the model simulation response, including the variance, does not change.

## References

1. Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P.Y., Hjalmarsson, H., Juditsky, A.: Nonlinear black-box modelling in system identification: a unified overview. Automatica **31**(12), 1691–1724 (1995)
2. Ljung, L.: System Identification - Theory for the User, 2nd edn. Prentice Hall, Upper Saddle River, NJ (1999)
3. Bai, E.W.: Prediction error adjusted Gaussian process for nonlinear non-parametric system identification. In: 16th IFAC Symposium on System Identification, pp. 101–106. IFAC, Brussels (2012)
4. Johansen, T.A., Shorten, R., Murray-Smith, R.: On the interpretation and identification of dynamic Takagi-Sugeno fuzzy models. IEEE Trans. Fuzzy Syst. **8**, 297–313 (2000)
5. Murray-Smith, R., Johansen, T.A. (eds.): Multiple Model Approaches to Modelling and Control. Taylor and Francis, London (1997)
6. Murray-Smith, R., Johansen, T.A., Shorten, R.: On transient dynamics, off-equilibrium behaviour and identification in blended multiple model structures. In: Proceedings of European Control Conference, pp. BA-14. Karlsruhe (1999)
7. Suykens, J.A.K., Gestel, T.V., Brabanteer, J.D., Moor, B.D., Vandewalle, J.: Least Squares Support Vector Machines. World Scientific, Singapore (2002)

8. Snelson, E., Rasmussen, C.E., Ghahramani, Z.: Warped Gaussian processes. In: Thrun, S., Saul, L., Schölkopf, B. (eds.) Advances in Neural Information Processing Systems, vol. 16, pp. 337–344 (2004)
9. Lazáro-Gredilla, M.: Bayesian warped Gaussian processes. Advances in Neural Information Processing Systems, vol. 26. MIT Press, Cambridge, MA (2013)
10. Nelles, O.: Nonlinear System Identification. Springer, Berlin (2001)
11. Goodwin, G.C.: Identification: experiment design. In: Singh, M.G. (ed.) Systems and Control Encyclopedia, vol. 4 (I–L), pp. 2257–2264. Pergamon Press, Oxford (1987)
12. Norgaard, M., Ravn, O., Poulsen, N.K., Hansen, L.K.: Neural Networks for Modelling and Control of Dynamic Systems. A Practitioner's Handbook. Advanced Textbooks in Control and Signal Processing. Springer, London (2000)
13. Pronzato, L.: Optimal experimental design and some related control problems. Automatica **44**(2), 303–325 (2008)
14. Kocijan, J., Přikryl, J.: Soft sensor for faulty measurements detection and reconstruction in urban traffic. In: Proceedings 15th IEEE Mediterranean Electromechanical Conference (MELECON), pp. 172–177. Valletta (2010)
15. Haykin, S.: Neural Networks, A Comprehensive Foundation. Macmillan College Publishing Company, New York, NY (1994)
16. Ackermann, E.R., de Villiers, J.P., Cilliers, P.J.: Nonlinear dynamic systems modeling using Gaussian processes: predicting ionospheric total electron content over South Africa. J. Geophys. Res. A: Space Phys. **116**(10) (2011)
17. Kocijan, J., Girard, A., Banko, B., Murray-Smith, R.: Dynamic systems identification with Gaussian processes. In: Troch, I., Breitenecker, F. (eds.) Proceedings of 4th IMACS Symposium on Mathematical Modelling (MathMod), pp. 776–784. Vienna (2003)
18. Kocijan, J., Petelin, D.: Output-error model training for Gaussian process models. In: Dobnikar, A., Lotrič, U., Šter, B. (eds.) Adaptive and Natural Computing Algorithms. Lecture Notes in Computer Science, vol. 6594, pp. 312–321. Springer, Berlin (2011)
19. Frigola, R., Rasmussen, C.E.: Integrated pre-processing for Bayesian nonlinear system identification with Gaussian processes. In: 52nd IEEE Conference on Decision and Control (CDC) (2013)
20. Frigola, R., Lindsten, F., Schön, T.B., Rasmussen, C.E.: Bayesian inference and learning in Gaussian process state-space models with particle MCMC. In: L. Bottou, C. Burges, Z. Ghahramani, M. Welling, K. Weinberger (eds.) Advances in Neural Information Processing Systems, vol. 26, pp. 3156–3164 (2013)
21. Wang, J., Fleet, D., Hertzmann, A.: Gaussian process dynamical models. Adv. Neural Inf. Process. Syst. **18**, 1441–1448 (2005)
22. Levine, W.S. (ed.): The Control Handbook. CRC Press, IEEE Press, Boca Raton (1996)
23. Ko, J., Fox, D.: GP-Bayesfilters: Bayesian filtering using Gaussian process prediction and observation models. Auton. Robots **27**(1), 75–90 (2009)
24. Deisenroth, M.P., Turner, R.D., Huber, M.F., Hanebeck, U.D., Rasmussen, C.E.: Robust filtering and smoothing with Gaussian processes. IEEE Trans. Autom. Control **57**(7), 1865–1871 (2012). doi:10.1109/TAC.2011.2179426
25. Deisenroth, M.P., Huber, M.F., Hannebeck, U.D.: Analytic moment-based Gaussian process filtering. In: Proceedings of the 26th Annual International Conference on Machine Learning, pp. 225–232. Montreal (2009)
26. Tong, C.H., Furgale, P., Barfoot, T.D.: Gaussian process Gauss–Newton: non-parametric state estimation. In: 2012 Ninth Conference on Computer and Robot Vision, pp. 206–213. Toronto (2012)
27. Turner, R., Rasmussen, C.E.: Model based learning of sigma points in unscented Kalman filtering. Neurocomputing **80**, 47–53 (2012)
28. Wang, Y., Chaib-draa, B.: A marginalized particle Gaussian process regression. In: Neural Information Processing Systems, vol. 25 (2012)
29. Wang, Y., Chaib-draa, B.: An adaptive nonparametric particle filter for state estimation. In: 2012 IEEE International Conference on Robotics and Automation, pp. 4355–4360. IEEE (2012)

30. Peltola, V., Honkela, A.: Variational inference and learning for non-linear state-space models with state-dependent observation noise. In: Proceedings of the 2010 IEEE International Workshop on Machine Learning for Signal Processing, MLSP 2010, pp. 190–195 (2010)
31. Turner, R., Deisenroth, M.P., Rasmussen, C.E.: System identification in Gaussian process dynamical systems. Nonparametric Bayes Workshop at NIPS. Whistler (2009)
32. Turner, R., Deisenroth, M.P., Rasmussen, C.E.: State-space inference and learning with Gaussian processes. In: Proceedings of 13th International Conference on Artificial Intelligence and Statistics, vol. 9, pp. 868–875. Sardinia (2010)
33. Snelson, E., Ghahramani, Z.: Sparse Gaussian processes using pseudo-inputs. In: Weiss, Y., Schölkopf, B., Platt, J. (eds.) Advances in Neural Information Processing Systems, vol. 18, pp. 1257–1264. MIT Press, Cambridge, MA (2006)
34. Hartikainen, J., Riihimäki, J., Särkkä, S.: Sparse spatio-temporal Gaussian processes with general likelihoods. In: Honkela, T., Duch, W., Girolami, M., Kaski, S. (eds.) Artificial Neural Networks and Machine Learning - ICANN 2011. Lecture Notes in Computer Science, vol. 6791, pp. 193–200. Springer, Berlin (2011)
35. Hartikainen, J., Seppänen, M., Särkkä, S.: State-space inference for non-linear latent force models with application to satellite orbit prediction. In: Proceedings of the 29th International Conference on Machine Learning (ICML-12), pp. 903–910. Edinburgh (2012)
36. Särkkä, S., Hartikainen, J.: Infinite-dimensional kalman filtering approach to spatio-temporal Gaussian process regression. In: Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS). JMLR: W&CP, vol. 22, pp. 993–1001 (2012)
37. Särkkä, S., Solin, A., Hartikainen, J.: Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing: a look at Gaussian process regression through Kalman filtering. IEEE Signal Process. Mag. **30**(4), 51–61 (2013). doi:10.1109/MSP.2013.2246292
38. Chiuso, A., Pillonetto, G., De Nicolao, G.: Subspace identification using predictor estimation via Gaussian regression. In: Proceedings of the IEEE Conference on Decision and Control (2008)
39. Blum, A.L., Langley, P.: Selection of relevant features and examples in machine learning. Artif. Intell. **97**(1–2), 245–271 (1997)
40. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–1182 (2003)
41. Kohavi, R., John, G.H.: Wrappers for feature subset selection. Artif. Intell. **97**(1–2), 273–324 (1997)
42. May, R., Dandy, G., Maier, H.: Review of input variable selection methods for artificial neural networks (Chap). Artificial Neural Networks - Methodological Advances and Biomedical Applications, pp. 19–44. InTech, Rijeka (2011)
43. Lind, I., Ljung, L.: Regressor selection with the analysis of variance method. Automatica **41**, 693–700 (2005)
44. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer Science + Business Media, New York, NY (2006)
45. He, X., Asada, H.: A new method for identifying orders of input-output models for nonlinear dynamical systems. In: Proceedings of the American Control Conference, San Francisco, CA, pp. 2520–2523 (1993)
46. Bomberger, J.D., Seborg, D.E.: Determination of model order for NARX models directly from input-output data. J. Process Control **8**(5–6), 459–468 (1998)
47. Broer, H., Takens, F.: Dynamical Systems and Chaos. Epsilon Uitgaven, Utrecht (2009)
48. Stark, J., Broomhead, D.S., Davies, M.E., Huke, J.: Delay embeddings for forced systems: II stochastic forcing. J. Nonlinear Sci. **13**(6), 519–577 (2003)
49. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press, Cambridge, MA (2006)
50. Thompson, K.: Implementation of Gaussian process models for non-linear system identification. Ph.D. thesis, University of Glasgow, Glasgow (2009)
51. Deisenroth, M.P.: Efficient reinforcement learning using Gaussian processes. Ph.D. thesis, Karlsruhe Institute of Technology, Karlsruhe (2010)

52. Murray-Smith, R., Girard, A.: Gaussian process priors with ARMA noise models. In: Proceedings of Irish Signals and Systems Conference, pp. 147–152. Maynooth (2001)
53. Ažman, K., Kocijan, J.: Identifikacija dinamičnega sistema z znanim modelom šuma z modelom na osnovi Gaussovih procesov. In: Zajc, B., Trost, A. (eds.) Zbornik petnajste elektrotehniške in računalniške konference (ERK), pp. 289–292. Portorož (2006). (In Slovene)
54. MAC Multi-Agent Control: Probabilistic reasoning, optimal coordination, stability analysis and controller design for intelligent hybrid systems (2000–2004). Research Training Network, 5th EU framework
55. Neal, R.M.: Bayesian Learning for Neural Networks. Lecture Notes in Statistics, vol. 118. Springer, New York, NY (1996)
56. Stein, M.L.: Interpolation of Spatial Data. Springer, New York, NY (1999)
57. Duvenaud, D.: The kernel cookbook: advice on covariance functions (2013)
58. Álvarez, M.A., Luengo, D., Lawrence, N.D.: Latent force models. J. Mach. Learn. Res. - Proc. Track **5**, 9–16 (2009)
59. Álvarez, M.A., Peters, J., Schölkopf, B., Lawrence, N.D.: Switched latent force models for movement segmentation. In: Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6–9 December 2010, Vancouver, pp. 55–63 (2010)
60. Honkela, A., Girardot, C., Gustafson, E.H., Liu, Y.H., Furlong, E.M.F., Lawrence, N.D., Rattray, M.: Model-based method for transcription factor target identification with limited data. Proc. Natl. Acad. Sci. USA **107**(17), 7793–7798 (2010)
61. Neo, K.K.S., Leithead, W.E., Zhang, Y.: Multi-frequency scale Gaussian regression for noisy time-series data. In: UKACC International Control Conference. Glasgow (2006)
62. Nguyen-Tuong, D., Peters, J.: Using model knowledge for learning inverse dynamics. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 2677–2682 (2010)
63. Zhang, Y., Leithead, W.E.: Exploiting Hessian matrix and trust region algorithm in hyperparameters estimation of Gaussian process. Appl. Math. Comput. **171**(2), 1264–1281 (2005)
64. Petelin, D., Filipič, B., Kocijan, J.: Optimization of Gaussian process models with evolutionary algorithms. In: Dobnikar, A., Lotrič, U., Šter, B. (eds.) Adaptive and Natural Computing Algorithms. Lecture Notes in Computer Science, vol. 6594, pp. 420–429. Springer, Berlin (2011)
65. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Natural Computing Series. Springer, Heidelberg (2003)
66. Hachino, T., Kadirkamanathan, V.: Multiple Gaussian process models for direct time series forecasting. IEEJ Trans. Electr. Electron. Eng. **6**(3), 245–252 (2011)
67. Hachino, T., Takata, H.: Identification of continuous-time nonlinear systems by using a Gaussian process model. IEEJ Trans. Electr. Electron. Eng. **3**(6), 620–628 (2008)
68. Storn, R., Price, K.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. J. Global Opt. **11**, 341–359 (1997)
69. Price, K., Storn, R., Lampinen, J.: Differential Evolution. Natural Computing Series. Springer, Heidelberg (2005)
70. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, pp. 1942–1948. IEEE Press. (1995)
71. Kennedy, J., Eberhart, R., Shi, Y.: Swarm Intelligence. Morgan Kaufmann, San Francisco, CA (2001)
72. Hachino, T., Yamakawa, S.: Non-parametric identification of continuous-time Hammerstein systems using Gaussian process model and particle swarm optimization. Artif Life Robotics **17**(1), 35–40 (2012)
73. Birge, B.: Particle swarm optimization toolbox. http://www.mathworks.com/matlabcentral/fileexchange/7506-particle-swarm-optimization-toolbox
74. Pohlheim, H.: GEATbx - the genetic and evolutionary algorithm toolbox for Matlab. http://www.geatbx.com/

75. McHutchon, A., Rasmussen, C.E.: Gaussian process training with input noise. In: J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, K. Weinberger (eds.) Advances in Neural Information Processing Systems, vol. 24, pp. 1341–1349 (2011)
76. Shi, J.Q., Choi, T.: Gaussian Process Regression Analysis for Functional Data. Chapman and Hall/CRC, Taylor & Francis group, Boca Raton, FL (2011)
77. Seeger, M.W., Kakade, S.M., Foster, D.P.: Information consistency of nonparametric Gaussian process methods. IEEE Trans. Inf. Theory **54**(5), 2376–2382 (2008)
78. Trobec, R., Vajteršic, M., Zinterhof, P. (eds.): Parallel Computing: Numerics, Applications, and Trends. Springer, London (2009)
79. Kurzak, J., Bader, D.A., Dongarra, J. (eds.): Scientific Computing with Multicore and Accelerators. Chapman & Hall/CRC Computational Science Series, 1st edn. CRC Press, Boca Raton, FL (2011)
80. Shen, J.P., Lipasti, M.H.: Modern Processor Design: Fundamentals of Superscalar Processors. McGraw-Hill Series in Electrical and Computer Engineering, 1st edn. McGraw-Hill, New York, NY (2004)
81. Kirk, D.B., Hwu, W.W.: Programming Massively Parallel Processors A Hands-on Approach, 1st edn. Morgan Kaufmann, San Francisco, CA (2010)
82. Tomov, S., Dongarra, J., Baboulin, M.: Towards dense linear algebra for hybrid GPU accelerated manycore systems. Parallel Comput. **36**(5), 232–240 (2010)
83. NVIDIA Corporation, Santa Clara, CA: Cuda Programming Guide Version 2.3.1 (2009)
84. Advanced Micro Devices, Inc.: AMD Accelerated Parallel Processing OpenCL Programming Guide. Sunnyvale, CA (2011)
85. Srinivasan, B.V., Duraiswami, R.: Scaling kernel machine learning algorithm via the use of GPUs. In: GPU Technology Conference, NVIDIA Research Summit. NVIDIA (2009)
86. Srinivasan, B.V., Hu, Q., Duraiswami, R.: GPUML: graphical processors for speeding up kernel machines. In: Workshop on High Performance Analytics - Algorithms, Implementations, and Applications, SIAM Conference on Data Mining. SIAM (2010)
87. Musizza, B., Petelin, D., Kocijan, J.: Accelerated learning of Gaussian process models. In: Proceedings of the 7th EUROSIM Congress on Modelling and Simulation (2010)
88. Blackford, L., Petitet, A., Pozo, R., Remington, K., Whaley, R., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., et al.: An updated set of basic linear algebra subprograms (BLAS). ACM Trans. Math. Softw. **28**(2), 135–151 (2002)
89. Volkov, V., Demmel, J.: LU, QR and Cholesky factorizations using vector capabilities of GPUs. Technical Report UCB/EECS-2008-49, EECS Department, University of California, Berkeley, CA (2008). http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-49.html
90. Humphrey, J.R., Price, D.K., Spagnoli, K.E., Paolini, A.L., J.Kelmelis, E.: CULA: hybrid GPU accelerated linear algebra routines. In: Kelmelis, E.J. (ed.) Proceeding of SPIE: Modeling and Simulation for Defense Systems and Applications V, vol. 7705. SPIE (2010)
91. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide, 3rd edn. Society for Industrial and Applied Mathematics, Philadelphia, PA (1999)
92. Smola, A.J., Bartlett, P.L.: Sparse greedy Gaussian process regression. Advances in Neural Information Processing Systems, vol. 13, pp. 619–625. MIT Press, Cambridge, MA (2001)
93. Wahba, G.: Spline Models for Observational Data. Society for Industrial and Applied Mathematics, Philadelphia, PA (1990)
94. Williams, C.K.I., Seeger, M.: Using the Nyström method to speed up kernel machines. Advances in Neural Information Processing Systems, pp. 682–688. MIT Press, Cambridge, MA (2001)
95. Quiñonero-Candela, J., Rasmussen, C.E.: A unifying view of sparse approximate Gaussian process regression. J. Mach. Learn. Res. **6**, 1939–1959 (2005)
96. Quiñonero-Candela, J., Rasmussen, C.E., Williams, C.K.I.: Approximation methods for Gaussian process regression (Chap). Large Scale Learning Machines, pp. 203–223. MIT Press, Cambridge, MA (2007)

97. Chalupka, K., Williams, C.K.I., Murray, I.: A framework for evaluating approximation methods for Gaussian process regression. J. Mach. Learn. Res. **14**, 333–350 (2013)
98. Murray, I.: Gaussian processes and fast matrix-vector multiplies. In: Presented at the Numerical Mathematics in Machine Learning workshop at the 26th International Conference on Machine Learning (ICML 2009), Montreal (2009). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.153.7688
99. Leithead, W.E., Zhang, Y., Leith, D.J.: Time-series Gaussian process regression based on Toeplitz computation of O(N2) operations and O(N) level storage. In: Joint 44th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC). Sevilla (2005)
100. Leithead, W.E., Zhang, Y.: O(N-2)-operation approximation of covariance matrix inverse in Gaussian process regression based on quasi-Newton BFGS method. Commun. Stat.-Simul. Comput. **36**(2), 367–380 (2007)
101. Zhang, Y., Leithead, W.E.: Approximate implementation of the logarithm of the matrix determinant in Gaussian process regression. J. Stat. Comput. Simul. **77**(4), 329–348 (2007)
102. Lawrence, N.D., Seeger, M., Herbrich, R.: Fast sparse Gaussian process methods: the informative vector machine. In: Becker, S., Thrun, S., Obermayer, K. (eds.) Advances in Neural Information Processing Systems, vol. 15, pp. 609–616. MIT Press, Cambridge, MA (2003)
103. Csató, L., Opper, M.: Sparse online Gaussian processes. Neural Comput. **14**(3), 641–668 (2002)
104. Sathiya Keerthi, S., Chu, W.: A matching pursuit approach to sparse Gaussian process regression. In: Weiss, Y., Schölkopf, B., Platt, J. (eds.) Advances in Neural Information Processing Systems, vol. 18, pp. 643–650. MIT Press, Cambridge, MA (2006)
105. Seeger, M., Williams, C.K.I., Lawrence, N.D.: Fast forward selection to speed up sparse Gaussian process regression. In: Ninth International Workshop on Artificial Intelligence and Statistics, Society for Artificial Intelligence and Statistics (2003)
106. Lazáro-Gredilla M., Quiñonero-Candela J., Figueiras-Vidal A.: Sparse spectral sampling. Technical report, Microsoft Research, Cambridge (2007)
107. Lazáro-Gredilla, M., Quiñonero-Candela, J., Rasmussen, C.E., Figueiras-Vidal, A.R.: Sparse spectrum Gaussian process regression. J. Mach. Learn. Res. **11**, 1865–1881 (2010)
108. Titsias, M.: Variational learning of inducing variables in sparse Gaussian processes. In: The 12th International Conference on Artificial Intelligence and Statistics (AISTATS), vol. 5, pp. 567–574 (2009)
109. Ni, W., Tan, S.K., Ng, W.J., Brown, S.D.: Moving-window GPR for nonlinear dynamic system modeling with dual updating and dual preprocessing. Ind. Eng. Chem. Res. **51**(18), 6416–6428 (2012)
110. Oba, S., Sato, M., Ishii, S.: On-line learning methods for Gaussian processes. In: Dorffner, G., Bischof, H., Hornik, K. (eds.) Artificial Neural Networks (ICANN 2001). Lecture Notes in Computer Science, vol. 2130, pp. 292–299. Springer, Berlin (2001). doi:10.1007/3-540-44668-0
111. Grbić, R., Slišković, D., Kadlec, P.: Adaptive soft sensor for online prediction based on moving window Gaussian process regression. In: 2012 11th International Conference on Machine Learning and Applications, pp. 428–433 (2012)
112. Ranganathan, A., Yang, M.H., Ho, J.: Online sparse Gaussian process regression and its applications. IEEE Trans. Image Process. **20**, 391–404 (2011)
113. Nguyen-Tuong, D., Seeger, M., Peters, J.: Real-time local GP model learning. From Motor Learning to Interaction Learning in Robots, pp. 193–207. Springer, Heidelberg (2010)
114. Tresp, V.: A Bayesian committee machine. Neural Comput. **12**, 2719–2741 (2000)
115. Shi, J.Q., Murray-Smith, R., Titterington, D.M.: Hierarchical Gaussian process mixtures for regression. Statist. Comput. **15**(1), 31–41 (2005)
116. Gregorčič, G., Lightbody, G.: Local model identification with Gaussian processes. IEEE Trans. Neural Netw. **18**(5), 1404–1423 (2007)
117. Petelin, D., Kocijan, J.: Control system with evolving Gaussian process model. In: Proceedings of IEEE Symposium Series on Computational Intelligence, SSCI 2011. IEEE, Paris (2011)

118. Angelov, P., Filev, D.P., Kasabov, N.: Evolving Intelligent Systems: Methodology and Applications. IEEE Press Series on Computational Intelligence. Wiley-IEEE Press, New York, NY (2010)
119. Åström, K.J., Wittenmark, B.: Computer Controlled Systems: Theory and Design. Prentice Hall, Upper Saddle River, NJ (1984)
120. Isermann, R., Lachman, K.H., Matko, D.: Adaptive Control Systems. Systems and Control Engineering. Prentice Hall International, Upper Saddle River, NJ (1992)
121. Fritzke, B.: Growing cell structures—a self-organizing network for unsupervised and supervised learning. Neural Netw. **7**(9), 1441–1460 (1994)
122. Angelov, P., Buswell, R.: Evolving rule-based models: a tool for intelligent adaptation. In: Proceedings of the Joint 9th NAFIPS International Conference, pp. 1062–1066. IEEE Press (2001)
123. Kasabov, N.K.: Evolving Connectionist Systems: Methods and Applications in Bioinformatics, Brain Study and Intelligent Machines. Springer, New York, NY (2002)
124. Abusnina, A., Kudenko, D.: Adaptive soft sensor based on moving Gaussian process window, pp. 1051–1056. IEEE (2013)
125. Petelin, D., Grancharova, A., Kocijan, J.: Evolving Gaussian process models for the prediction of ozone concentration in the air. Simul. Modell. Pract. Theory **33**(1), 68–80 (2013)
126. Deisenroth, M.P., Rasmussen, C.E.: PILCO: A model-based and data-efficient approach to policy search. In: Proceedings of the 28th International Conference on Machine Learning (ICML 2011). Bellevue, WA (2011)
127. Ni, W., Tan, S.K., Ng, W.J.: Recursive GPR for nonlinear dynamic process modeling. Chem. Eng. J. **173**(2), 636–643 (2011)
128. Ni, W., Wang, K., Chen, T., Ng, W.J., Tan, S.K.: GPR model with signal preprocessing and bias update for dynamic processes modeling. Control Eng. Pract. **20**(12), 1281–1292 (2012)
129. Duvenaud, D., Lloyd, J.R., Grosse, R., Tenenbaum, J.B., Ghahramani, Z.: Structure discovery in nonparametric regression through compositional kernel search. In: Proceedings of the 30th International Conference on Machine Learning (2013)
130. Seeger, M.: Low rank updates for the Cholesky decomposition. Technical report, University of California, Berkeley, CA (2008)
131. Murray-Smith, D.J.: Methods for the external validation of continuous system simulation models: a review. Math. Comput. Modell. Dyn. Syst. **4**(1), 5–31 (1998)
132. Cawley, G.C., Talbot, N.L.C.: On over-fitting in model selection and subsequent selection bias in performance evaluation. J. Mach. Learn. Res. **11**, 2079–2107 (2010)
133. Hvala, N., Strmčnik, S., Šel, D., Milanić, S., Banko, B.: Influence of model validation on proper selection of process models — an industrial case study. Comput. Chem. Eng. **29**, 1507–1522 (2005)
134. Kocijan, J., Girard, A., Banko, B., Murray-Smith, R.: Dynamic systems identification with Gaussian processes. Math. Comput. Modell. Dyn. Syst. **11**(4), 411–424 (2005)
135. Girard, A.: Approximate methods for propagation of uncertainty with Gaussian process models. Ph.D. thesis, University of Glasgow, Glasgow (2004). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.63.8313
136. Girard, A., Rasmussen, C.E., Candela, J.Q., Murray-Smith, R.: Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting. In: Becker, S., Thrun, S., Obermayer, K. (eds.) Advances in Neural Information Processing Systems, vol. 15, pp. 542–552. MIT Press, Cambridge, MA (2003)
137. Groot, P., Lucas, P., van den Bosch, P.: Multiple-step time series forecasting with sparse Gaussian processes. In: Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011), pp. 105–112. Ghent (2011)
138. Gutjahr, T., Ulmer, H., Ament, C.: Sparse Gaussian processes with uncertain inputs for multi-step ahead prediction. In: 16th IFAC Symposium on System Identification, pp. 107–112. Brussels, (2012)
139. Girard, A., Rasmussen, C., Murray-Smith, R.: Gaussian process priors with uncertain inputs: multiple-step ahead prediction. Technical report DCS TR-2002-119, University of Glasgow, Glasgow (2002)

140. Kocijan, J., Likar, B.: Gas-liquid separator modelling and simulation with Gaussian-process models. Simul. Modell. Pract. Theory **16**(8), 910–922 (2008)
141. Ažman, K., Kocijan, J.: Application of Gaussian processes for black-box modelling of biosystems. ISA Trans. **46**, 443–457 (2007)
142. Cho, J., Principe, J.C., Erdogmus, D., Motter, M.A.: Quasi-sliding model control strategy based on multiple-linear models. Neurocomputing **70**, 960–974 (2007)
143. Gauthier, J.P., Hammouri, H., Othman, S.: A simple observer for nonlinear systems applications to bioreactors. IEEE Trans. Autom. Control **6**, 875–880 (1992)
144. Lind, I.: Regressor selection in system identification using ANOVA. Licentiate thesis, University of Linköping, Linköping (2001)

http://www.springer.com/978-3-319-21020-9

Modelling and Control of Dynamic Systems Using Gaussian Process Models

Kocijan, J.

2016, XVI, 267 p. 117 illus., 17 illus. in color. With online files/update., Hardcover
ISBN: 978-3-319-21020-9