Projected Sequential Gaussian Processes: A C++ tool for interpolation of large data sets with heterogeneous noise☆

Remi Barillec[a,*], Ben Ingram[b], Dan Cornford[a], Lehel Csató[c]

[a]*Non-linearity and Complexity Research Group, Aston University, Birmingham, B4 7ET, UK*
[b]*Facultad de Ingeniería & Centro de Geomática, Universidad de Talca, Camino Los Niches Km. 1, Curicó, Chile*
[c]*Faculty of Mathematics and Informatics, Babeş-Bolyai University, Kogalniceanu 1, RO-400084, Cluj-Napoca Romania*

Abstract

Heterogeneous data sets arise naturally in most applications due to the use of a variety of sensors, and measuring platforms. Such data sets can be heterogeneous in terms of the error characteristics, and sensor models. Treating such data is most naturally accomplished using a Bayesian or model based geostatistical approach, however such methods generally scale rather badly with the size of data set, and require computationally expensive Monte Carlo based inference. Recently within the machine learning and spatial statistics communities many papers have explored the potential of reduced rank representations of the covariance matrix, often referred to as projected or fixed rank approaches. In such methods the covariance function of the posterior process is represented by a reduced rank approximation which is chosen such that there is minimal information loss. In this paper a sequential Bayesian framework for inference in such projected processes is presented. The observations are considered one at a time which avoids the need for high dimensional integrals typically required in a Bayesian approach. A C++ library, `gptk`, which is part of the INTAMAP web service, is introduced which implements projected, sequential estimation and adds several novel features. In particular the library includes the ability to use a generic observation operator, or sensor model, to permit data fusion. It is also possible to cope with a range of observation error characteristics, including non-Gaussian observation errors. Inference for the covariance parameters is explored, including the impact of the projected process approximation on likelihood profiles. We illustrate the projected sequential method in application to synthetic and real data sets. Limitations and extensions are discussed.

Keywords: low-rank approximations, sensor fusion, heterogeneous data

1. Introduction

Large, heterogeneous datasets are becoming more common (Cressie and Johannesson, 2008) due to our accelerating ability to collect data; whether it be from satellite-based sensors, aerial photography, large monitoring networks or large repositories of data accessible from Web sources or increasingly frequently a mixture of some or all of the above.

An issue faced when dealing with large global datasets is that of heterogeneities in the data collection process. In the context of a large monitoring network, the network infrastructure is often based on many existing smaller networks with potentially distinct data measurement mechanisms. Heterogeneity is also evident when exploiting dense aerial photography data as ancillary data to improve analysis of sparsely sampled data (Bourennane et al., 2006). Managing this heterogeneity requires specific modelling of the observation process, taking account that one

☆Code available from server at `http://code.google.com/p/gptk/downloads/list` or at `http://www.iamg.org/CGEditor/index.htm`.

*Corresponding author
Email address: `r.barillec@aston.ac.uk` (Remi Barillec)

is interested in the underlying *latent* (not directly observed) process, as is commonly done in the data assimilation setting (Kalnay, 2003). We assume that the underlying latent spatial process, $f = f(\mathbf{x})$, where $\mathbf{x}$ represents spatial location, is partially observed at points $\mathbf{x}_i$ by sensors:

$$y_i = h[f(\mathbf{x}_i)] + \varepsilon_i \,, \tag{1}$$

where $h[\cdot]$ represents the *sensor model* (also called the *observation operator* or *forward model*) that maps the latent state into the observables and ε represents the observation noise, in observation space. The observation noise represents the noise that arises from inaccuracies in the measurement equipment and also the representativity of the observation with respect to the underlying latent process at a point. The observation $y_i = y(\mathbf{x}_i)$ can be directly of the underlying latent spatial process (with noise), i.e. $y_i = f(\mathbf{x}_i) + \varepsilon_i$ where h is the identity mapping, but typically the sensor model might describe the observation process or more simply be a mechanism for representing sensor bias using a linear sensor model such as $y_i = [f(\mathbf{x}_i) + b] + \varepsilon_i$. The mapping f can be non-linear as is often the case with remote sensing, where f expresses the complex relationship between radiance or beam reflectivity and the latent process. Examples include, in hydrological applications, the power law between radar reflectivity and rainfall intensity (Collier, 1989; Marshall and Palmer, 1948), and the various relationships linking satellite radiance to temperature or vegetation types (Jensen, 2000). It is possible that each sensor has a unique sensor model, however it is more common for groups of sensors to have similar sensor models (for instance when the observations come from 2 different overlapping sensor networks, as is the case with precipitation fields estimated from both weather radar and rain gauge data). Note that it will often be the case that each sensor has its own estimate of uncertainty, for example where the observation results arise from processing raw data as seen in Boersma et al. (2004). The notation used in this paper is summarised in Table 1.

Symbol	Meaning
$\mathbf{x}$	location in 2D space
$f_i = f(\mathbf{x}_i)$	*latent* process at location $\mathbf{x}_i$
$y_i = y(\mathbf{x}_i)$	observation at location $\mathbf{x}_i$
$f = f_{1:n} = f(\mathbf{x}_{1:n})$	*latent* process at locations $\mathbf{x}_1, \ldots, \mathbf{x}_n$
$h(\cdot)$	*sensor model* or *observation operator*
ε	*observation error* following a given distribution
n	the total number of observations in the data set
m	the total number of locations included in the active set, $\mathcal{AS}$

Table 1: Notation used within the paper.

We assume in this work that the latent process is, *or can be well approximated by*, a Gaussian process over $\mathbf{x}$ (see Cressie, 1993; Rasmussen and Williams, 2006). A Gaussian process model is attractive because of its tractability and flexibility. We denote the Gaussian process prior by $p_0(f|\mathbf{x},\theta)$ where θ is a vector of *hyper-parameters* in the mean and covariance functions. In general we will assume a zero mean function, $\mu(\mathbf{x}) = 0$, although mean functions could be included, either directly or more conveniently by using non-stationary covariance functions (Rasmussen and Williams, 2006, section 2.7). The covariance function, $c(\mathbf{x},\mathbf{x}';\theta)$, is typically parametrised by θ in terms of length scale (*range*), process variance (*sill variance*) and *nugget* variance. A range of different valid covariance functions are available, each of which imparts certain properties to realisations of the prior Gaussian process. The posterior process is then given by:

$$p_{\text{post}}(f|y_{1:n},\mathbf{x},\theta,h) = \frac{p(y_{1:n}|f,\mathbf{x},h)p_0(f|\mathbf{x},\theta)}{\int p(y_{1:n}|f,\mathbf{x},h)p_0(f|\mathbf{x},\theta)df} \tag{2}$$

where $p(y_{1:n}|f,\mathbf{x},h)$ is the likelihood of the model, that is the probability of the n observations in the data set, $y_{1:n}$, given the model and the sensor model. The integral in the denominator is a

constant, often called the *evidence* or *marginal likelihood*, $p(y_{1:n}|\mathbf{x},\theta)$, which can be optimised to estimate the hyper-parameters, θ. The likelihood model is linked closely to the sensor model eq. (1). Assuming that the *errors* on the observations are independent, we can write $p(y_{1:n}|f,\mathbf{x}) = \prod_i p_i(y_i|f,\mathbf{x}_i,h_i)$ since the observations only depend on the latent process at their single location, and the errors are in the observation space. Note that if any of the errors, ε_i, are not Gaussian then the corresponding likelihood will yield a non-Gaussian process posterior. Also if the sensor model – h_i – is non-linear, even with Gaussian observation errors the corresponding posterior distribution will be non-Gaussian.

A key step in using Gaussian process methods is the estimation of θ given a set of observations, $y_{1:n}$. This is typically achieved by maximising the marginal likelihood, $p(y_{1:n}|\mathbf{x},\theta)$. Note that ideally we should seek to integrate out the hyper-parameters, however this is computationally challenging, particularly for the *range* parameters which require numerical integration. In this work we adopt a maximum (marginal) likelihood framework for parameter inference.

Large sizes of the datasets also present a major obstacle when likelihood-based spatial interpolation methods are applied. In terms of memory and processor computational requirements scale quadratically and cubically respectively. Typically, computation with no more than a few thousand observations can be achieved using standard likelihood based spatial interpolation models (Cressie, 1993) in reasonable time. The main computational bottleneck lies in the need to invert a covariance matrix of the order of the number of observations. For prediction with fixed hyper-parameters this is a one-off cost and can be approximated in many ways, for example using local prediction neighbourhoods which introduce discontinuities in the predicted surface. For hyper-parameter estimation matrices must be inverted many times during the optimisation process (Rasmussen and Williams, 2006, chapter 5). The most common solution to the hyper-parameter estimation problem adopted in geostatistics is to use a moment based estimator, the empirical variogram, and fit the parametric model to this (Journel and Huijbregts, 1978; Cressie, 1993), however these methods will not work when we assume that the field of interest, f, is not directly observed, but rather there exists a sensor model such that $y_i = h[f(\mathbf{x}_i)] + \varepsilon_i$. The use of such a latent process framework corresponds very closely to the model-based geostatistics outlined in Diggle and Ribeiro (2007).

Instead of employing a batch framework where all the observations are processed at once, in this work we consider a sequential algorithm to solve the previously mentioned challenges. There are other advantages that can be obtained by using sequential algorithms. Processing of the data can begin before the entire dataset has been collected. An example might be a satellite scanning data across the globe over a time period of a few hours. As the data is collected, the model is updated sequentially. This can also be of use in real-time mapping applications where data might arrive from a variety of communication systems all with different latencies. The sequential approach we employ is described in detail in Section 3. The main idea is that we sequentially approximate the posterior process, eq. (2), by the best approximating Gaussian process as we update the model one observation at a time. In order to control the computational complexity we have adopted a parametrisation for the approximate posterior process which admits a representation in terms of a reduced number of *active points*. Thus the name projected, sequential Gaussian process (`psgp`) for the algorithm.

We have chosen to implement `psgp` using C++ to ensure that program execution is efficient and can be optimised where required. The algorithm forms the core of our Gaussian Process Toolkit (`gptk`) library and is available for download from Google Code (Ingram et al., 2010). Installation instructions and a simple tutorial example are provided in Appendix C.

In addition, a web processing service interface to `psgp` can be accessed from the INTAMAP project web site[1] or installed from the CRAN R repository (Ingram and Barillec, 2009).

[1] `http://www.intamap.org`

We start in Section 2 by discussing large datasets and how these can be treated using a number of different approximations. In Section 3 we introduce the model parameterisation, present the sequential algorithm updates the parameterisation given observations and outline hyper-parameter estimation. An overview and discussion of design principles used in our implementation is presented in Section 4. Section 5 shows how the software can be applied to synthetic and real-world data. Section 6 contains a discussion of the algorithm performance and gives conclusions. The paper builds upon Cornford et al. (2005) and Csató and Opper (2002) by allowing more flexible approaches to sensor models and dealing in more detail with algorithmic and implementation details and practical deployment of the new C++ `gptk` library.

2. Interpolation for large data sets

Techniques for treating large-scale spatial datasets can generally be organised into categories depending whether they utilise sparsely populated covariance matrices (Furrer et al., 2006), spectral methods (Fuentes, 2002) or low-rank covariance matrix approximations (Snelson and Ghahramani, 2006; Lawrence et al., 2003; Cressie and Johannesson, 2008; Banerjee et al., 2008). Traditional geostatistics utilises a moving-window approach where a prediction at a given location is calculated from a predefined number of observations rather than utilising the entire dataset (Haas, 1990). In this case, instead of solving one large equation, many smaller systems of equations are solved instead. Each of the described techniques has associated advantages and disadvantages; here we consider low-rank covariance matrix approximations.

A common technique for treating large-datasets is simply to sub-sample the dataset and use only a smaller number of observations during analysis. This is probably the most naïve low-rank covariance matrix approximation. All but the smaller subset of observations are discarded. Choosing the subset of observations that are to be retained can be complex. In Lawrence et al. (2003) the algorithm runs for a number of cycles sequentially inserting and removing observations determined by the magnitude of the reduction of uncertainty in the model. In this way a good subset of observations is chosen to represent the posterior process, although observations not within this subset will have no impact on the posterior, other than through the selection process.

Csató and Opper (2002) present a similar approach, which is related to geostatistical theory in Cornford et al. (2005). Instead of discarding some observations, it is suggested that the effect of the discarded observations can be projected onto the low-rank covariance matrix in a sequential manner identifying the most informative observations in the process. It is shown how this can be done in such a way so as to minimise loss of information. Related to this is the technique of Snelson and Ghahramani (2006) where, instead of a sequential projection scheme, the entire dataset is projected on to a set of *active points* in a batch framework, a procedure that is complex. (Banerjee et al., 2008) present a similar low-rank approximation, but apply full Bayesian inference rather than just Bayesian prediction. The *active points* are selected *a priori*, hence small-scale features in the dataset can not be always be captured by the model parameterisation if the insufficient active points are selected. Cressie and Johannesson (2008) present an alternative where the Frobenius norm between the full and an approximate covariance matrices is minimised using a closed expression.

The approach of Csató and Opper (2002), on which the `psgp` implementation is based, has a number of advantages. First, since it is a sequential algorithm (Opper, 1996), the complexity of the model, or the rank of the low-rank matrix can be determined during runtime. In scenarios where the complexity of the dataset is unknown *a priori* this is advantageous. Secondly, in a practical setting, it is not uncommon that observations arrive for processing sequential in time. Thirdly, by processing the data in a sequential fashion, non-Gaussian likelihoods, $p_i(y_i|f,\mathbf{x}_i,h_i)$, can be used (Cornford et al., 2005). The projection step mentioned earlier is employed to project

from a potentially non-Gaussian posterior distribution to the nearest Gaussian posterior distribution, again minimising the induced error in doing so. There are two ways in which the update coefficients, necessary for the projection, can be calculated: analytic methods requiring the first and second derivatives of the likelihood function with respect to the model parameters and by population Monte Carlo sampling (Cappé et al., 2004).

The inclusion of this population Monte Carlo sampling algorithm is a novel feature of this implementation. The likelihood function for each observation can be described without having to calculate complex derivatives. Furthermore, our sampling based method enables sensor models, $h[\cdot]$ to be included, permitting data fusion. At present a basic XML parser is implemented[2] which means that the `gptk` library can process sensor models described in MathML, as might be typical in a SensorML description in an Observations and Measurements document part of the Sensor Web Enablement suite of standards proposed by the Cox (2007); Open Geospatial Consortium (2007).

Non-stationarity is a very common feature of geospatial data. The parametrisation of Cressie and Johannesson (2008) includes a non-stationary covariance function (defined in terms of basis function). The `psgp` implementation also supports non-stationary kernels, although the only such kernel currently implemented is the Neural Network kernel (Williams, 1998).

3. The projected sequential Gaussian process model

The `psgp` algorithm relies essentially on two key features:

- a recursive *parametrisation* of the posterior update, which allows the posterior process to be updated sequentially, by considering one observation at time,
- the exploitation of potential redundancy in the data by projecting the observations onto an *active* subset of fixed size,

which combined together lead to the low-rank, recursive parametrisation in Equations (3-5).

The *parametrisation* of the Gaussian process approximation to the posterior distribution from eq. (2) is achieved by expressing the mean function $\mu(\mathbf{x})$ and covariance kernel $c(\mathbf{x},\mathbf{x}')$ as functions of a low-rank vector $\vec{\alpha}$ and a low-rank matrix $\vec{C}$ as:

$$\mu_{\text{post}}(\mathbf{x}) = \mu_0(\mathbf{x}) + \sum_{i\in\mathcal{AS}} \alpha_i c_0(\mathbf{x},\mathbf{x}_i), \tag{3}$$

$$c_{\text{post}}(\mathbf{x},\mathbf{x}') = c_0(\mathbf{x},\mathbf{x}') + \sum_{i,j\in\mathcal{AS}} c_0(\mathbf{x},\mathbf{x}_i) C(i,j) c_0(\mathbf{x}_j,\mathbf{x}') \tag{4}$$

where $\mu_0(\mathbf{x}) = 0$ and $c_0(\mathbf{x},\mathbf{x}')$ are the prior mean and covariance functions respectively. The low-rank approximation allows efficient computation due to the selection of an *active set* $\mathcal{AS} = \{\mathbf{x}_1,\ldots,\mathbf{x}_m\}$ of m locations where typically $m \ll n$. That is, $\vec{\alpha}$ is a vector of length m and $\vec{C}$ is an $m \times m$ matrix. Since `psgp` computation scales cubically with the size of the *active set*, m should be chosen as small as possible, however there is clearly a trade-off, which is discussed later. Computation of the parameters of the posterior parametrisation (the vector $\vec{\alpha}$ and the matrix $\vec{C}$) is a challenging task for a number of reasons. The parametrisation is an approximation that is based on the *active set*. Optimal *active set* locations depend on covariance function model parameters hence as these are re-estimated the *active set* must be reselected. The algorithm runs for a number of cycles and the processes of *active set* selection and covariance model parameter estimation are interleaved obtaining improved estimates during each cycle.

[2] The XML parser did not make it to the latest version of the library, as it needs further documenting and introductory examples. We are hoping to release it as part of the next version.

Algorithm 1 `psgp` algorithm outline for fixed θ.

Initialise model parameters $(\vec{\alpha}, \vec{C})$, $(\vec{a}, \vec{\Lambda}, \vec{P})$ to empty values.
for epIter = 1 to Data Recycling Iterations **do**
    $t \leftarrow 1$
    Randomise the order of location/observation pairs.
    while t < numObservations **do**
        Get next location and observation $(\mathbf{x}_t, y_t)$.
        if epIter > 1 **then** *// Observation already processed*
            Remove contribution of current observation $(\vec{\tilde{\alpha}}, \vec{\tilde{C}})$
        end if
        if Likelihood update is analytic **then** *// Compute update coefficients*
            Compute (q_+, r_+)
              from first and second derivatives of the marginal likelihood
        else
            Use population Monte Carlo sampling for computing (q_+, r_+)
        end if
        Calculate γ, *// $\gamma > 0$ is a heuristic measure; the difference between*
                // the current GP and that from the previous iteration
        if $\gamma > \varepsilon$ **then** *// Using full update of the parameters*
            Add location $\mathbf{x}_t$ to the *Active Set*
            Increase the size of $(\vec{\tilde{\alpha}}, \vec{\tilde{C}})$
            Using (q_+, r_+), update $(\vec{\tilde{\alpha}}, \vec{\tilde{C}})$ and EP parameters $(\vec{a}, \vec{\Lambda}, \vec{P})$;
        else *// Model parameters updated by a projection step*
            Project observation effect (q_+, r_+) onto the current *Active Set*
            Update $(\vec{\tilde{\alpha}}, \vec{\tilde{C}})$ and the EP parameter matrices $(\vec{a}, \vec{\Lambda}, \vec{P})$
        end if
        if Size of active set > Maximum Desired Size **then**
            Calculate the informativeness for each Active Set element
            Find the least informative element, delete it from the active set
            Update $(\vec{\tilde{\alpha}}, \vec{\tilde{C}})$ and EP parameters $(\vec{a}, \vec{\Lambda}, \vec{P})$
        end if
        $t \leftarrow t + 1$
        $(\vec{\alpha}, \vec{C}) \leftarrow (\vec{\tilde{\alpha}}, \vec{\tilde{C}})$
    end while
end for

Using the parametrisation from (3) and (4), the update rules are written as:

$$\begin{aligned}\vec{\alpha}_{t+1} &= \vec{\alpha}_t + q_+ \, \vec{s}_{t+1} \\ \vec{C}_{t+1} &= \vec{C}_t + r_+ \, \vec{s}_{t+1}\vec{s}_{t+1}^T\end{aligned} \qquad \text{with} \qquad \vec{s}_{t+1} = \vec{C}_t\vec{c}_{t+1} + \vec{e}_{t+1} \tag{5}$$

where $\vec{c}_{t+1} = [c_0(\mathbf{x}_{t+1}, \mathbf{x}_1), \ldots, c_0(\mathbf{x}_{t+1}, \mathbf{x}_t)]^T$ is the prior covariance between the location being considered at iteration $t+1$ and all current points in the active set. $\vec{e}_{t+1} = [0, 0, \ldots, 1]^T$ is the $t+1$-th unit vector that extends the size of $\vec{\alpha}_{t+1}$ and $\vec{C}_{t+1}$ by one. The update coefficients q_+ and r_+ come from the parametrisation of the posterior mean function and covariance function updates given a new observation y_i at location $\mathbf{x}_i$. They are defined as the first and second derivatives, with respect to the posterior mean at $\mathbf{x}_i$, of the log-evidence. That is:

$$q_+ = \frac{\partial}{\partial \mu(\mathbf{x}_i)} \ln \int p(f_i)\, p(y_i|f_i)\, df_i \tag{6}$$

$$r_+ = \frac{\partial^2}{\partial \mu(\mathbf{x}_i)^2} \ln \int p(f_i)\, p(y_i|f_i)\, df_i. \tag{7}$$

where $f_i = f(\mathbf{x}_i)$ is the posterior process at $\mathbf{x}_+$. $p(f_i)$ denotes the posterior process (prior to the update) with mean $\mu(\mathbf{x}_i)$. The derivations leading to this particular parametrisation is reproduced, for the interested reader, in Appendix A. The full detail can be found in (Csató and Opper, 2002; Csató, 2002).

The consequence of the update rule is that: (1) all locations need to be processed since all of them contribute to the resulting mean and covariance functions, and (2) the contribution of each observation is retained in the posterior parametrisation.

The benefits of the parametrisation and the sequential nature of the updates are two-fold. First, the update coefficients (q_+ and r_+) can be computed for a variety of user-defined likelihood models, $p_i(y_i|f, \mathbf{x}_i, h_i)$, as shown in Section 3.1 and secondly, the update coefficients can be computed in a manner which does not require that the model complexity to be increased, as shown in Section 3.2.

3.1. Computing the update coefficients

The update coefficients, q_+ and r_+, can be computed in two main ways. Essentially these update coefficients can be seen as computing the first two moments of the updated Gaussian approximation at $\mathbf{x}_{t+1}$, and thus require the evaluation of integrals. In Csató and Opper (2002) analytic formulations for q_+ and r_+ are given for Gaussian, Laplace and one sided exponential noise models. Within the `gptk` package at present only the Gaussian noise model is implemented analytically. In this paper we extend the updates to include a sampling based strategy, since the presence of a non-linear sensor model, h, will almost certainly make analytic updates impossible.

The update step is iterated several times, within an inner loop to process all observations, and within an outer loop to recycle over data orderings, as shown in Algorithm 1. It is thus important that the computation of q_+ and r_+ is as efficient as possible. The sequential nature of the approximation means that the required integrals are low dimensional, and for scalar observations, one dimensional. When analytic computation of q_+ and r_+ is not possible, one can resort to alternative sampling methods. Several methods might be envisaged, ranging from quadrature to Markov chain Monte Carlo methods. The selection of Population Monte Carlo (PMC) sampling (Cappé et al., 2004) was based on its simplicity and efficiency. PMC is a sequential importance sampling technique, where samples from one iteration are used to improve the importance sampling, or proposal, distribution at the subsequent steps in the algorithm. The technique is known to be efficient and we have found it to be stable too.

The basic implementation is summarised in Algorithm 2. The aim is to evaluate the expectation of the likelihood $p(y_i|f_i)$ with respect to the current (in the sequential update) posterior $p(f_i|\mathbf{x}_i)$. The importance sampling method consists in generating a sample from a more tractable

Algorithm 2 Population Monte Carlo algorithm used in `psgp` (for a single observation)

1: Inputs:
2: - a location $\mathbf{x}_i$ with predictive distribution from the psgp: $p(f_i|\mathbf{x}_i) = N(\mu_i, \sigma_i^2)$
3: - an observation y_i with likelihood $p_i(y_i|f_i, \mathbf{x}_i, h_i)$
4: Initialise the proposal distribution $\hat{p}(f) = N(\mu_*, \sigma_*^2)$ with $\mu_* = \mu_i$ and inflated variance $\sigma_*^2 = s\sigma_i^2$
5: **for** pmcIter = 1 to PMC Iterations **do**
6: **for** j = 1 to Number of PMC samples **do**
7: Sample f_j from $\hat{p}(f)$
8: Set $w_j = p_i(y_i|f_j, \mathbf{x}_i, h_i) \times p(f_j|\mathbf{x}_i) \;/\; \hat{p}(f_j)$ *// Compute the importance weights*
9: **end for**
10: Normalise each importance weight $w_j = w_j / \sum_k w_k$
11: $\mu_* = \sum_j w_j f_j$
12: $\sigma_*^2 = \sum_j w_j f_j^2 - \mu_*^2$
13: $\hat{p}(f) \leftarrow N(\mu_*, \sigma_*^2)$ *// Update the proposal distribution*
14: **end for**
15: $q_+ = (\mu_* - \mu_i)/\sigma_i^2$ *// Update q and r using the final values of μ_* and σ_*^2*
16: $r_+ = (\sigma_*^2 - \sigma_i^2)/(\sigma_i^2)^2$

distribution (lines 4 and 7), called the proposal distribution, and weight each sample according to the true distribution. The weight also incorporates the function we are computing the expectation of, in this case the likelihood. The proposal distribution is then updated using the Monte-Carlo empirical estimates of the mean and variance μ_* and σ_*^2, and the algorithm is repeated. Few iterations are typically needed for convergence of the algorithm.

The main benefit of this approach is that one only needs to evaluate the sensor model forward to compute the likelihood of the observation $p(y_i|f, \mathbf{x}_i, h_i)$ and thus we can supply h as a MathML object to be evaluated at run-time without having to recompile the code for new sensor models. In practice only two iterations of PMC, each using 100 samples, are required to get excellent approximation of the first and second moments, and the initial variance inflation factor, s is empirically chosen to be 4.

3.2. Sparse extension to the sequential updates

Using purely the sequential update rules from eq. (5), the parameter space explodes: for a dataset of size n there are $O(n^2)$ parameters to be estimated. Sparsity within parametrised stochastic processes is achieved using the sequential updates as above and then projecting the resulting approximation to remove the least informative location from the active set, $\mathcal{AS}$; this elimination is performed with a minimum loss in the information-theoretic sense (Cover and Thomas, 1991).

In order to have minimum information loss, one needs to replace the unit vector $\vec{e}_{t+1}$ from eq. (5) with its projection $\vec{\pi}_{t+1}$ to the subspace determined by the elements in the *active set*: $\vec{\pi}_{t+1} = K_{\mathcal{AS}}^{-1}\, \vec{k}_{t+1}$ – see Csató and Opper (2002) for details. An important extension of the result above is that it is possible *to remove an element* from the active set. We proceed as follows: consider an element from the active set denoted as $\mathbf{x}_*$. Assume that the current observation was the last added via the sequential updates from eq. (5). Reconstruct the vector $\vec{k}_* = [c_0(\mathbf{x}_*, \mathbf{x}_1), \ldots, c_0(\mathbf{x}_*, \mathbf{x}_t)]^T$ and obtain the *virtual* update coefficients q_* and r_* using either analytic or sampling based methods. Once these coefficients are known, the sparse update is given by substituting $\vec{\pi}_* = K_{\mathcal{AS}}^{-1}\, \vec{k}_*$, where $\mathcal{AS}$ is the reduced active set in eq. (5).

3.3. Expectation-propagation

The sequential algorithm, as presented above, allows a *single iteration* over the data-set. There are however situations when one would like to re-use the data and further refine the resulting approximation algorithm. In particular for non-Gaussian errors and non-linear sensor models the sequence in which the observations were included could be important, and thus it would be useful to process the observations in a random order several times to minimise the dependence on the initial data ordering.

The *expectation-propagation* – or EP – framework of Minka (2001) allows re-use of the data. The methodology employed within the EP framework is similar to the one presented in the sparse updates subsection: for each observation – assuming one has the likelihood and observation models – we store the update coefficients (q_+, r_+) for each location $\mathbf{x}_i$, which we denote (q_i, r_i). Since (q_i, r_i) are updates to the first and second moments of the approximated posterior, we can represent the approximation as a *local Gaussian* and have a second parametrisation in the form of $\exp\left[-\lambda_i(\pi_i^T f_{\mathcal{AS}} - a_i)^2/2\right]$ where $f_{\mathcal{AS}}$ is the vector of jointly Gaussian random variables over the active set. To sum up, we have a second *local representation* to the posterior process with parameter vectors $\vec{a}$ and $\vec{\Lambda}$ of size $n \times 1$ the 'projection'matrix $\vec{P} = [\pi_1, \ldots, \pi_n]$, which is of size $m \times n$ with m the size of the active set as before.

Thus with a slight increase in memory requirement we can re-use the data set to obtain a better approximation to the posterior. In several practical application this improvement is essential, specially when mean and covariance function hyper-parameters, θ, need to be optimised for the data set: it can also provide improved stability in the algorithm Minka (2001).

3.4. The complete `psgp` algorithm

A complete inference experiment using the `psgp` library typically consists of the following steps (line numbers refer to the example in Algorithm 3):

1. Initialisation
    - Implement (if needed) the observation operator (or supply the MathML expression)
    - Create a covariance function object with some initial parameters (line 3)
    - Create a likelihood object for the observations (line 4). This can either be a single likelihood model for all observations, or a vector of likelihood models with an associated vector of indexes indicating which model to use for each observation. If an observation operator is used, it is passed to the relevant likelihood models.
    - Initialise the `psgp` object by passing in the locations ($\mathbf{x}$), observations (y) and covariance function (line 6). Optionally, the maximum number of active points and number of cycles through the observations can also be specified.
    - Compute the posterior model for the current (initial) parameters (line 8, detail in Algorithm 1). Alternately, if several likelihood models are needed (e.g. our observations come from different sensors), one can use the overloaded form `psgp.computePosterior(modelIndex, modelVector)`, where `modelVector` is a vector of likelihood models and `modelIndex` is a vector containing, for each observation, the index of the corresponding likelihood model.
2. Parameter estimation
    - In order to estimate the parameters, a model trainer object must be chosen and instantiated (line 9). Custom options can be set, such as the maximum number of iterations or whether to use a finite difference approximation to the gradient of the objective function instead of its analytical counterpart.

Algorithm 3 Full `psgp` algorithm in pseudocode form.

```
Require: double range, sill, mu, sigma2;          // Initial parameters, initialised previously
Require: double (*h)(double);          // Pointer to observation operator function, declared elsewhere

 1: ExponentialCF cf(range, sill);  // Create a covariance function object with initial hyper-parameters

 2: GaussianSampLikelihood lh(mu, sigma2, &h)        // Initialise the likelihood model, passing in
    parameters
 3:                                                  // and, optionally, the observation operator

 4: psgp psgp(X, y, cf, 400);              // Initialise a psgp object given the locations, observations,
 5:                                        // covariance function and a limit of 400 active points

 6: psgp.computePosterior(lh);                // Compute the posterior under the likelihood model

 7: SCGModelTrainer opt(psgp);              // Initialise optimisation object and link to psgp object
 8: for hypIter = 1 to Hyper-Parameter Estimation Iterations do   // Optimise the hyper-parameters
 9:     opt.train(5);                                             // Perform 5 optimisation steps
10:     psgp.computePosterior(lh);          // Recompute posterior for updated hyper-parameters
11: end for

12: psgp.makePrediction(meanPred, varPred, Xpred);      // Predict at a set of new locations Xpred
```

- The parameter estimation consists of an inner loop (line 11), in which optimal parameters are sought which minimise the objective function. Because the objective function, optimal hyper-parameter values and active set are *all interdependent*, it is important to take small optimisation steps and re-estimate the active set as we go – this can be thought of as an Expectation-Maximisation algorithm. Typically, the parameter optimisation (line 11) and update of the active set (line 12) alternate in an outer loop (lines 10-13), only a few iterations of which are usually needed for convergence.

3. Prediction

   - Once the hyper-parameters, θ, and algorithm parameters, $(\vec{\alpha}, \vec{C})$, have been estimated, the `psgp` object is able to make predictions at a set of new locations (line 14). The predictive mean and variance at these locations is returned. An optional covariance function can also be used for prediction. This is useful if the covariance function includes a nugget term but noise-free predictions are wanted: we can pass in and use the covariance function without the nugget term.

Algorithm 3 provides a code sample illustrating the above procedure.

4. Design of C++ library

The algorithm discussed in this paper is one of a number of Gaussian process algorithms that we intend to develop, hence we emphasise the importance of having a flexible and extensible framework. We have chosen C++ as the implementation language as this allows us to produce both fast and portable code. We utilise the Matlab like IT++ (Ottosson et al., 2009) library which itself uses the highly optimised vector/matrix operations available from BLAS (Lawson et al., 1979; Blackford et al., 2002) and LAPACK (Anderson et al., 1999).

4.1. Covariance functions

The base functionality of any Gaussian process algorithm is the need to calculate covariance matrices given different covariance functions. We provide a base `CovarianceFunction` abstract class which all covariance functions must implement. The abstract class defines a number of typical operations such as computing the covariance between two vectors of locations. Additionally, where available, the functions for calculating gradients of the covariance functions with respect to their parameters are implemented.

We have implemented a basic number of covariance functions such as the common `ExponentialCF`, `GaussianCF`, `Matern3CF`, `Matern5CF` (these are Matern covariance functions with roughness parameters 3/2 and 5/2 respectively). A `NeuralNetCF` is also available, which implements a Neural Network kernel (Rasmussen and Williams, 2006). Bias and nugget terms are implemented as `ConstantCF` and `WhiteNoiseCF` respectively. Further combinations of these covariance functions can be designed using the `SumCF` class. All the above provide an analytic implementation of the gradient with respect to the hyper-parameters, θ.

4.2. Likelihood models

A `LikelihoodType` interface is implemented by and subclassed as either a `Analytic Likelihood` or a `SamplingLikelihood`. These classes are further extended to define specific likelihood types. Likelihood models implementing the `AnalyticLikelihood` must provide the first and second derivative information used in the update via q_+ and r_+. However, for non-Gaussian likelihood models or indirect observations through a non-linear h, exact expressions of these derivatives are often not available. Classes implementing the `SamplingLikelihood` interface provide an alternative whereby the population Monte-Carlo algorithm discussed in Section 3.1 is used to infer this derivative information. These allow the use of an observation operator (optional) and non-Gaussian noise models (such as the `ExponentialSampLikelihood`). When observations have different noise characteristics, a combination of likelihood models can be used.

4.3. Optimisation of hyper-parameters

Optimal parameters for the covariance function (hyper-parameters) can be estimated from the data through minimisation of a suitable error function. To that effect, the Gaussian process algorithms (including `psgp`) must implement an `Optimisable` interface, which provides an `objective` function and its `gradient`. Typically, one chooses the objective function to be the *evidence* or *marginal likelihood* of the data (Rasmussen and Williams, 2006). Because computing the full evidence is expensive, alternate options are implemented by `psgp`: the first is an approximation to the full evidence, whereby only the active observations are taken into account, and the second is an upper-bound to the evidence which takes into account the projection of all observations onto the active set.

In order to minimise the objective function, several gradient-based local optimisers are included in the package. The optimisers are linked to an `Optimisable` object and minimise its `objective` function. General options for optimisation can be specified such as number of iterations, parameter tolerance and objective function tolerance. One also has the option to use a finite difference approximation to the gradient rather than the analytical one. This is usually slower but can be useful for testing purposes.

More generally, the Gaussian process algorithms also implement a `ForwardModel` interface which ensures the communication of parameters remains consistent, through appropriate accessors and modifiers methods.

4.4. Miscellaneous tasks

Aside of the core `psgp` classes, the `gptk` library also provides basic support for other secondary tasks. These include import/export of the data from/to CSV (comma separated values)

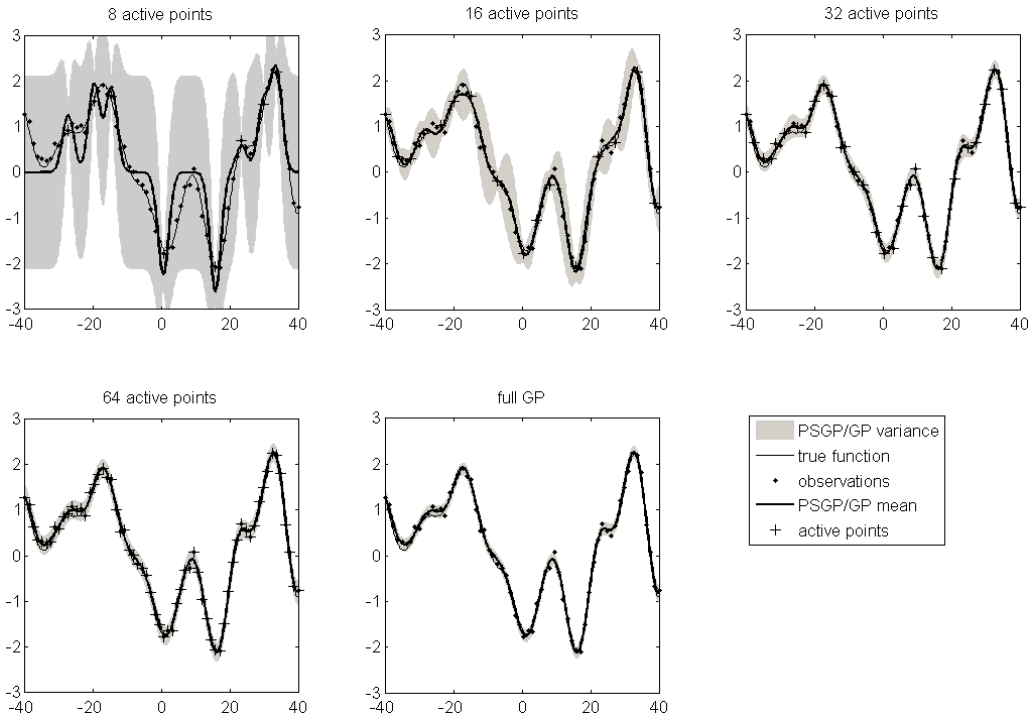

Figure 1: Quality of the `psgp` approximation for different active set sizes.

files, a widely supported matrix file format, through the `csvstream` class. Basic 1D plotting can also be achieved (in Linux) using the `GraphPlotter` utility, based on GNUPlot (Williams et al., 2009).

The software uses a number of default settings which are described in the user manual and documentation. However, the default parameters may not always be appropriate for all applications. Functions for setting and getting particular parameter values are included.

5. Examples and use

To illustrate the `psgp` algorithm we consider several examples ranging from simple, illustrative cases to the more complex scenario of spatial interpolation of radiation data.

5.1. Illustration of the algorithm

To illustrate the impact of projecting the observations onto the set of active points, we look at the quality of the `psgp` approximation for increasing sizes of the active set. Results are shown on Figure 1. The latent function is sampled from a known Gaussian process (thin solid line). 64 observations are taken at uniformly spaced locations. These are projected onto an active set comprising of 8, 16, 32 and the 64 observations (from left to right, and top to bottom). The mean (thick solid line) of the `psgp` model posterior and 2 standard deviations (grey shading) are shown along with the active points (black dots) and observations (grey crosses). The full Gaussian process using the 64 observations is shown on the bottom right plot for comparison.

When using 64 active observations, the `psgp` algorithm performs, as one would expect, as the full Gaussian process, yielding the same predictive mean and variance. When taking the number of active points to 32 (half the data), the `psgp` still provides a robust approximation to the true posterior process, although the variance is slightly larger due to the fact that the approximation induces a small increase in uncertainty. With 16 active points (a quarter of the data), the mean process is still captured accurately, however the uncertainty grows faster between active points, which relates to the covariance estimation which needs to ascribe longer ranges to cover the

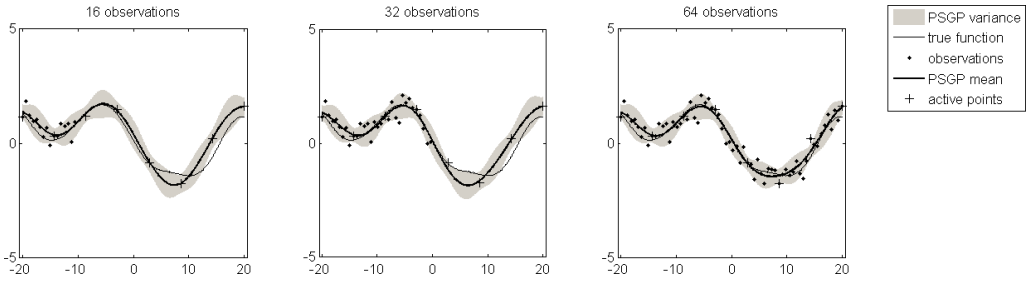

Figure 2: Evolution of the psgp approximation as observations are processed

domain. Retaining only 8 active points leaves only a rough, although still statistically valid, approximation to the underlying process. This simple experiment clearly shows the trade-off between the size of the active set and the accuracy of the approximation, although we stress that poor approximations also have appropriately inflated uncertainty.

The sequential nature of the psgp algorithm is illustrated on Figure 2. For a fixed number of active points, the posterior approximation is plotted (from left to right) after 16, 32 and the 64 observations (grey crosses) have been projected onto the active set (black dots). Observations are, in this example, presented sequentially from left to right rather than randomly, for illustration purposes. The effect is mainly seen on the mean process, showing that the observations not retained in the active set are still taken into account in the posterior process. This is most noticeable in the right hand part of the last two plots (32 to 64 observations), where the quality of the fit is greatly improved by projecting the effect of the observations in that region onto the 8 active points.

5.2. Heterogeneous, non-Gaussian observation noise

When the errors associated with observations are non-Gaussian and/or have varying magnitudes, we can include this knowledge in our model. Figure 3 shows a somewhat contrived illustration of why utilising additional knowledge about a dataset is crucial to obtaining good results during prediction. Additive noise of three types is applied to observations of a simple underlying function. 210 uniformly spaced observations are corrupted with:

- moderate Gaussian white noise in the range [0,70],
- Exponential (positive) noise in the range [71,140],
- low Gaussian white noise in the range [141, 210].

Furthermore, in the region [141, 210], the function is observed through a non-linear observation operator $h[f(x)] = \frac{2}{27}x^3 - 2$. The importance of using a correct combination of likelihood models is illustrated by comparing a standard Gaussian process (left plot) using a single Gaussian likelihood model with variance set to the empirical noise variance, and the psgp (right plot) using the correct likelihood models.

In the centre, the GP overestimates the true function due to the positive nature of the observation noise, while in the rightmost region, it underestimates the mean since it is treating the observations as direct measurements of the underlying function. Using the correct likelihood models and an active set of 50 observations, the psgp is able to capture the true function more accurately, as would be expected.

5.3. Parameter estimation

Figures 4 and 5 show the marginal likelihood profiles for the upper bound of the evidence (used to infer optimal hyper-parameters) for Example 5.1 and Example 5.2 respectively. These

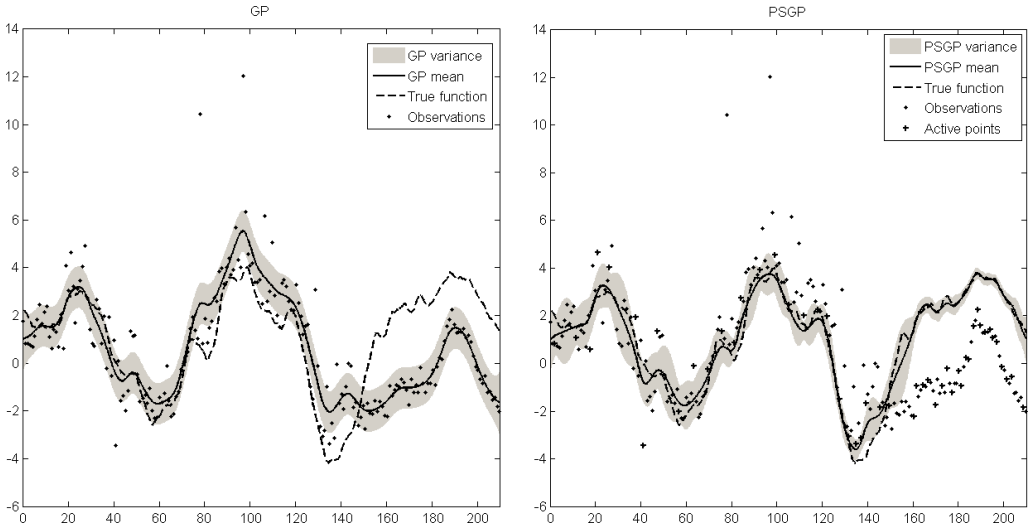

Figure 3: Example with heterogeneous observation noise.

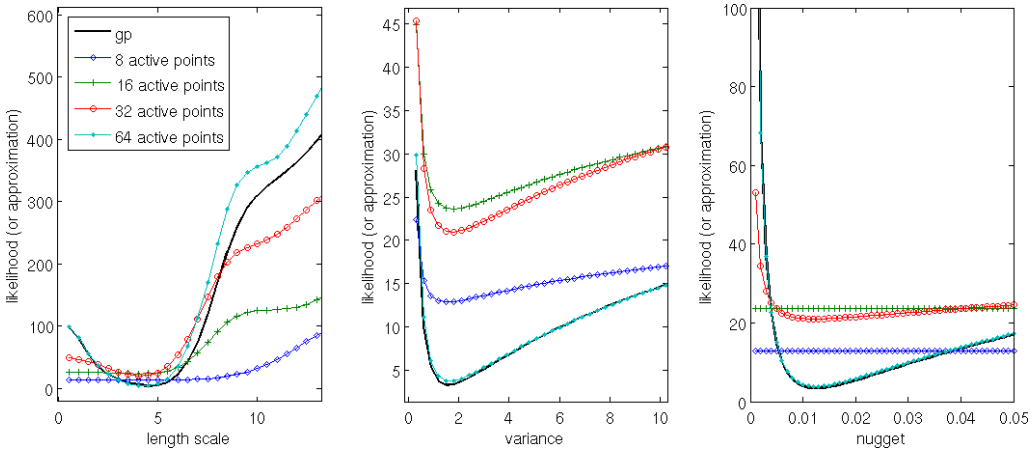

Figure 4: Marginal likelihood profiles (upper bound approximation) for Example 5.1.

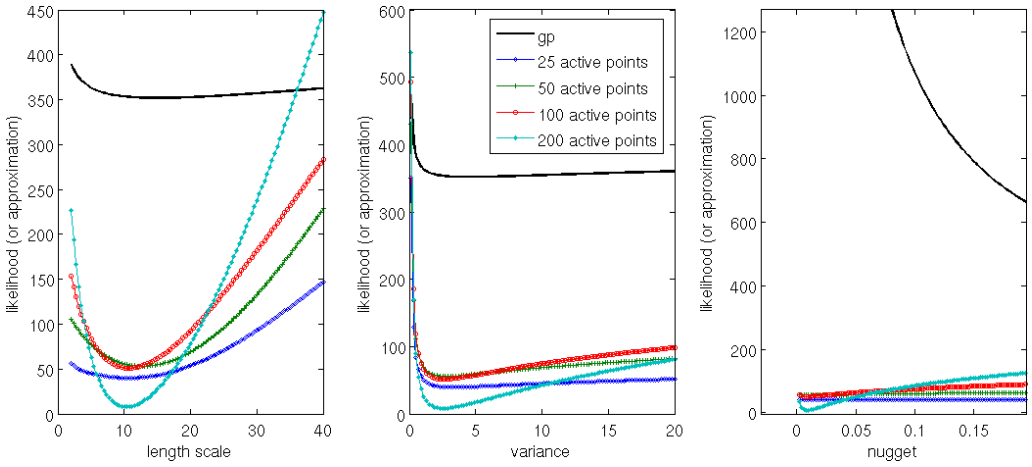

Figure 5: Marginal likelihood profiles (upper bound approximation) for Example 5.2.

profiles show the objective function as a single hyper-parameter is varied while keeping the others fixed to their most likely values. The profiles are shown, from left to right, for the length scale (or range), the sill variance and the nugget variance. The objective function is evaluated for active sets of size 8 (diamonds), 16 (crosses), 32 (circles) and 64 (dots). For comparison, the full evidence, as given by a full Gaussian process (using the 64 observations) is plotted as a thick black line.

For Example 5.1 (Figure 4), we observe in the region where the optimal hyper-parameters lie (corresponding to the minimum of the objective function) that the upper bound approximation to the true evidence (solid black line) provides a reasonable, but faster, substitute, as long as enough active points are retained. However, for low numbers of active points (8 or 16), the evidence with respect to the range and nugget terms is poorly approximated. In such cases, it is likely that the "optimal" range and nugget values will either be too large or too small depending on initialisation, resulting in unrealistic correlations and/or uncertainty in the posterior process. This highlights the importance of a retaining a large enough enough active set, the size of which will typically depend on the roughness of the underlying process, with respect to the typical sampling density.

In the case of complex observation noise (Example 5.2, Figure 5), we can see that the profiles of the Gaussian process (GP) evidence and the `psgp` upper bound are very dissimilar, due to the different likelihood models used. The evidence for the GP reflects the inappropriate likelihood model by showing little confidence in the data (high optimal nugget value - not shown on the right hand plot due to a scaling issue) and difficulty in choosing an optimal set of parameters (flat profiles for the length scale and variance). Profiles for the `psgp` show much steeper minima for the range and variance, while keeping the nugget term very small (high confidence in the data). Again, this confirms that using a complex likelihood model is critical for extracting maximum information from indirect observations or observations with non-Gaussian noise.

5.4. Application to spatial interpolation

5.4.1. Experiment setup

The third example shows an application of the `psgp` method to the interpolation of spatial observations. The dataset is taken from the INTAMAP case studies[3]. The data consists of spatial measurements of gamma dose rate from the EURDEP monitoring network (Stöhlker et al., 2009), spread across Europe. We restrict the domain to Germany only, where the coverage is the densest and most uniform. Out of a total of 2016 observations, 403 are kept aside for validation and a `psgp` is fitted to the remaining ones in order to infer rates in regions where no data is available.

The `psgp` configuration used for that example includes the following settings:

- the covariance function chosen is a mixture of exponential, white noise and bias covariance functions,
- the size of the active set is limited to 400 active points,
- hyper-parameters are estimated with a direct approximation to the evidence (based on the active set only).

To benchmark `psgp`, we run the same experiment using the popular `gstat` package (Pebesma and Wesseling, 1998) to perform simple kriging with a moving-window. Rudimentary analysis of the empirical variogram suggests that a spherical covariance is an appropriate model for the data. The spherical variogram model was fitted using ordinary least-squares to obtain the model parameters. These estimated parameters were then used to perform the kriging.

[3] `http://www.intamap.org/sampleRadiation.php`

5.4.2. Model validation

To validate the model, we look at several diagnostics. Figure 6 shows a scatter plot of the predicted values at the 403 unobserved locations against the actual values, for `psgp` (left) and `gstat` (right). The parameters for both methods are shown in Table 2, while Table 3 shows the Mahalanobis distance and Root Mean Square error for the predicted data, in both cases. The theoretical expected value for the Mahalanobis distance is 403 (the number of predictions).

	Range	Sill	Nugget
PSGP (exponential kernel)	25.08	7511.18	97.22
GSTAT (spherical variogram)	5.63	457.98	102. 12

Table 2: Estimated parameters of PSGP and GSTAT

There is a very good agreement between both methods' mean predictions, as indicated by the almost identical Root Mean Square Error and the very similar scatter plots (the distribution of points is again almost identical). The main difference lies in the estimation of uncertainty associated with the prediction. `psgp` is over-confident in its prediction, as indicated by the large range parameter and confirmed by the inflated Mahalanobis distance. `gstat` shows a very good estimation of the uncertainty in terms of Mahalanobis distance. `psgp` is able to do as well as `gstat` in terms of mean prediction, in spite of using an active set smaller by a factor of 4. However, this comes at the expense of quantifying the uncertainty. The overconfidence of `psgp` could be due to the approximation error not being accounted for properly. Further investigation is required to investigate the phenomenon more in detail and determine whether this is a characteristic of this particular dataset or a limitation of the method itself.

	Root Mean Square Error	Mahalanobis distance
PSGP (exponential kernel)	11.69	1,164
GSTATS (spherical variogram)	11.63	419

Table 3: Diagnostics on the validation set

Regarding computation times, it takes a few minutes on a recent computer to estimate the `psgp` paramaters, and 10-20 secondes to determine the optimal active set. This is slower than `gstat`, which only requires a few seconds for parameter estimation. This is because the parameter estimation involves minimisation of the evidence of the data, which is a somewhat costly operation. `gstat` uses a variogram fitting approach to determine the parameters, which is much faster. It is worth pointing out that the parameter estimation only needs to be done once, and further observations can then be processed by `psgp` without the need for additional training. It would also be possible to use a variogram-based approach to determine the parameters of `psgp` (or at least provide a good first guess for the parameters, thus reducing the number of optimisation steps required), improving the overall computation time.

5.4.3. Grid prediction

To assess the ability of `psgp` to capture the spatial characteristics of the data, we perform a prediction on a uniform grid covering the data. The grid spans from latitude 47.4 to 55.02 and longitude 6.04 to 15.03, at a square grid resolution of 0.0381 degrees (200 by 236 grid cells). The mean and standard deviation of the `psgp` prediction are shown on Figure 7 (centre and right plots, respectively) along with a scatter plot of the observations (left plot). For comparison, similar information is shown on Figure 8 for `gstat`.

The results show a strong agreement between the mean prediction fields, which both capture the main characteristics of the data. The predictive variance is lower for `psgp` than for `gstat`,

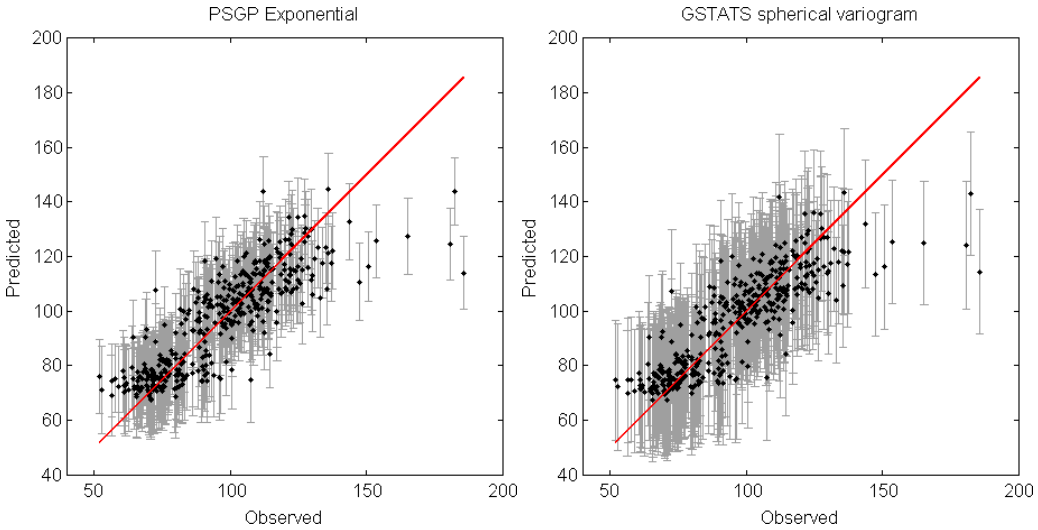

Figure 6: Scatter plot of observed values against predicted values.

which is most probably a consequence of the overconfidence observed on the validation data. The striking similarity of the predicted mean fields is a positive indication that it is possible to retain the spatial characteristics of the observed spatial field, to a great degree of accuracy, for only a portion of the data size. The variance shows a rapid increase in uncertainty as we move outside the observed area, which is expected given the local variability of the data.

6. Discussion and conclusions

The examples employed to demonstrate the software described are naïve in the sense that a minimal analysis has been reported due to space constraints. Two simple examples have shown and motivated why such an algorithm is important, particularly when implemented in a fast, efficient manner. The configuration of the algorithm was not discussed in detail as there is insufficient space, however the software provides significant scope for configuration. Fine tuning specific parameters can induce a further computational efficiency increase. For example, specifying the active set *a priori* instead of the default swapping-based iterative refinement is a useful option for reducing computation time. If the threshold for acceptance into the active set, γ, is set too low, then this can cause an increase in computation time due to frequent swapping in and out of active points.

It is clear from the profile marginal likelihood plots that for a given data set there will be a minimal number of active points required to make the estimate of the hyper-parameters robust. This is of some concern and suggest that good initialisation of the algorithm will be rather important, since a poor initialisation with ranges that are too long, might never recover if too few active points are permitted in the active set. Further research is required on how to select the active set, and its size most efficiently. Another feature we commonly observe is that where there is an insufficient active set allocate the model prefers to increase the nugget variance to inflate the predictive variance. We believe that an informative prior which strongly prefers lower nugget variances could help address this pathology.

Applying the `psgp` code to real data reveals the difficulty of choosing optimal values for the algorithm settings. For example the choice of a maximum active set size of 400 locations is driven by a desire to balance accuracy with computational speed. In different applications having a larger active set might make sense. The main benefit of the `psgp` approach is that one can treat large complex data sets in near real time. An interesting question which requires more

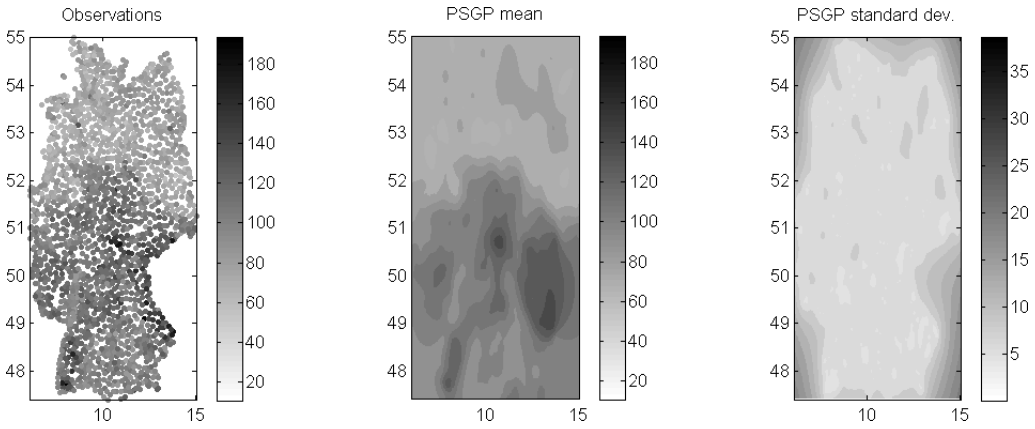

Figure 7: Interpolation using an exponential covariance function.

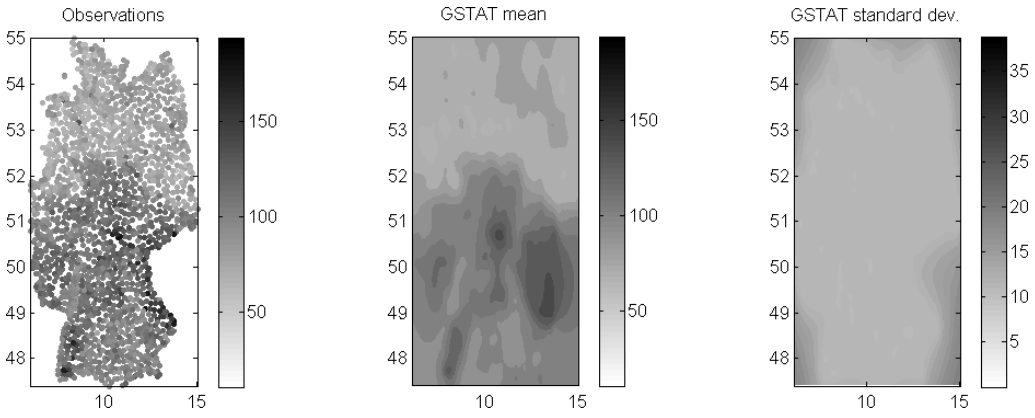

Figure 8: Interpolation using gstat

work is how we could exploit multi-core or multi-processor environments to further improve computational speed.

Future work will consider multiple outputs (co-kriging), the use of external predictors (universal kriging and regression kriging) in a batch projected Bayesian setting, and focus further on validation of the models.

Acknowledgements

This work is funded by the European Commission, under the Sixth Framework Programme, by Contract 033811 with DG INFSO, action Line IST-2005-2.5.12 ICT for Environmental Risk Management and as part of the RCUK funded MUCM project (EP/D048893/1).

Appendix A. Parametrisation of the posterior updates

This section provides some further insight into the parametrisation of the posterior Gaussian process described in Section 3 (Equations 3-5). The aim of this section is to identify the parameters q and r necessary to update the mean function and covariance function of the process given one or more observations. We consider the general case where several observations are considered at once, in which case the update requires a set of q_i and R_{ij} parameters (q is a vector and R a matrix). In the case of a single observation udpate, these parameters become the scalars $q+$ and $r+$ introduced in Section 3. These derivations and that of the following section are adapted from Appendices A and B in Csató and Opper (2002). Full detail can be found in Lehel Csató's PhD thesis (Csató, 2002).

Appendix A.1. Parametrisation of the mean function update

We consider a prior Gaussian process p_0 with mean function $\mu_0(\mathbf{x})$ and covariance function $c_0(\mathbf{x},\mathbf{x}')$. Assume we observe $\mathbf{y} = \{y_1,...,y_n\}$ at locations $\mathbf{X} = \{\mathbf{x}_1,...,\mathbf{x}_n\}$. The corresponding process response is denoted $\mathbf{f}_y = (f_0,...,f_n)$, where $f_i = f(\mathbf{x}_i)$. We denote $\mathbf{f}$ the process reponse over the entire domain. The posterior process given the observation is then expressed, following Bayes' rule:

$$p_{post}(\mathbf{f}) = \frac{1}{Z}\, p_0(\mathbf{f})\; p(\mathbf{y}|\mathbf{f}_y) \tag{A.1}$$

where $Z = \int p_0(\mathbf{f})\; p(\mathbf{y}|\mathbf{f}_y)\; d\mathbf{f}$ is a normalising constant. Note that the likelihood term only depends on the process at the observed locations, i.e. $p(\mathbf{y}|\mathbf{f}) = p(\mathbf{y}|\mathbf{f}_y)$.

The aim othe parametrisation is to express the posterior mean and variance in a recursive manner. This can be achieved using the Gaussian expectation identity from Appendix B, as detailed below.

We start by computing the mean function of the posterior process:

$$\mu_{post}(\mathbf{x}) = \int f_x\; p_{post}(\mathbf{f})\; d\mathbf{f} \tag{A.2}$$

$$= \frac{1}{Z}\int f_x\; p_0(\mathbf{f})\; p(\mathbf{y}|\mathbf{f}_y)\; d\mathbf{f} \tag{A.3}$$

$$= \frac{1}{Z}\int f_x\; p_0(f_x,\mathbf{f}_y)\; p(\mathbf{y}|\mathbf{f}_y)\; df\, d\mathbf{f}_y \tag{A.4}$$

where $f_x = f(x)$ is the process reponse at location $\mathbf{x}$ and the process only depends on the observed locations and $\mathbf{x}$ (the dependencies on other locations have been integrated out).

Applying the result from Appendix B, Equation (B.6):

$$\int x_i\, p(x)\, g(x)\, dx = \mu_i \int p(x)\, g(x)\, dx + \sum_{j=1}^{n} \Sigma_{i,j} \int p(x)\, \frac{\partial g(x)}{\partial x_j}\, dx \tag{A.5}$$

to $x_i = f_x$, $x = (f_x, \mathbf{f}_y)$ and $g(x) = p(\mathbf{y}|\mathbf{f}_y)$ leads to:

$$\mu_{post}(\mathbf{x}) = \frac{1}{Z}\left[\mu_0(\mathbf{x}) \int p_0(f_x,\mathbf{f}_y)\; p(\mathbf{y}|\mathbf{f}_y)\; df\, d\mathbf{f}_y + \sum_{i=1}^{n} c_0(\mathbf{x},\mathbf{x}_i) \int p_0(f_x,\mathbf{f}_y)\; \frac{\partial p(\mathbf{y}|\mathbf{f}_y)}{\partial f_i}\; df\, d\mathbf{f}_y\right]. \tag{A.6}$$

Note that the sum in the second term only includes n terms as $\partial p(\mathbf{y}|\mathbf{f}_y)/\partial f_x = 0$. Furthermore, f_x disappears from the integrals as it only appears in the prior. The normalisation factor cancels out the first integral, and leads to the parametrisation:

$$\mu_{post}(\mathbf{x}) = \mu_0(\mathbf{x}) + \sum_{i=1}^{n} c_0(\mathbf{x},\mathbf{x}_i)\, q_i. \tag{A.7}$$

with

$$q_i = \frac{\int p_0(\mathbf{f}_y)\; \frac{\partial}{\partial f_i} p(\mathbf{y}|\mathbf{f}_y)\; df\, d\mathbf{f}_y}{\int p_0(\mathbf{f}_y)\; p(\mathbf{y}|\mathbf{f}_y)\; df\, d\mathbf{f}_y}. \tag{A.8}$$

It is possible, through a change of variable in the numerator integral, to rewrite q_i as (see Csató and Opper (2002) for full derivation):

$$q_i = \frac{\partial}{\partial \mu_0(x_i)} \ln \int p_0(\mathbf{f}_y)\, p(\mathbf{y}|\mathbf{f}_y)\, d\mathbf{f}_y. \tag{A.9}$$

Appendix A.2. Parametrisation of the covariance function update

The expression of $R_i j$ follows a similar derivation. Start by noting that:

$$c_{post}(\mathbf{x}, \mathbf{x}') = \mu_{post}(\mathbf{x}\mathbf{x}') - \mu_{post}(\mathbf{x})\, \mu_{post}(\mathbf{x}') \tag{A.10}$$

and apply the result from Appendix B twice to get:

$$c_{post}(\mathbf{x}, \mathbf{x}') = c_0(\mathbf{x}, \mathbf{x}') + \sum_{i=1}^{n} \sum_{j=1}^{n} c_0(\mathbf{x}, \mathbf{x}_i)\, (D_i j - q_i q_j)\, c_0(\mathbf{x}_j, \mathbf{x}') \tag{A.11}$$

with

$$D_{ij} = \frac{1}{Z} \int p_0(\mathbf{f}_y)\, \frac{\partial^2}{\partial f_j \partial f_i}\, p(\mathbf{y}|\mathbf{f}_y)\, d\mathbf{f}_y. \tag{A.12}$$

R_{ij} is then defined as $D_{ij} - q_i q_j$. Like q_i, r_{ij} can be rewritten as the partial derivative of the logarithm of the expectation, after a change of variable and rearranging, leading to:

$$R_{ij} = \frac{\partial^2}{\partial \mu_0(x_i) \partial \mu_0(x_j)} \ln \int p_0(\mathbf{f}_y)\, p(\mathbf{y}|\mathbf{f}_y)\, d\mathbf{f}_y. \tag{A.13}$$

Appendix A.3. Recursive update

If we consider a single observation $(\mathbf{x}_+, y_+)$, we can update the mean function and covariance function of the posterior process as follows:

$$\mu^+_{post}(\mathbf{x}) = \mu_{post}(\mathbf{x}) + q_+ c_{post}(\mathbf{x}, \mathbf{x}_+) \tag{A.14}$$

$$c^+_{post}(\mathbf{x}, \mathbf{x}') = c_{post}(\mathbf{x}) + r_+ c_{post}(\mathbf{x}, \mathbf{x}_+)\, c_{post}(\mathbf{x}_+, \mathbf{x}') \tag{A.15}$$

where

$$q_+ = \frac{\partial}{\partial \mu_{post}(x_+)} \ln \int p_{post}(f_+)\, p(y_+|f_+)\, df_+ \tag{A.16}$$

$$r_+ = \frac{\partial^2}{\partial \mu_{post}(x_+)^2} \ln \int p_{post}(f_+)\, p(y_+|f_+)\, df_+. \tag{A.17}$$

These equation can be applied recursively from the prior, yielding to the following parametrisation (after n observations have been used to update the posterior):

$$\mu_{post}(\mathbf{x}) = \mu_0(\mathbf{x}) + \sum_{i=1}^{n} \alpha_i\, c_0(\mathbf{x}, \mathbf{x}_i) \tag{A.18}$$

$$c_{post}(\mathbf{x}, \mathbf{x}') = c_0(\mathbf{x}, \mathbf{x}')) + \sum_{i,j=1}^{n} c_0(\mathbf{x}, \mathbf{x}_i)\, \mathbf{C}(i, j)\, c_0(\mathbf{x}_j, \mathbf{x}') \tag{A.19}$$

with α and $\mathbf{C}$ depending on $q+$ and $r+$ as defined in Equation (5).

Appendix B. Gaussian expectation identity

Consider an n-dimensional Gaussian distribution $p(x)$ with mean μ and covariance matrix Σ. The expectation of x with respect to $p(x)\, g(x)$, where g is an arbitrary likelihood function, can be written:

$$\int x\, p(x)\, g(x)\, dx = \mu \int p(x)\, g(x)\, dx + \Sigma \int p(x)\, \nabla g(x)\, dx. \tag{B.1}$$

The proof stems from the partial integration rule (Gradshteyn and Ryzhik, 1994):

$$\int p(x)\,\nabla g(x)\,dx = -\int \nabla p(x)\,g(x)\,dx \tag{B.2}$$

in which fast decay of the Gaussian function is used to dimiss one of the terms. Substituting in (B.2) the gradient of the Gaussian distribution:

$$\frac{\nabla p(x)}{dx} = -\Sigma^{-1}(x-\mu)\,p(x) \tag{B.3}$$

leads to:

$$\int p(x)\,\nabla g(x)\,dx = -\Sigma^{-1}\int (x-\mu)\,p(x)\,g(x)\,dx \tag{B.4}$$

and, finally, after multiplication on both sides by Σ and rearranging:

$$\int x\,p(x)\,g(x)\,dx = \mu\int p(x)\,g(x)\,dx + \Sigma\int p(x)\,\nabla g(x)\,dx. \tag{B.5}$$

Last, note that for a given component x_i, this result becomes:

$$\int x_i\,p(x)\,g(x)\,dx = \mu_i\int p(x)\,g(x)\,dx + \sum_{j=1}^{n}\Sigma_{i,j}\int p(x)\,\frac{\partial g(x)}{\partial x_j}\,dx. \tag{B.6}$$

where the sum in the second term corresponds to the i-th element of the vector:

$$\Sigma\int p(x)\,\nabla g(x)\,dx.$$

Appendix C. Installation of the `gptk` library

Appendix C.1. Installation of IT++

`gptk` requires the IT++ library to be installed. On linux systems, IT++ binaries can usually be found in standard repositories. For instance, on Ubuntu 8.10, run the following command to install IT++:

```
$ sudo apt-get install libitpp6gf libitpp-dev libitpp6-dbg
```

Alternately, IT++ can be installed directly from source. This can be done through the following steps:

1. Download `itpp-4.0.6.tar.gz` and `itpp-external-3.2.0.tar.gz` from `http://sourceforge.net/projects/itpp/files`.
2. Extract, configure, build and install `itpp-external`:

   ```
   $ tar -xzvf itpp-external-3.2.0.tar.gz
   $ cd itpp-external-3.2.0
   $ ./configure
   $ make
   $ sudo make install
   $ cd ..
   ```

3. Extract, configure, build and install `itpp`:

   ```
   $ tar -xzvf itpp-4.0.6.tar.gz
   $ cd itpp-4.0.6
   $ ./configure --enable-debug --enable-exceptions
   $ make
   $ make check
   $ sudo make install
   $ cd ..
   ```

   Note: compilation of the library can take a long time on some less recent computers (about 15 minutes on a centrino duo laptop).

Appendix C.2. Installation of `gptk`

1. Download the `gptk` archive from `http://code.google.com/p/gptk/downloads/list`
2. Extract the archive:

   ```
   $ tar -xzvf libgptk-0.2.tar.gz
   ```

3. Run the `configure` script

   ```
   $ cd libgptk-0.2.tar.gz
   $ ./configure
   ```

   If you have installed IT++ in a non standard folder (e.g. `/opt/itpp`), you will need to indicate the path to the `itpp-config` script (excluding the script itself):

   ```
   $ ./configure --with-itpp-config=/opt/itpp/bin/
   ```

   We recommend you install `gptk` to a local directory, e.g. `/opt/gptk`:

   ```
   $ ./configure --prefix=/opt/gptk
   ```

4. Build and install the library:

   ```
   make && sudo make install
   ```

   This will build the library, the tests and the demonstration examples.
5. Take a look at the demonstration examples source code in the `examples/` folder and read the next section to get started with `gptk`.

Appendix D. Introduction to `gptk`

This simple tutorial shows how to train `psgp` on a univariate example. The data is generated from a sine function and corrupted with white noise. `psgp` is trained on the observed data and the posterior mean and variance are plotted.

Appendix D.1. Example source code

```
2  /*
3   * Simple noisy sine example
4   *
5   * Author: Remi Barillec <r.barillec@aston.ac.uk>
6   */
7  #include <iostream>
8  #include <itpp/itbase.h>
9
10 #include "itppext/itppext.h"
11 #include "plotting/GraphPlotter.h"
12 #include "covariance_functions/SumCF.h"
13 #include "covariance_functions/GaussianCF.h"
14 #include "covariance_functions/WhiteNoiseCF.h"
15 #include "gaussian_processes/PSGP.h"
16 #include "likelihood_models/GaussianLikelihood.h"
17 #include "optimisation/SCGModelTrainer.h"
```

We start by including the headers of all required files. These are essentially the IT++ library base header and the `gptk` classes needed in the example.

```
19 using namespace std;
20 using namespace itpp;
21 using namespace itppext;
```

We also import all declarations in the standard library, the IT++ library and our set of extension routines to IT++, `itppext`.

```
23 #define NUM_X 200           // X resolution
24 #define NUM_OBS 20          // Number of observations
25 #define NUM_ACTIVE 5        // Number of active points
26 #define NOISE_VAR 0.1       // Observation noise variance
```

We define a few constants: the resolution at which we sample the data, the total number of observations, the number of active points and the observation noise variance.

```
28 int main(int argc, char* argv[]) {
29
30   // Generate data from a sine
31   vec X = linspace(0, 2*pi, NUM_X);
32   vec Y = sin(X);              // True function
33
34   // Randomly select a subset of the data as observations,
35   // and add white noise
36   ivec Iobs = randperm(NUM_X).left(NUM_OBS);
37   vec Xobs = X(Iobs);
38   vec Yobs = Y(Iobs) + sqrt(NOISE_VAR)*randn(NUM_OBS);
```

The first part of the example consists in generating some data from a sine function. We first take some points between 0 and π (line 31) and compute the corresponding $y(x) = sin(x)$ (line 32). We then take a random subset of the points and corrupt them with noise. On line 36,

randperm returns a vector of random permutations of all numbers between 1 and its argument, which we then truncate to keep only the first NUM_OBS values. We then extract the x_i (line 37) and y_i (line 38) corresponding to the selected indices, and corrupt the y_i with Gaussian white noise of fixed variance (line 38).

```
40  // Covariance function: Gaussian + Nugget
41  double range = 0.5;       // Initial range
42  double sill = 1;         // Initial sill
43  double nugget = NOISE_VAR;  // Nugget variance
44
45  GaussianCF   gaussianCovFunc(range, sill);
46  WhiteNoiseCF nuggetCovFunc(nugget);
47  SumCF covFunc;
48  covFunc.add(gaussianCovFunc);
49  covFunc.add(nuggetCovFunc);
```

We then initialise the covariance function for psgp. Here, we decide to use a Gaussian kernel with white noise. The range and sill (or length scale and variance) of the Gaussian kernel and the variance of the white nosie are set to sensible values (lines 41-43). We then create a GaussianCF and a WhiteNoiseCF objects and add them together inside a SumCF object (lines 45-49).

```
51  // Initialise psgp - we need to convert the input vector
52  // to a matrix
53  mat XobsMat = mat(Xobs);
54  PSGP psgp(XobsMat, Yobs, covFunc, NUM_ACTIVE);
```

We then create a PSGP object, specifying the training inputs (XobsMat) and outputs (Yobs), the covariance function structure (covFunc, previously defined) and the number of active points. Note that the PSGP constructor expects a matrix of inputs rather than a vector, so line 53 converts the vector of inputs into a matrix.

```
56  // Use a Gaussian likelihood model with fixed variance
57  GaussianLikelihood gaussLik(nugget);
58
59  // Compute the posterior (first guess), which projects the whole
60  // observed data onto a subset of optimal active points.
61  psgp.computePosterior(gaussLik);
```

The next step is to select the set of active points, based on the initial parameters. This requires a likelihood to be specified for each observation, or as is the case here, to use a common likelihood for all observations. A GaussianLikelihood object is created with variance equal to the observation noise (assumed known). The computePosterior method of the PSGP object is the core routine, which selects the optimal active points based on the specified likelihood.

```
63  // Estimate parameters using Scaled Conjugate Gradients
64  SCGModelTrainer scg(psgp);
65  scg.Train(10);
66
67  // Recompute the posterior and active points for the new parameters
68  psgp.resetPosterior();
69  psgp.computePosterior(gaussLik);
```

Having selected a set of initial active points, we go on to estimate the parameters of the covariance function. This is done using an SCGModelTrainer, which applies a scaled-conjugate gradient minimisation to the error function (by default, the error function is the log-evidence of the data). The psgp object is a type of Optimisable object, meaning it provides an objective function and its gradient, which are used by the SCGModelTrainer.

Once the parameters have been optimised (line 65), the active set is recomputed from scratch, by first resetting the posterior and recomputing it.

```
71   // Make predictions - we use the Gaussian covariance function only
72   // to make non-noisy prediction (no nugget)
73   vec psgpmean = zeros(NUM_X);
74   vec psgpvar  = zeros(NUM_X);
75   psgp.makePredictions(psgpmean, psgpvar, mat(X), gaussianCovFunc);
```

Prediction is then made using the `makePrediction` method of `PSGP`. The first 2 arguments are vectors for storing the predicted mean and variance, respectively. The next argument is the matrix of locations at which the prediction is made (here, the entire domain). The last argument (optional) can be used to specify a different covariance function for prediction. This is mainly useful, as here, to make non-noisy prediction: one just needs to pass in the covariance function structure deprived of the nugget term.

```
77   // Plot results using GraphPlotter (a gnuplot interface)
78   GraphPlotter gplot = GraphPlotter();
79
80   // Plot psgp mean and variance
81   gplot.plotPoints(X, psgpmean, "psgp mean", LINE, BLUE);
82   gplot.plotPoints(X, psgpmean + 2.0*sqrt(psgpvar), "error bar", LINE, CYAN);
83   gplot.plotPoints(X, psgpmean - 2.0*sqrt(psgpvar), "error bar", LINE, CYAN);
84
85   // Plot true function and observations
86   gplot.plotPoints(X, Y, "true function", LINE, RED);
87   gplot.plotPoints(Xobs, Yobs, "observations", CROSS, BLACK);
88
89   // Plot active points
90   vec activeX = (psgp.getActiveSetLocations()).get_col(0);
91   vec activeY = Yobs(psgp.getActiveSetIndices());
92   gplot.plotPoints(activeX, activeY, "active points", CIRCLE, BLUE);
93
94   return 0;
95 }
```

The last step consists in plotting the data using the `GraphPlotter`. The `GraphPlotter` uses Gnuplot as a back end to generate and display the plots. Plotting is done by calling the `plotPoints` method, passing in vectors of x and y coordinates. The next 3 arguments are the title of the plot, the type (`LINE` for a solid line, `CROSS` for discrete '+' markers) and the colour.

Appendix D.2. Building the example

The source code above is provided in the example file `demo_sine.cpp`. We show here how it is compiled and linked against the libraries (`gptk` and IT++). We assume the `itpp-config` script is found in the system path. Otherwise, the full path to the script must be given (that is, replace any instance of `itpp-config` with `/opt/itpp/bin/itpp-config` or whichever path `itpp` is installed in).

```
97   g++ -c  `itpp-config --cflags` -I/opt/gptk/include demo_sine.cpp
98   g++ `itpp-config --libs` -L/opt/gptk/lib -lgptk demo_sine.o -o demo_sine
```

The example can then be run using

```
99   ./demo_sine
```

If the libraries (IT++ and `gptk`) are not in standard location (e.g. `/usr/lib`), the command above will fail, complaining about missing libraries. To fix this error, one needs to

add the libraries' path to the runtime library path (where the system looks for installed libraries). For example, if both IT++ and `gptk` are installed in `/opt`, this is done by changing the `LD_LIBRARY_PATH` environment variable:

```
100  export LD_LIBRARY_PATH=/opt/itpp/lib:/opt/gptk/lib:$LD_LIBRARY_PATH
101  ./demo_sine
```

Further, more advanced examples can be found in the `examples/` folder that ships with the library.

References

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D., 1999. LAPACK Users' Guide. Society for Industrial and Applied Mathematics, Philadelphia, PA. 3rd edition. 429pp.

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

Blackford, L.S., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., Lumsdaine, A., Petitet, A., Pozo, R., Remington, K., Whaley, R.C., 2002. An updated set of basic linear algebra subprograms (BLAS). ACM Transactions on Mathematical Software 28, 135–151.

Boersma, K.F., Eskes, H.J., Brinksma, E.J., 2004. Error analysis for tropospheric NO2 retrieval from space. Journal of Geophysical Research, Atmospheres 109, D04311. doi: 10.1029/2003JD003962.

Bourennane, H., Dère, C., Lamy, I., Cornu, S., Baize, D., Van Oort, F., King, D., 2006. Enhancing spatial estimates of metal pollutants in raw wastewater irrigated fields using a topsoil organic carbon map predicted from aerial photography. The Science of the Total Environment 361 (1-3), 229–248.

Cappé, O., Guillin, A., Marin, J., Robert, C., 2004. Population Monte Carlo. Journal of Computational and Graphical Statistics 12, 907–929.

Collier, C., 1989. Applications Of Weather Radar Systems: A Guide To Uses Of Radar Data In Meteorology And Hydrology. John Wiley and Sons, New York. 294pp.

Cornford, D., Csato, L., Opper, M., 2005. Sequential, Bayesian Geostatistics: A principled method for large data sets. Geographical Analysis 37, 183–199.

Cover, T.M., Thomas, J.A., 1991. Elements of Information Theory. John Wiley & Sons. 542pp.

Cox, S., 2007. Observations and Measurements, Part 1. Open Geospatial Consortium, Inc. 85pp. http://www.opengeospatial.org/standards/om, [Accessed June 23, 2010].

Cressie, N., Johannesson, G., 2008. Fixed rank kriging for very large spatial data sets. Journal of the Royal Statistical Society, Series B, Statistical Methodology 70 (1), 209–226.

Cressie, N.A.C., 1993. Statistics for Spatial Data. John Wiley and Sons, New York. 928pp.

Csató, L., 2002. Gaussian Processes - Iterative Sparse Approximations. Ph.D. Dissertation. Neural Computing Research Group, Aston University. Birmingham, UK. 116pp.

Csató, L., Opper, M., 2002. Sparse online Gaussian processes. Neural Computation 14 (3), 641–668.

Diggle, P.J., Ribeiro, P.J., 2007. Model-based Geostatistics. Springer Series in Statistics. 230pp.

Fuentes, M., 2002. Spectral methods for nonstationary spatial processes. Biometrika 89 (1), 197–210.

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

Gradshteyn, I., Ryzhik, I., 1994. Table of Integrals, Series, and Products. Academic Press, New York. 1204pp.

Haas, T.C., 1990. Kriging and automated variogram modeling within a moving window. Atmospheric Environment. Part A. General Topics 24 (7), 1759–1769.

Ingram, B., Barillec, R., 2009. Projected spatial gaussian process (psgp) methods. http://cran.r-project.org/web/packages/psgp/. [Accessed June 23, 2010].

Ingram, B., Barillec, R., Cornford, D., 2010. Gaussian process toolkit (gptk). http://code.google.com/p/gptk/. [Accessed June 23, 2010].

Jensen, J., 2000. Remote Sensing Of The Environment: An Earth Resource Perspective. Geographic Information Science, Prenctice-Hall, Upper Saddle River, New Jersey. 544pp.

Journel, A.G., Huijbregts, C.J., 1978. Mining Geostatistics. Academic Press, London. 610pp.

Kalnay, E., 2003. Atmospheric Modelling, Data Assimilation and Predictability. Cambridge University Press, Cambridge. 364pp.

Lawrence, N., Seeger, M., Herbrich, R., 2003. Fast sparse Gaussian process methods: The informative vector machine. Advances in Neural Information Processing Systems 15, 609–616.

Lawson, C.L., Hanson, R.J., Kincaid, D., Krogh, F.T., 1979. Basic linear algebra subprograms for FORTRAN usage. ACM Transactions on Mathematical Software 5, 308–323.

Marshall, J., Palmer, W., 1948. The distribution of raindrops with size. Journal of Atmospheric Sciences 5, 165–166.

Minka, T.P., 2001. Expectation propagation for approximate Bayesian inference. Uncertainty in Artificial Intelligence 17, 362–369.

Open Geospatial Consortium, I., 2007. Sensor web enablement standard. http://www.opengeospatial.org/ogc/markets-technologies/swe. [Accessed June 23, 2010].

Opper, M., 1996. Online versus offline learning from random examples: General results. Phys. Rev. Lett. 77 (22), 4671–4674.

Ottosson, T., Piatyszek, A., et al., 2009. IT++, a C++ library of mathematical, signal processing and communication routines. http://sourceforge.net/apps/wordpress/itpp/. [Accessed June 23, 2010].

Pebesma, E., Wesseling, C., 1998. Gstat: a program for geostatistical modelling, prediction and simulation. Computers & Geosciences 24 (1), 17–31.

Rasmussen, C.E., Williams, C.K.I., 2006. Gaussian Processes for Machine Learning. The MIT Press. 266pp.

Snelson, E., Ghahramani, Z., 2006. Sparse Gaussian processes using pseudo-inputs. Advances in Neural Information Processing Systems 18, 1257–1264.

Stöhlker, U., Bleher, M., Szegvary, T., Conen, F., 2009. Inter-calibration of gamma dose rate detectors on the european scale. Radioprotection 44, 777–783.

Williams, C., 1998. Computation with infinite neural networks. Neural Computation 10 (5), 1203–1216.

Williams, T., Kelley, C., et al., 2009. Gnuplot, an interactive plotting program. `http://www.gnuplot.info`. [Accessed June 23, 2010].