



Neural Computing Research Group
Aston University
Birmingham B4 7ET
United Kingdom
Tel: +44 (0)121 333 4631
Fax: +44 (0)121 333 4586
<http://www.ncrg.aston.ac.uk/>

Projected Sequential Gaussian Processes: A C++ tool for interpolation of heterogeneous data sets

Remi Barillec, Ben Ingram, Dan Cornford, Lehel Csató

Abstract

Within MUCM there might occasionally arise the need to use large training set sizes, or employ observations with non-Gaussian noise characteristics or non-linear sensor models in a calibration stage. This technical report deals with Gaussian process models in these non-Gaussian, and / or large data set size cases. Treating such data within Gaussian processes is most naturally accomplished using a Bayesian approach, however such methods generally scale rather badly with the size of data set, and require computationally expensive Monte Carlo based inference in non-Gaussian settings. 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, `psgp`, 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.

Contents

1	Introduction	3
2	Context	3
3	Interpolation for large data sets	6
4	The projected sequential Gaussian process model	7
4.1	Computing the update coefficients	8
4.2	Sparse extension to the sequential updates	9
4.3	Expectation-propagation	9
4.4	The complete <code>psgp</code> algorithm	10
5	Design of C++ library	11
5.1	Covariance functions	12
5.2	Likelihood models	12
5.3	Optimisation of hyper-parameters	12
5.4	Miscellaneous tasks	13
6	Examples and use	13
6.1	Illustration of the algorithm	13
6.2	Heterogeneous, non-Gaussian observation noise	15
6.3	Parameter estimation	17
6.4	Application to spatial interpolation	17
7	Discussion and conclusions	19

1 Introduction

This report details the work carried out in MUCM within Work Package 1.2 and also as part of the EC funded INTAMAP project. The INTAMAP project develops methods for automatic interpolation of spatially-distributed environmental data such as rainfall measurements, radiation soil samples, etc. In this report, we look at an approximation to Gaussian processes called Projected Sparse Gaussian Processes (PSGP) and their application to the specific problem of spatial interpolation. The methodology is relevant to MUCM in the case where a large training set is needed, since the inference scales linearly with the data set size and quadratically with the number of active points retained. The method might also be appropriate in the dynamic emulation case where the parametrised posterior can be updated efficiently as each new iteration of the dynamic model is undertaken, and the effect of very close (in input-output space) sample points can be projected onto the existing approximation without undue numerical issues arising.

The method might also be of some benefit in the case where observations are available and the aim is calibration. In this setting the observations might not be directly of the simulator outputs and might also be subject to non-Gaussian errors, leading to non-Gaussian (or t) posterior distributions. In that setting the projection onto the best approximating Gaussian posterior might have some significant computational benefits compared with a Monte Carlo approach.

The text of the report is largely taken from a paper submitted to Computers and Geosciences which discusses projected sequential Gaussian processes and their implementation in the C++ programming language, in the context on the INTAMAP project, which funded much of the relevant work described herein. The report acts as partial documentation to the software which is released under an open source licence.

2 Context

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 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:

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

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 observa-

tion 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 $y_i = [f(\mathbf{x}_i) + b] + \varepsilon_i$. 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. Note 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 2.

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 $\mathbf{x}_1, \dots, \mathbf{x}_n$
$h(\cdot)$	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:

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

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 numer-

ical 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 4. 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 library.

We have chosen to implement our algorithm using C++ to ensure that program execution is efficient and can be optimised where required. A web processing service interface to the psgp library can be accessed from the INTAMAP project web site¹.

We start in Section 3 by discussing large datasets and how these can be treated using a number of different approximations. In Section 4 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 5. Section 6 shows how the software can be applied to synthetic and real-world data. Section 7 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++ psgp library.

¹<http://www.intamap.org>

3 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). Each technique 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. 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 which means that the `psgp` library can process sensor models described in MathML, as might be typical in a SensorML description in an Observations and Measurements document² part of the Sensor Web Enablement³ suite of standards proposed by the Open Geospatial Consortium.

²<http://www.opengeospatial.org/standards/om>

³<http://www.opengeospatial.org/ogc/markets-technologies/swe>

Algorithm 1 psgp algorithm outline for fixed θ .

```

1: Initialise model parameters  $(\vec{\alpha}, \vec{C})$ ,  $(\vec{a}, \vec{\Lambda}, \vec{P})$  to empty values.
2: for eplter = 1 to Data Recycling Iterations do
3:    $t \leftarrow 1$ 
4:   Randomise the order of location/observation pairs.
5:   while t < numObservations do
6:     Get next location and observation  $(\mathbf{x}_t, y_t)$ .
7:     if eplter > 1 then // Observation already processed
8:       Remove contribution of current observation  $(\vec{\alpha}, \vec{C})$ 
9:     end if
10:    if Likelihood update is analytic then // Compute update coefficients
11:      Compute  $(q_+, r_+)$ 
12:      from first and second derivatives of the marginal likelihood
13:    else
14:      Use population Monte Carlo sampling for computing  $(q_+, r_+)$ 
15:    end if
16:    Calculate  $\gamma$ , //  $\gamma > 0$  is a heuristic measure; the difference between
17:    // the current GP and that from the previous iteration
18:    if  $\gamma > \epsilon$  then // Using full update of the parameters
19:      Add location  $\mathbf{x}_t$  to the Active Set
20:      Increase the size of  $(\vec{\alpha}, \vec{C})$ 
21:      Using  $(q_+, r_+)$ , update  $(\vec{\alpha}, \vec{C})$  and EP parameters  $(\vec{a}, \vec{\Lambda}, \vec{P})$ ;
22:    else // Model parameters updated by a projection step
23:      Project observation effect  $(q_+, r_+)$  onto the current Active Set
24:      Update  $(\vec{\alpha}, \vec{C})$  and the EP parameter matrices  $(\vec{a}, \vec{\Lambda}, \vec{P})$ 
25:    end if
26:    if Size of active set > Maximum Desired Size then
27:      Calculate the informativeness for each Active Set element
28:      Find the least informative element, delete it from the active set
29:      Update  $(\vec{\alpha}, \vec{C})$  and EP parameters  $(\vec{a}, \vec{\Lambda}, \vec{P})$ 
30:    end if
31:     $t \leftarrow t + 1$ 
32:     $(\vec{\alpha}, \vec{C}) \leftarrow (\vec{\alpha}, \vec{C})$ 
33:  end while
34: end for

```

4 The projected sequential Gaussian process model

To explain the psgp approach, we give an outline (Algorithm 1) for the computation of the posterior distribution. The basis of this method is *the parametrisation* of the Gaussian process approximation to the posterior distribution from eq. (2) with mean function $\mu(\mathbf{x})$ and covariance kernel $c(\mathbf{x}, \mathbf{x}')$. Although the prior Gaussian process, $p_0(f|\mathbf{x}, \theta)$ has a zero mean function, the posterior Gaussian process approximation will have a non-zero mean function due to the impact of the observations. The low-rank posterior distribution is parametrised using a vector $\vec{\alpha}$ and a matrix \vec{C} as:

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

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

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 $\mathcal{AS} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ of m locations where typically $m \ll n$. 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.

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

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

where $\vec{c}_{t+1} = [c_0(\mathbf{x}_{t+1}, \mathbf{x}_1), \dots, c_0(\mathbf{x}_{t+1}, \mathbf{x}_r)]^T$ is the prior covariance between the location being considered at iteration $t+1$ and all current points in the active set. $\vec{e}_{t+1} = [0, 0, \dots, 1]^T$ is the $t+1$ -th unit vector that extends the size of $\vec{\alpha}_{t+1}$ and \vec{C}_{t+1} by one. The 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 4.1 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 4.2.

4.1 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 `psgp` 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. 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 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

Algorithm 2 The 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  $\pi_*(f) = N(\mu_*, \sigma_*^2)$  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  $\pi_*(f)$ 
8:     Set  $w_j = \pi_*(f_j) \times p_i(y_i|f_j, \mathbf{x}_i, h_i) / p(f_j|\mathbf{x}_i)$  // 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:   $\pi_*(f) \leftarrow N(\mu_*, \sigma_*^2)$  // Update the proposal distribution
14: end for
15:  $q_+ = (\mu_* - \mu_i) / \sigma_i^2$  // Update  $q$  and  $r$  using the final values of  $\mu_*$  and  $\sigma_*^2$ 
16:  $r_+ = (\sigma_*^2 - \sigma_i^2) / (\sigma_i^2)^2$ 

```

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.

4.2 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 Oppér (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 $\vec{k}_* = [c_0(\mathbf{x}_*, \mathbf{x}_1), \dots, c_0(\mathbf{x}_*, \mathbf{x}_t)]^T$ 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).

4.3 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 (2000) 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 $\exp[-\lambda_i(\boldsymbol{\pi}_i^T \mathbf{f}_{\mathcal{A}_S} - a_i)^2/2]$ where $\mathbf{f}_{\mathcal{A}_S}$ 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 $\vec{P} = [\boldsymbol{\pi}_1, \dots, \boldsymbol{\pi}_n]$, 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, $\boldsymbol{\theta}$, need to be optimised for the data set: it can also provide improved stability in the algorithm Minka (2000).

4.4 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 1)
- Create a likelihood object for the observations (line 3). 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 (\mathbf{y}) and covariance function (line 5). 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 7, 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 8). 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.
- The parameter estimation consists of an inner loop (line 10), in which optimal parameters are sought which minimise the objective function. Because the objective function, optimal hyper-parameter values and active set are *all inter-dependent*, 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 10) and update of the active set (line 11) alternate in an outer loop (lines 9-12), 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 13). 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.

Algorithm 3 The 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
2: // hyper-parameters

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

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

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

8: SCGModelTrainer opt(psgp); // Initialise optimisation object and link to psgp object
// Optimise the hyper-parameters

9: for hypIter = 1 to Hyper-Parameter Estimation Iterations do
10:   opt.train(5); // Perform 5 optimisation steps
11:   psgp.computePosterior(lh); // Recompute posterior for updated hyper-parameters
12: end for

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

```

5 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++⁴ library

⁴<http://itpp.sourceforge.net/>

which itself uses the highly optimised vector/matrix operations available from BLAS⁵ and LAPACK⁶.

5.1 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 `SumCovarianceFunction` class. All the above provide an analytic implementation of the gradient with respect to the hyper-parameters, θ .

5.2 Likelihood models

A `LikelihoodType` interface is implemented by and subclassed as either a `AnalyticLikelihood` 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 4.1 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.

5.3 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

⁵<http://www.netlib.org/blas>

⁶<http://www.netlib.org/lapack>

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.

5.4 Miscellaneous tasks

Aside of the core `psgp` classes, the library also provides basic support for other secondary tasks. These include import/export of the data from/to CSV (comma separated values) 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`⁷.

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.

6 Examples and use

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

6.1 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 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

⁷<http://www.gnuplot.info/>

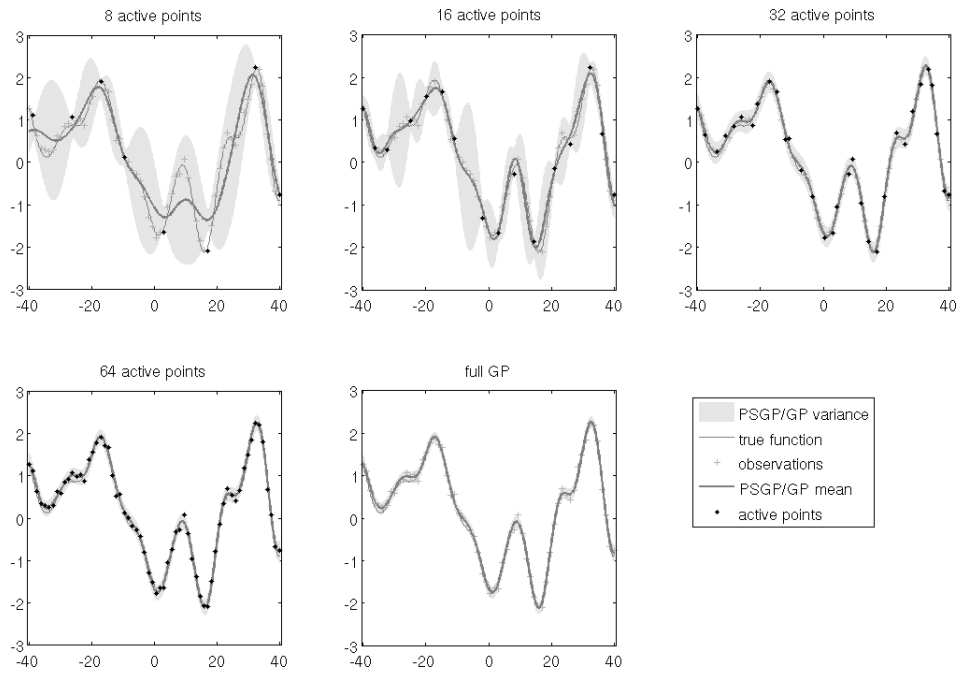


Figure 1: Quality of the `psgp` approximation for different active set sizes.

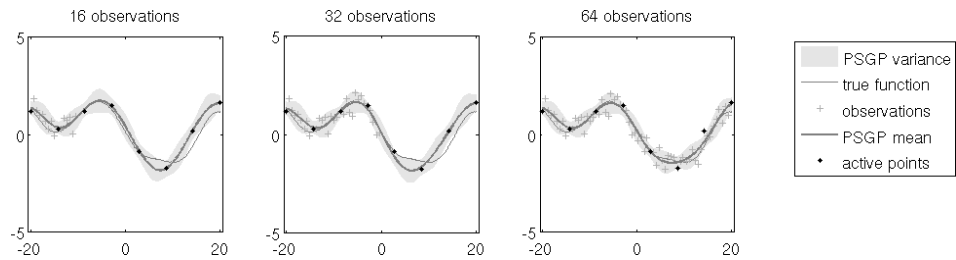


Figure 2: Evolution of the `psgp` approximation as observations are processed

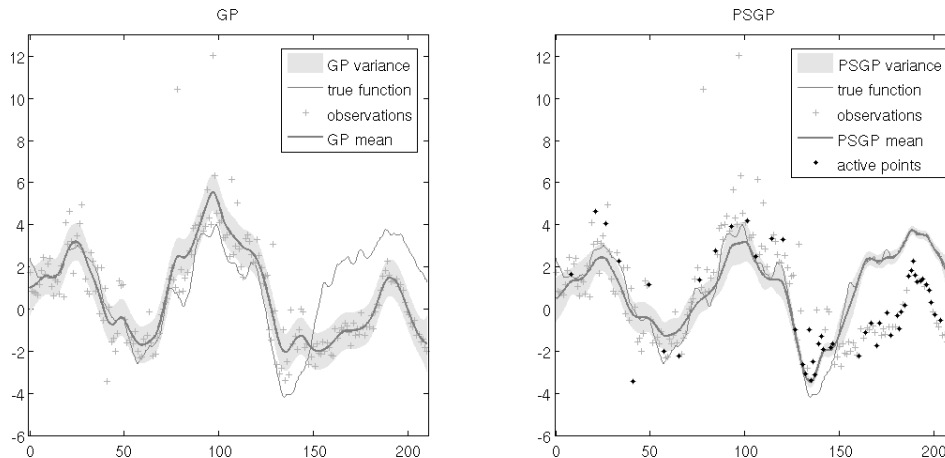


Figure 3: Example with heterogeneous observation noise.

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.

6.2 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.

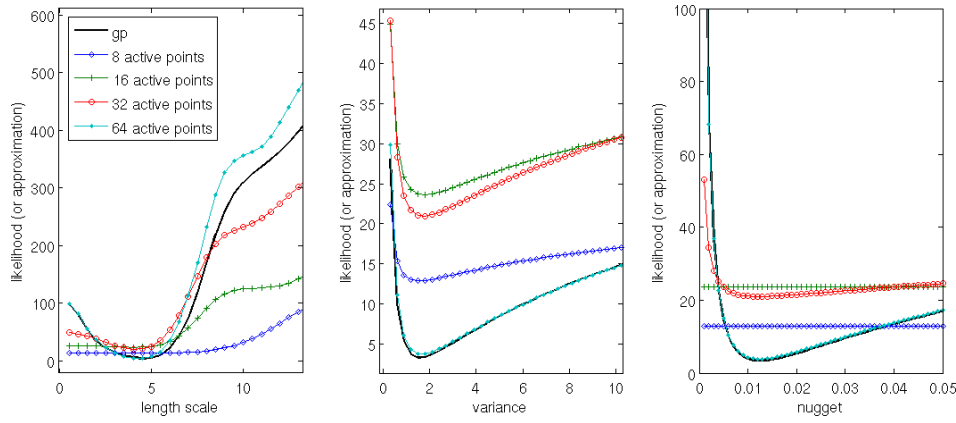


Figure 4: Marginal likelihood profiles (upper bound approximation) for Example 6.1.

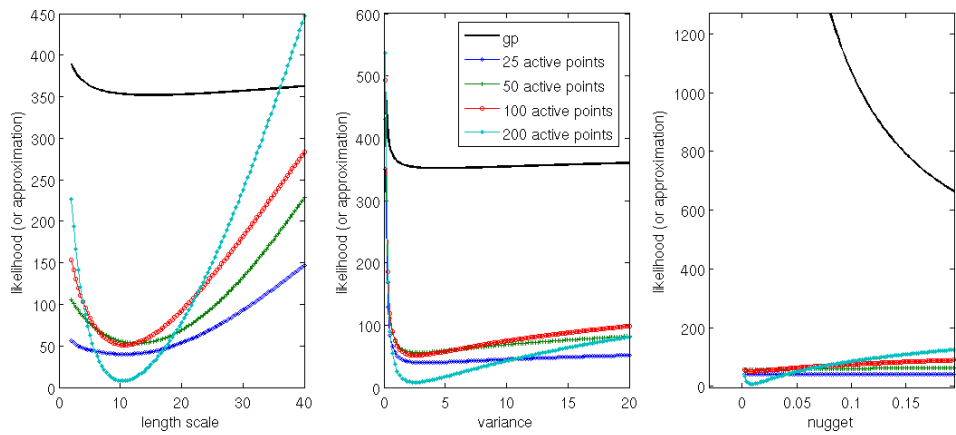


Figure 5: Marginal likelihood profiles (upper bound approximation) for Example 6.2.

6.3 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 6.1 and Example 6.2 respectively. These 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 6.1 (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 retaining a large 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 6.2, 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.

6.4 Application to spatial interpolation

The third example shows an application of the `psgp` method to the interpolation of spatial observations. The dataset is taken from the INTAMAP case studies⁸. It consists of 2834 spatial measurements of gamma dose rate from the EURDEP monitoring network (Stöhler et al., 2009). 566 observations 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 on that example includes the following settings:

- the covariance function chosen is a mixture of a Matern 5/2, white noise and bias covariance functions. We also look at replacing the Matern 5/2 term with an Exponential covariance function to illustrate the flexibility of the model,
- 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),
- prediction is done over a uniform grid covering the data.

⁸http://www.intamap.org/sample_data.php\#radiation

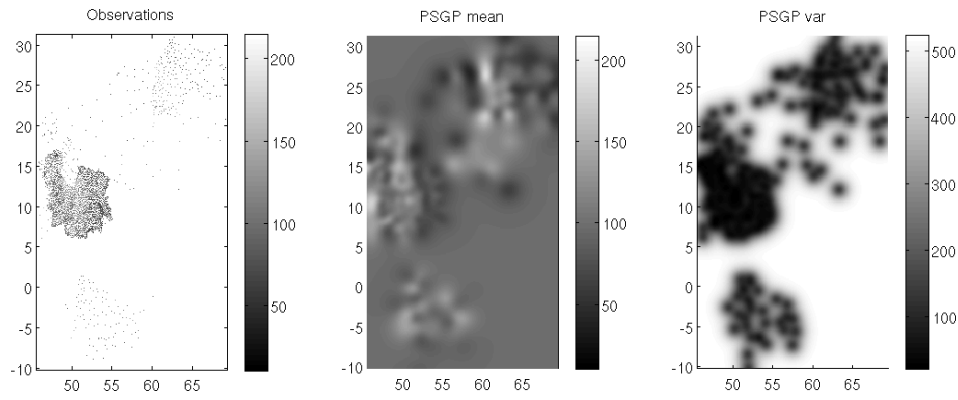


Figure 6: Interpolation using a Matern 5/2 covariance function.

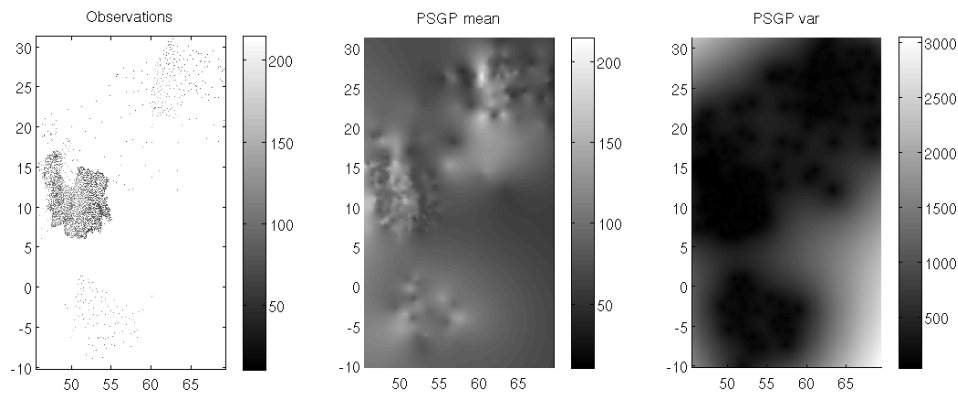


Figure 7: Interpolation using an Exponential covariance function.

The predictive mean and variances are shown in Figure 6 (centre and right plot, respectively) along with the observation locations (left plot). Similar information is shown on Figure 7, this time using an Exponential covariance function instead of the Matern 5/2. In both cases, the mean process captures the main characteristics of the data although the resulting mean field is a little too smooth for the Matern case in regions of high variability. This is a problem often associated with stationary covariance functions, which assume the spatial correlation of the data only depends on the distance between locations, not on the locations themselves. Thus, the optimal set of hyper-parameters will try to accommodate both the small scale and large scale patterns found in the data, favouring a good overall fit at the expense of local accuracy. The addition to the framework of non-stationary covariance functions and the use of more flexible mixtures of covariance functions could help address this problem. 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.

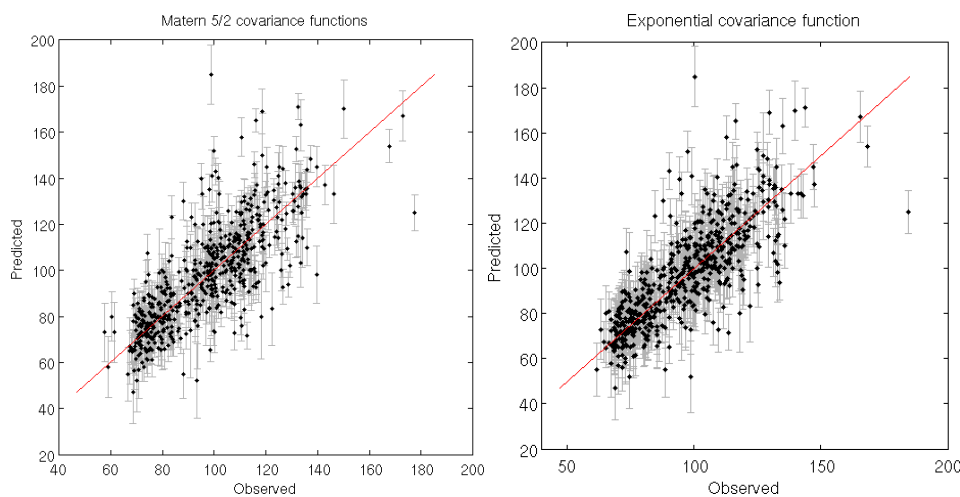


Figure 8: Scatter plot of observed values against predicted values.

Figure 8 shows a scatter plot of the predicted values at the 566 unobserved locations against the actual values, for the Matern 5/2 case (left) and the Exponential case (right). There is an overall agreement between the prediction and the targets, which fall within 2 standard deviations of the predicted value most of the time. The prediction variances are consistent with the spread of the mismatch, demonstrating that the uncertainty has been assessed correctly by the model.

7 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 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 was 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). This report was submitted for publication in *Computers and Geoscience*.

References

- K F Boersma, H J Eskes, and E J Brinksma. Error analysis for tropospheric NO₂ retrieval from space. *Journal of Geophysical Research: Atmospheres*, 109:D04311, 2004.
- H. Bourenane, C. Dère, I. Lamy, S. Cornu, D. Baize, F. Van Oort, and D. King. 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, 2006.
- O. Cappé, A. Guillin, J.M. Marin, and C.P. Robert. Population Monte Carlo. *Journal of Computational and Graphical Statistics*, 12:907–929, 2004.
- D Cornford, L Csato, and M Opper. Sequential, Bayesian Geostatistics: A principled method for large data sets. *Geographical Analysis*, 37:183–199, 2005.
- Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, 1991.
- N. Cressie and G. Johannesson. Fixed rank kriging for very large spatial data sets. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):209–226, 2008.

- N A C Cressie. *Statistics for Spatial Data*. John Wiley and Sons, New York, 1993.
- L. Csató and M. Opper. Sparse online Gaussian processes. *Neural Computation*, 14(3): 641–669, 2002.
- P J Diggle and P J Ribeiro. *Model-based Geostatistics*. Springer Series in Statistics, 2007.
- M. Fuentes. Spectral methods for nonstationary spatial processes. *Biometrika*, 89(1): 197–210, 2002.
- R. Furrer, M. G. Genton, and D. Nychka. Covariance tapering for interpolation of large spatial datasets. *Journal of Computational and Graphical Statistics*, 15(3):502–523, 2006.
- A G Journel and C J Huijbregts. *Mining Geostatistics*. Academic Press, London, 1978.
- E. Kalnay. *Atmospheric Modelling, Data Assimilation and Predictability*. Cambridge University Press, Cambridge, 2003.
- N.D. Lawrence, M. Seeger, and R. Herbrich. Fast sparse Gaussian process methods: The informative vector machine. *Advances in Neural Information Processing Systems*, 15: 609–616, 2003.
- Thomas P. Minka. *Expectation Propagation for Approximate Bayesian Inference*. PhD thesis, Dep. of El. Eng. & Comp. Sci.; MIT, 2000.
- Manfred Opper. Online versus offline learning from random examples: General results. *Phys. Rev. Lett.*, 77(22):4671–4674, 1996.
- C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.
- E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. *Advances in Neural Information Processing Systems*, 18:1257, 2006.
- U. Stöhlker, M. Bleher, T. Szegvary, and F. Conen. Inter-calibration of gamma dose rate detectors on the european scale. *Radioprotection*, 44:777–783, 2009.