Adaptive Surrogate Modelling for Global Optimization

Richard P. Dwight, Jouke de Baar, Iliass Azijli
TU Delft, Department of Aerodynamics

August 27, 2018

Contents

Summary

We discuss surrogate modelling techniques for global optimization problems. The focus is on Efficient Global Optimization (EGO). In order to achieve a deep understanding of this probabilistic technique, we begin with a short introduction to probability theory and Bayesian statistics. This gives us the mathematical tools to develop Gaussian process regression. We examine how this surrogate modelling framework can be generalised to incorporate gradient information, variably-fidelity simulations, and even experimental data. Random and deterministic, non-adaptive and adaptive design space sampling techniques are discussed. Given this framework a derivation of EGO becomes entirely natural.

1 Introduction

We discuss numerical methods for solving global optimization problems. The problem is stated as: find $\bar{\boldsymbol{\xi}} \in \mathcal{H}_d$ such that

$$f(\bar{\boldsymbol{\xi}}) \leq f(\boldsymbol{\xi}) \qquad \forall \boldsymbol{\xi} \in \mathcal{H}_d.$$

Alternatively: find

$$\bar{\boldsymbol{\xi}} = \arg\min_{\boldsymbol{\xi} \in \mathcal{H}_d} f(\boldsymbol{\xi}).$$

Here $f : \mathcal{H}_d \to \mathbb{R}$ is a computationally expensive cost function, $\mathcal{H}_d = [0,1]^d$ is the design space which for convenience we take to be the unit hypercube in d-dimensions, and $\boldsymbol{\xi}$ is the vector of design variables.

Finding the global minimum of f in the design space is a natural goal, not needing much justification. The only reason why we are so often content with local minima is that finding them is a much simpler problem. To see why consider the problem of verifying that $\bar{\boldsymbol{\xi}}$ is a local minimum. It suffices to consider f in a small ball in the design space close to $\bar{\boldsymbol{\xi}}$. Contrast with the problem of verifying that $\bar{\boldsymbol{\xi}}$ is a global minimum. We must simultaneously consider f in the entire design space. This makes such a big difference because of the respective volumes of these two spaces. The volume of a hypersphere of radius r in d-dimensions is

$$V_d(r) = \frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2}+1\right)} r^d,$$

where $\Gamma(\cdot)$ is the gamma function. The volume of the *largest* hypersphere embedded in $\mathcal{H}_d$ is therefore $V_d(\frac{1}{2})$. The volume of $\mathcal{H}_d$ itself is 1. These volumes are plotted on a log scale for varying dimension in Figure 1, and we see that the hypersphere is a dramatically reducing portion of $\mathcal{H}_d$ as d increases. This may be interpreted as saying that most of the volume is at the corners (and $\mathcal{H}_d$ has 2^d corners), or that most points in $\mathcal{H}_d$ are far away from each other, or that high-dimensional spaces are much bigger than our intuition suggests.

There are a variety of efficient algorithms to find local minima even in very high dimensions by exploiting derivatives of f. Here we show how derivatives may be used in global optimization by way of surrogates — increasing the dimension d at which global optimization is practical.

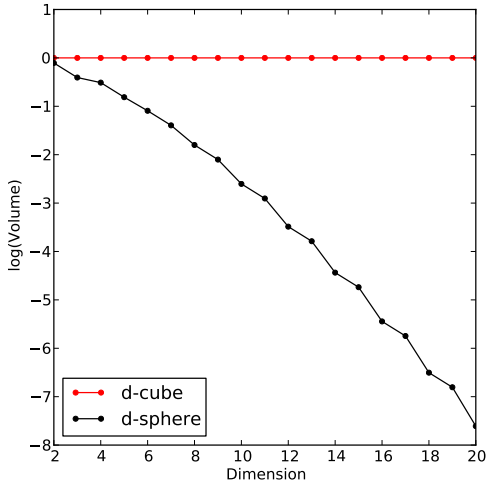


Figure 1: Curse of dimensionality illustrated. Volume of unit hyper-cube compared to unit hyper-sphere for varying dimension.

1.1 Surrogate modelling approaches

Given that it is necessary to search the entire space for the global optimum, it seems reasonable to build a cheap approximation to the cost function over the entire design space, and improve that approximation with additional samples from the cost function where necessary. The approximation we call the *response surface* or *surrogate model* interchangeably. *Adaptivity* is the process of strategically adding samples to this surrogate in order to improve the approximation of the minimum. If a cheap surrogate exists, for the purposes of illustration we assume that finding the global minimum is a straight-forward and computationally cheap problem — which may not always be justified for deliberately pathological test-functions such as those discussed in Section 1.2, but is usually true for optimization problems arising in engineering. This is especially true since gradients of surrogates are usually available, accurate and cheap.

Surrogates are standard in optimization. Newton's method

$$\boldsymbol{\xi}_{n+1} = \boldsymbol{\xi}_n - [f''(\boldsymbol{\xi}_n)]^{-1} f'(\boldsymbol{\xi}_n)$$

can be regarded as approximating the cost function by a quadratic surrogate consisting of a 3-term Taylor expansion (a quadratic form) about the previous iterate:

$$f(\boldsymbol{\xi}_n + \Delta\boldsymbol{\xi}_n) \approx f(\boldsymbol{\xi}_n) + f'(\boldsymbol{\xi}_n)\Delta\boldsymbol{\xi}_n + \frac{1}{2!}\Delta\boldsymbol{\xi}_n^T f''(\boldsymbol{\xi}_n)\Delta\boldsymbol{\xi}_n,$$

where $\Delta\boldsymbol{\xi}_n := \boldsymbol{\xi}_{n+1} - \boldsymbol{\xi}_n$, and then finding the (unique) minimum of this surrogate assuming it exists. Quasi-Newton Trust-Region (QNTR) methods use the same surrogate, but an approximation of f'' in terms of multiple evaluations of f'. The conjugate gradient method is similarly based on the assumption that locally f is approximately a quadratic form. These are indeed excellent local approximations for smooth f close to a minimum, and are cheap approximations in high-dimensional design spaces. However finding global optima clearly requires approximations that are non-local. In this work we consider much

more general and flexible surrogate models (in particular Kriging) that are able to interpolate (or sometimes regress) *all* evaluations of the cost function made at all sample locations so far in the course of the optimization.

Methods that can find global minima with some reliability, e.g. genetic algorithms and differential evolution, are capable of sampling distant regions of the design space simultaneously, as well as concentrating on local regions which show "promise". This allows a comparison of both distant points, and closely co-located points. Our goal when building a response surface for finding the global maximum should attempt to do the same thing: achieve a balance between sampling globally and locally. Clearly some kind of adaptive sampling strategy is required, but what exactly should it look like? The answer in the context of this tutorial is *efficient global optimization*[1] (EGO) Jones et al. (1998). The basic principle of EGO, which will be elucidated in the following is:

> *We add samples at locations in the design space where the* expected improvement *in the current best minimum is greatest.*

Expectation needs to be quantitatively defined, and this is done probabilistically.

1.2 Test functions for global optimization

To give examples of the kind of cost functions we are interested in, and to provide test-cases for our algorithms, we consider the following test functions.

Viermin function A d-dimensional function defined on $\boldsymbol{\xi} \in [-6, 6]^d$ by

$$f_v(\boldsymbol{\xi}) = 0.01 \sum_{i=1}^{d} \left[\left(\xi_i + \frac{1}{2} \right)^4 - 30\xi_i^2 - 20\xi_i \right]$$

It possesses a unique global minimum at

$$\boldsymbol{\xi}_i = -4.4537713 \quad \forall i \in \{1, \ldots, d\}$$

at which f_v takes the value -2.61638. Plotted in Figure 2.1, the function is representative of a problem that might arise in engineering for which local optimization is inadequate.

Six-hump camel-back function A 2d function defined on $\boldsymbol{\xi} \in [-3, 3] \times [-2, 2]$ by

$$f_c(\xi_1, \xi_2) = \left(4 - 2.1\xi_1^2 + \frac{1}{3}\xi_1^4 \right) \xi_1^2 + \xi_1\xi_2 + (-4 + 4\xi_2^2)\xi_2^2.$$

It possesses two global minima, at

$$\boldsymbol{\xi} = (-0.0898, 0.7126) \quad \text{and} \quad \boldsymbol{\xi} = (0.0898, -0.7126),$$

at which f_c takes the value -1.0316. Plotted in Figure 2.2, the function is representative of a problem that might arise in engineering for which local optimization is inadequate.

[1] Perhaps unfortunately this has become the name of the specific algorithm — which is at least a testament to its effectiveness.

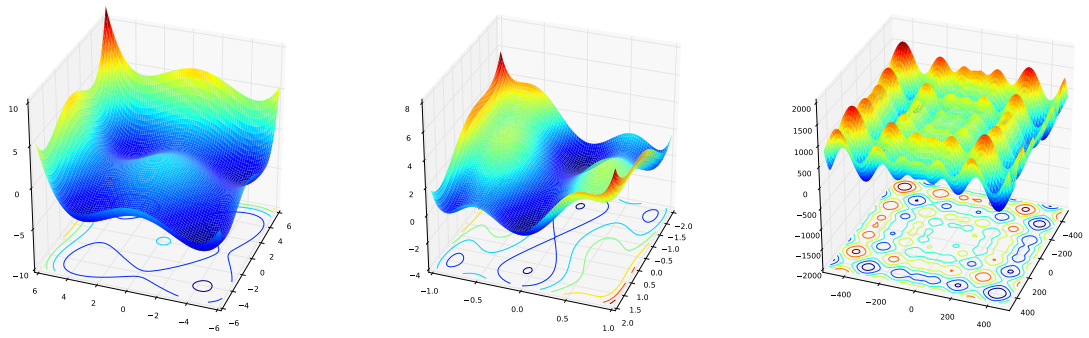
Schwefel function A d-dimensional function defined on $\boldsymbol{\xi} \in [-500, 500]^d$ by

$$f_s(\boldsymbol{\xi}) = 418.9829d - \sum_{i=1}^{d} \xi_i \sin \sqrt{|\xi_i|}.$$

There is a unique global minimum at $\boldsymbol{\xi} = (1, 1, \ldots, 1)$, taking the value $f_s = 0$. Plotted in the 2d case in Figure 2.3, the Schwefel functions is pathological, unlikely to be encountered in practice[2], but useful as a test of the behaviour of algorithms in the worst case. Python code:

```
import numpy as np
def schwefel(xi):
    """
    Schwefel test function in d-dimensions.
      xi - array of coordinates, size d
    """
    d = xi.size
    return 418.9829*d - np.sum(xi * np.sin(np.sqrt(np.abs(xi))))
```

The Python module `algopy` implementing algorithmic differentiation, was used to automatically obtain gradients and Hessians of these functions.



2.1: 2.2: 2.3:

Figure 2: Test functions for global optimization: (a) Viermin, (b) Camel, (c) Schwefel.

1.3 Outline

Our ultimate goal is a presentation of EGO from a probabilistic (in fact Bayesian) perspective. To accomplish this we briefly introduce Bayesian inference in Section 2, followed by a Bayesian derivation of the Kriging surrogate model in Section 3. Surrogate modelling requires sample points, non-adaptive and adaptive sampling methods are introduced in Section 4, again with an emphasis on a probabilistic background. With these technologies we are well equipped to describe EGO in a straightforward manner, Section 5. Appendix A offers an overview of the Kriging literature, for those interested in further reading.

[2]Actually this is not entirely true, we will see similar functions in Section 4 when trying to find maxima of surrogate error estimates for adaptation — but there finding the global maximum is not very critical.

1.4 Computer codes

The code listings contained in this document are available at `https://aerodynamics.lr.tudelft.nl/~rdwight/vki`. They may be slightly modified in minor details, and additional functionality is provided. Code examples given in this document all run with Python $\geq$2.7 and $\geq$3.5, and utilize the `numpy` package version $\geq$1.7.

In Matlab, a robust, widely used, free and open Kriging code is `DACE` (`http://www2.imm.dtu.dk/projects/dace/`), but does not implement cokriging with gradient information, which we discuss in Section 3.2. Our own simple Matlab and Python codes associated with our research work are available at `https://aerodynamics.lr.tudelft.nl/~bayesiancomputing/`.

2 Conditional Probability and Bayesian Inference

In order to derive a rigorous definition of *expected improvement* as needed by EGO, we require some tools from statistics and probability.

The meaning of probability is most clear when used to describe the frequency of possible outcomes in a repeatable experiment — for example throwing a 6-sided die. In this setting probabilities can be established quantitatively to any given accuracy by repeating the experiment sufficiently many times; they are objective. We however also naturally think in terms of probabilities for events which are most definitely not repeatable. A classical example is the probability that the sun will rise tomorrow morning. Most people would agree that the probability is very close to – but not quite – unity: $\rho(\text{sun rises}) = 1 - \epsilon$. But the sun can only rise (or not rise) one time tomorrow, so what exactly do we mean by this estimate of "probability"? Usually we are talking about our confidence in a particular outcome; we possess some lack-of-knowledge about the unique truth which we are attempting to subjectively quantify with a probability. Sometimes we have an excellent basis for assigning a subjective probability, for example a 6-sided die rolled behind a screen. The outcome is fixed but we don't know it, we would still quite reasonably assign a probability of $\rho(\text{die} = 1) = 1/6$. After looking behind the screen however we would update our assessment of the probabilities on the basis of new data. Hence the probability is not a property of the physical system itself (which is not random after the die has been cast), but of our own knowledge of the system – hence subjective. In many other cases, in the absence of any data, we can do no better than an expert- or marginally-informed guess, as for the sun rising tomorrow. Probability representing lack of knowledge is called *epistemic*, that representing real physical variability *aleatory*. In the following all probabilities represent epistemic uncertainty.

Bayesian inference is a procedure for updating our estimates of our epistemic uncertainty given observations of the system — i.e. updating probability distributions for uncertain variables given data. Given two random variables A and B, we write the probability of A taking in the value a as $\rho_A(a)$ or just $\rho(a)$ for convenience. If A and B are independent events ($A \perp B$) then

$$\rho_{A,B}(a \cap b) = \rho_A(a)\rho_B(b) \text{ for } A \perp B,$$

where $\rho_{A,B}(a \cap b)$ is the probability that A takes the value a *and* B takes the value b. But

more generally Bayes' theorem tells us that for dependent variables

$$\rho_{A,B}(a \cap b) = \rho_A(a|B=b)\rho_B(b) = \rho_B(b|A=a)\rho_A(a),$$

where $\rho_A(a|B=b)$ is the conditional probability that A takes the value a, given that B takes the value b. Using this relation we can "invert" probabilities:

$$\rho_A(a|B=b) = \frac{\rho_B(b|A=a)\rho_A(a)}{\rho_B(b)}$$

i.e. find A conditional on B given B conditional on A. Note that A can be either a discrete or continuous random variable. In the continuous case $\rho(a)$ is a probability density function (pdf) with

$$\int_{-\infty}^{\infty} \rho(a)\,\mathrm{d}a = 1,$$

for example a normal distribution with mean μ and standard-deviation σ. We write:

$$A \sim \mathcal{N}(\mu, \sigma^2)$$

and

$$\rho_A(a) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{\frac{(a-\mu)^2}{2\sigma^2}\right\}.$$

In applying Bayes' theorem to engineering we are usually interested in continuous variables, and in updating estimates of parameters given data. Consider a physical system parameterized by continuous parameters $\boldsymbol{\alpha}$, with an observable output $\mathbf{d}$ (data). There exists some true values of both $\boldsymbol{\alpha}$ and $\mathbf{d}$ but we don't know them precisely (we can't measure $\mathbf{d}$ perfectly and we can't measure $\boldsymbol{\alpha}$ at all). Bayesian inference gives us a means obtain estimates of $\boldsymbol{\alpha}$ as follows:

1. Define a *prior*: a pdf on $\boldsymbol{\alpha}$ encoding all information which is known about these parameters prior to observing the data. We denote this pdf $\rho_0(\alpha)$. For example if it is known from experience that a parameter *certainly* takes a value between 0.9 and 1.1, then a suitable prior might be a uniform distribution on $[0.9, 1.1]$. There is always some arbitrariness in the choice of prior – Bayesian updating is fundamentally subjective.

2. Describe the relationship between $\boldsymbol{\alpha}$ and $\mathbf{d}$ to come up with an estimate of the *likelihood function* $L(\mathbf{d}) := \rho(\mathbf{d}|\boldsymbol{\alpha})$ – i.e. the probability of observing the data $\mathbf{d}$ *given that* the true value of the parameters is $\boldsymbol{\alpha}$. In order to estimate the likelihood we have to relate $\boldsymbol{\alpha}$ to $\mathbf{d}$, and we do this using a theoretical model, either in the form of an analytic formula, an empirical relationship, or a simulation code. Let the model be $m(\boldsymbol{\alpha})$, returning an theoretical estimate of $\mathbf{d}$ for given $\boldsymbol{\alpha}$.

   Now we can not expect that $m(\boldsymbol{\alpha}) = \mathbf{d}$, even if we know the true $\boldsymbol{\alpha}$, because of measurement noise and error in the model. We can account for these error with a *statistical model*:

$$\mathbf{d} = m(\boldsymbol{\alpha}) + \delta(\boldsymbol{\alpha}) + \epsilon$$

where ϵ is a random variable accounting for experimental noise, typically an unbiased normal distribution:

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

and $\delta(\cdot)$ a random variable accounting for model error. Since model error is difficult to quantify, usually this is neglected and we assume:

$$\mathbf{d} = m(\boldsymbol{\alpha}) + \epsilon.$$

Given which

$$\rho(\mathbf{d}|\boldsymbol{\alpha}) := \rho_\epsilon(\mathbf{d} - m(\boldsymbol{\alpha})),$$

where $\rho_\epsilon(\cdot)$ is the pdf of ϵ.

3. For given prior and likelihood, Bayes' theorem gives an explicit expression for the *posterior*, the probability of parameters given the observed data:

$$\rho(\alpha|\mathbf{d}) \propto \rho(\mathbf{d}|\alpha)\rho_0(\alpha),$$

where the constant of proportionality (independent of α) is not usually of interest. In the Bayesian framework this posterior is regarded as the answer to the question: What is known about α?, and is therefore the updated estimate of the parameters, the solution of the inverse problem. If we require a deterministic estimate of α – rather than the probabilistic posterior – then a reasonable choice is the most-likely value of α, the maximum *a posteriori* estimate (MAP estimate):

$$\alpha_{\text{MAP}} := \underset{\alpha}{\operatorname{argmax}}\, \rho(\alpha|\mathbf{d}).$$

Although Bayes' gives an explicit expression for the posterior probability of any $\boldsymbol{\alpha}$, this may be expensive to evaluate as each sample involves a run of the simulation code. Even for cheap models it may still be difficult to extract useful information from the posterior for high-dimensional $\boldsymbol{\alpha}$. The MAP estimate is one way. To get more information, often we represent the posterior by samples which can be obtained with Gibbs sampling, or Markov-chain Monte-Carlo (McMC) numerical methods.

3 Surrogate modelling with Kriging

We derive a Kriging from Bayesian perspective. Without loss of generality consider constructing a surrogate model on the d-dimensional unit hypercube $\mathcal{H}_d$. Assume we are interested in the value of a function

$$f(\boldsymbol{\xi}) : \mathcal{H}_d \to \mathbb{R},$$

at n locations $\boldsymbol{\xi}_i \in \mathcal{H}_d$ for $i \in \{1, \ldots, n\}$. At m of these locations we actually observe the value of f, at the remaining $n - m$ locations we want to predict it. Let $\mathbf{x} = (x_1, \ldots, x_n)$ be the value of our (scalar) predictions at all n locations, and let $\mathbf{y} = (y_1, \ldots, y_m)$ be the observed values of f at m locations. Thus in the notation of the previous section our

parameters $\boldsymbol{\alpha} := \mathbf{x}$ (these are the quantities we want to estimate), and our data $\mathbf{d} := \mathbf{y}$. However now we also have some spatial relationship between the variables, each x_i being associated with a location $\boldsymbol{\xi}_i$.

Following the Bayesian inference recipe of the previous section, our first task is to define a prior on $\mathbf{x}$. This should reflect our beliefs about f in the absence of knowledge of $\mathbf{y}$. Now one might say that generally we know nothing about f, but in practice this is rarely true. In the optimization context f is a cost function resulting from a physical problem, and there is usually advance knowledge on the range of output we expect. So we might consider a prior like:

$$x_i \sim \mathcal{N}(\mu_x, \sigma_x^2) \quad \forall i, \tag{1}$$

with μ_x and σ_x based on expert knowledge, where we would be wise to be conservative and choose σ_x large. However we also expect a certain smoothness of f as $\boldsymbol{\xi}$ varies.[3] With the prior of (1) two closely co-located x_is can take very different values with high probability. In order to incorporate smoothness we have to introduce some correlation between x_is in proportion to their distance. This is achieved by a multivariate normal distribution for all $\mathbf{x}$ simultaneously

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}_x, P) \tag{2}$$

with pdf

$$\rho(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}\sqrt{\det P}} \exp\left\{\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_x)^T P^{-1}(\mathbf{x}-\boldsymbol{\mu}_x)\right\} \tag{3}$$

where $\boldsymbol{\mu}$ is now a vector of means, and P is the $n \times n$ covariance matrix.[4] Very roughly speaking the entry P_{ij} determines the correlation between x_i and x_j, larger implying more direct correlation, and $P_{ij} = 0$ implying no direct dependance. For example if $\boldsymbol{\mu}_x = \mu_x \mathbf{1}$ and $P = \sigma_x^2 I$ then (2) reduces to (1). For smoothness we expect that P_{ij} is large when $h_{ij} := \|\xi_i - \xi_j\|$ is small, and small when h_{ij} is large.[5] A reasonable choice is

$$P_{ij} = \sigma_x^2 r(\|\boldsymbol{\xi}_i - \boldsymbol{\xi}_j\|)$$

where

$$r(h) := \exp\left\{-\theta h^2\right\}$$

and θ is the correlation range. With increasing distance correlation gets weaker. For small theta, even distant points are tightly correlated, for large theta, even neighbouring points are weakly correlated. Now that the prior is fully specified we can draw random samples from it. In 1d these are plotted in Figure 3 for various θ. These samples can be viewed as some of the possible surrogates that will be constructed by the method.

The next step is to describe the relationship between variables and observations. In the Section 2 a computational model was necessary to relate $\boldsymbol{\alpha}$ to $\mathbf{d}$. In Kriging we directly observe a subset of our parameters $\mathbf{x}$ - which take the value $\mathbf{y}$, and therefore the "model" must simply selects those x_i that we do observe. Without loss of generality assume we

[3] Without *some* smoothness global optimization is hopeless.

[4] For (3) to be a pdf it must have integral 1 over $\mathbb{R}^n$, which implies that P^{-1} and hence P must be strictly positive definite (otherwise the integral would be infinite).

[5] There is a choice of metric $\|\cdot\|$ to be made - in the following we used the Euclidian distance.

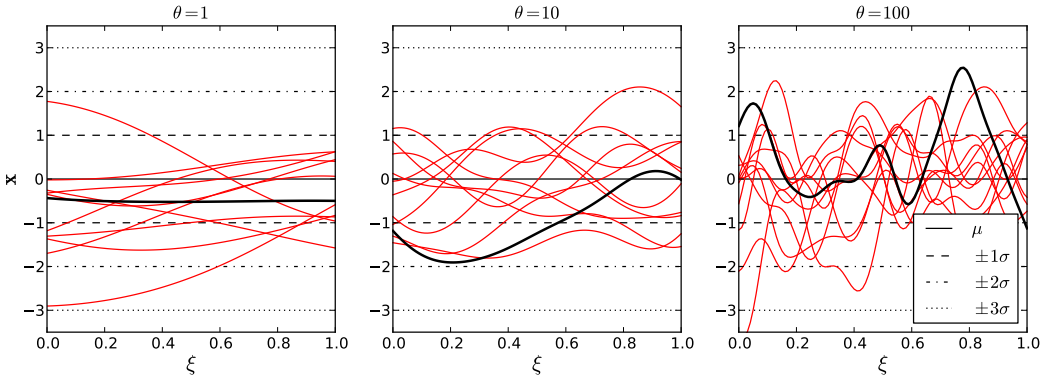


Figure 3: Random samples from the prior (2) with $\boldsymbol{\mu} = 0$, $\sigma = 1$, and 3 values of correlation length. 11 samples per plot, first sample in black.

observe the first m x_is. The the *observation operator* H which maps $\mathbf{x}$ to the observations is then defined by the $m \times n$ matrix

$$H_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\},$$

and if we had perfect confidence in our observations our statistical model would be

$$\mathbf{y} = H\mathbf{x},$$

where clearly no modelling error is present. Alternatively we might consider our observations to be noisy, with some additive noise, giving the relationship between our predictions and observations:

$$\mathbf{y} = H\mathbf{x} + \epsilon, \tag{4}$$

where

$$\epsilon \sim \mathcal{N}(0, \sigma_y^2).$$

We assume σ_y is known. This results in a mechanism with which we can introduce a controlled amount of regression into the surrogate. Even if $\mathbf{y}$ results from a computer code it can contain errors that can be construed as noise. Errors due to lack of convergence, a steady approximation to an unsteady flow, etc. Of course many errors in numerical simulation are certainty not unbiased (mean 0), for example discretization error. However given an output *corrected* for discretization error (with e.g. an *a posteriori* error estimate), the remaining error might be considered unbiased.[6] Given (4) we can write

$$\mathbf{y}|\mathbf{x} \sim \mathcal{N}(H\mathbf{x}, R) \tag{5}$$

where $R = \sigma_y^2 I$ the covariance matrix of observational noise.

[6]Remember we are trying to represent the state of our knowledge with probability. So if we can't guess whether the actual error is more likely to be +ve or -ve, then it's reasonable to say that it is unbiased. This is not to say that we must restrict ourselves to unbiased (or Gaussian, or additive) choices for ϵ. We can choose any distribution in the statistical model and prior, but the expression for the posterior is only analytically available for certain choices.

Finally we combine the prior (2) and likelihood (5) with Bayes' theorem

$$\rho(\mathbf{x}|\mathbf{y}) \propto \rho(\mathbf{y}|\mathbf{x})\rho_0(\mathbf{x})$$

to obtain the posterior. Thanks to our use of multivariate Gaussians in the prior and statistical model, the posterior is also multivariate Gaussian:

$$\mathbf{x}|\mathbf{y} \sim \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\Sigma})$$

with mean

$$\hat{\boldsymbol{\mu}} := \mathbb{E}(\mathbf{x}|\mathbf{y}) = (H^T R^{-1} H + P^{-1})^{-1}(HR^{-1}\mathbf{y} - P^{-1}\boldsymbol{\mu}_x) = \boldsymbol{\mu}_x + K(\mathbf{y} - H\boldsymbol{\mu}_x), \quad (6)$$

and covariance

$$\hat{\Sigma} := \mathrm{Cov}(\mathbf{x}|\mathbf{y}) = (H^T R^{-1} H + P^{-1})^{-1} = (I - KH)P, \quad (7)$$

where K is the *Kalman gain matrix*

$$K := PH^T(R + HPH^T)^{-1}$$

and the second equalities in both (6) and (7) are the result of basic linear algebra. Thus at each location ξ_i we approximate f by a Gaussian distribution with mean $\hat{\mu}_i$ and standard-deviation $\sqrt{\hat{\Sigma}_{ii}}$. The latter is a local estimate of the error in the surrogate, and will be small close to observed locations, and large in between. For applications see Figures 4, 5 and 6.

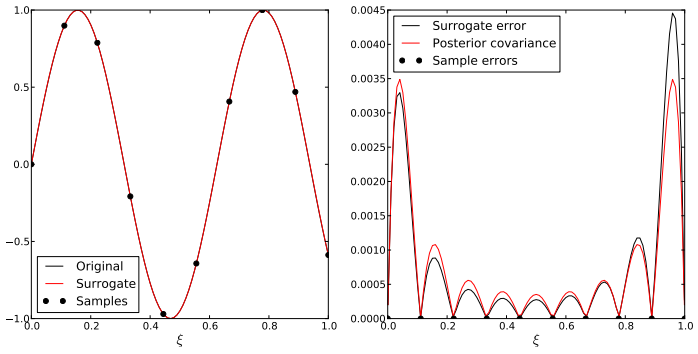


Figure 4: Kriging reconstruction of sine wave with 10 samples. Comparison of true error in the surrogate with 1σ of posterior process.

Note that in the above we have assumed knowledge of the mean and standard-deviation of $f(\cdot)$ on $\mathcal{H}_d$, as well as its correlation length θ. This case is sometimes called *simple*-Kriging. All three parameters may be computed given sufficient samples of f, though the correlation length may vary spatially. They are however unavailable in practice. Fortunately $\hat{\mu}$ is very insensitive to μ_x and σ_x, and only moderately sensitive to θ. On the other hand $\hat{\Sigma}$ is moderately sensitive to both θ and σ_x. We discuss standard approaches for estimating these hyperparameters in Section 3.3. This results in *ordinary*-Kriging.

Python code implementing the above algorithm is as follows:

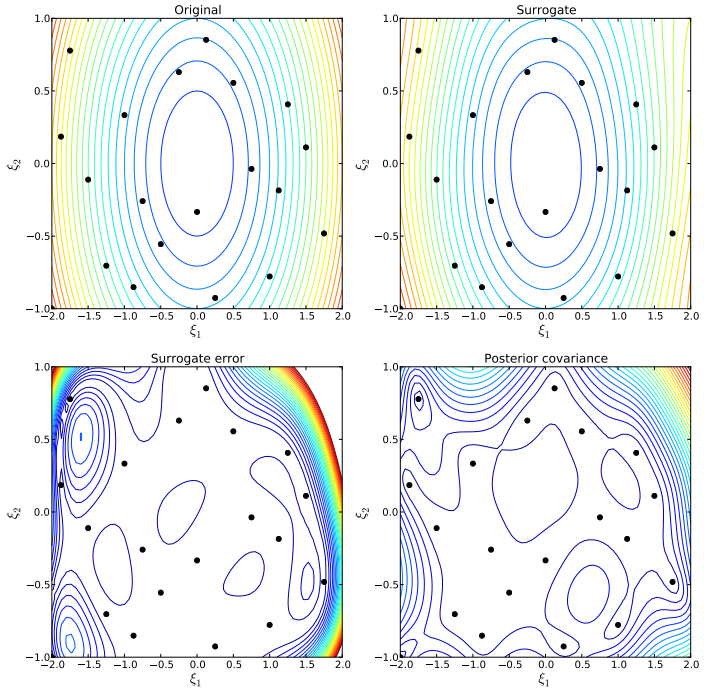


Figure 5: Kriging reconstruction of a (smooth) parabola in 2d from 20 samples.

```
def r(xi_i, xi_j, theta):
    """Gaussian covariance function - xi_i, xi_j coordinates in
    d-dimensions.  Euclidian norm.  Return scalar."""
    return np.exp(-theta * np.sum((xi_i - xi_j)**2))

def kriging(xi, x, observed, theta, sigma_d, mu):
    """
    Kriging in d-dimensions for a single variable - following the
    Bayesian derivation and notation.  The following assumptions are
    limitations of the current implementation which may be lifted
    without substantially modifying the derivation (just coding).

    Assumptions:
      - Constant regression at specified mean mu.
      - Same constant error for all observations (sigma_d)
      - Stationarity of the Gaussian process (constant standard
        deviation of the prior).

    Arguments:
      xi       - Array of sample locations (both observations and predictions).
                 Array (n x d).
      x        - Sample values (values not at observation locations are not used).
                 Array (n).
      observed - Bool array specifying which values are observed.  Array (n),
                 True - observed, False - not observed.
      theta    - Correlation coefficient in each direction.  Scalar.
      sigma_d  - Standard-deviations of observation error.  Scalar.
      mu, std  - Mean, standard-deviation of the approximated function.  Scalar.
      Return   - Dictionary of prior and posterior statistics.
    """
    ### Determine problem dimensions from input.
    n, d = xi.shape                     # Number of samples, design-space dimension
    H    = np.identity(n)[observed]     # Observation operator
    y    = np.dot(H, x)                 # Observations
```

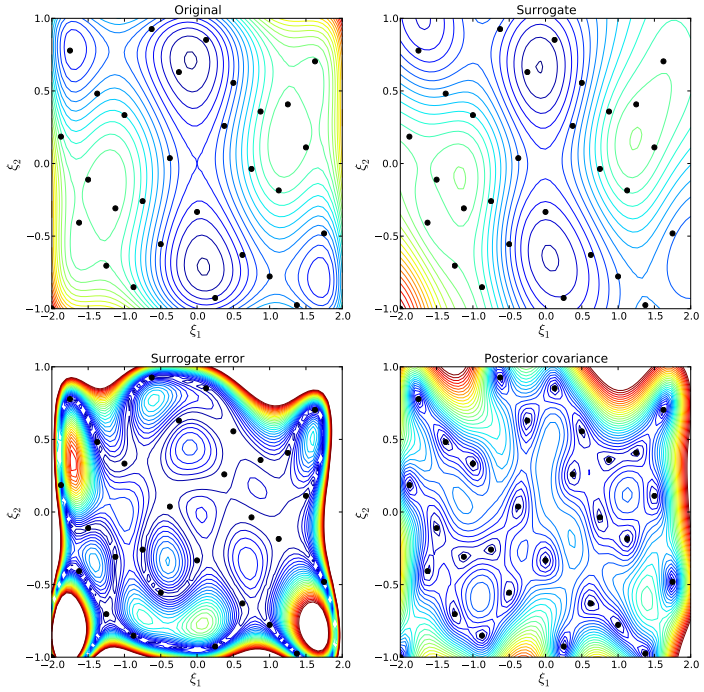


Figure 6: Kriging reconstruction of the camel function from 30 samples.

```
m    = y.size                          # Number of observations

### Observation error covar matrix
R = np.diag(np.ones(m) * sigma_d)

### Prior mean and covariance at the sample locations.
mu_prior = mu*np.ones(n)
P = np.zeros((n,n))
for i in range(n):
    for j in range(n):
        P[i,j] = r(xi[i], xi[j], theta)
P *= std**2

### The gain matrix.
A = R + np.dot(H, np.dot(P, H.T))
K = np.dot(P, np.dot(H.T, np.linalg.inv(A)))

### Posterior mean and covariance (prediction):
#   E(x|y) ("predictor")
mu_post  = mu_prior + np.dot(K, y - np.dot(H, mu_prior))
#   Cov(x|y) ("mean-squared error estimator")
cov_post = np.dot(np.identity(n) - np.dot(K, H), P)

### Return all this statistical information.
return {'mu_prior': mu_prior, 'cov_prior': P,        # Prior
        'mu_post': mu_post,   'cov_post': cov_post}  # Posterior
```

Please note that this code (like all herein) is intended as a minimal, simple, reference implementation with a view to exposition, not as a practical code for large `d` and `n`. In particular there is no error checking. Apart from checking input arguments, one should verify that `A` is positive definite in floating-point arithmetic (it will always be positive definite in exact arithmetic). In terms of efficiency there are many optimizations possible. Most impor-

tantly the prior and posterior covariance matrices must be formed *only* for those sample locations which are observed - not both those observed and predicted as above. The direct inverse of A should be replaced by a Cholesky factorisation, or better an Algebraic Multigrid (AMG) solve, as A^{-1} is not needed explicitly. Beyond the scope of this tutorial are many other possible efficiency improvements, such as using a correlation function with compact support so that P is sparse, essential for very large numbers of observations. For better implementations see `DACE` (`http://www2.imm.dtu.dk/projects/dace/`) and `https://aerodynamics.lr.tudelft.nl/~bayesiancomputing/`.

In fact the most time consuming component of the above implementation for a moderate number of observations is the construction of P. This is due to the loop over matrix entries being written out explicitly. This is slow in Python with `numpy` operations on complete arrays are much faster (as in Matlab). For a vectorized implementation of `r()` see Appendix B. The next bottleneck is the evaluation of the posterior covariance. This is only needed in full if we intend to use it as the prior of the a following surrogate, for example when performing adaptation. For an error estimate only the diagonal of $\hat{\Sigma}$ is needed. Given these optimizations the code runs fairly fast.

3.1 Interpretation as Radial Basis Function Interpolation

Kriging may be regarded a generalisation of radial basis function (RBF) interpolation (a.k.a. volume splines). This is standard interpolation as a linear sum of basis functions. These functions are chosen to be radially symmetric about the sample points. As the reader with a numerical analysis background is more likely to be familiar with this approach to interpolation, its relationship to Kriging is briefly outlined here.

Definition 3.1 (Radial function)**.** *A function $\phi : \mathbb{R}^n \to \mathbb{R}$ is called radial provided there exists a univariate function $\varphi : [0, \infty) \to \mathbb{R}$ such that*

$$\Phi(\boldsymbol{\xi}) = \varphi(h), \quad h = \|\boldsymbol{\xi}\|, \tag{8}$$

and $\|\cdot\|$ is some norm on $\mathbb{R}^n$, usually the Euclidean norm. Hence, for two points $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$,

$$\|\boldsymbol{\xi}_1\| = \|\boldsymbol{\xi}_2\| \Rightarrow \Phi(\boldsymbol{\xi}_1) = \Phi(\boldsymbol{\xi}_2). \tag{9}$$

The interpolation problem using radial basis functions (RBF's) can then be formulated as follows: Given a set of n distinct data points $\{\boldsymbol{\xi}_i : i = 1, \ldots, n\}$ and data values $\{y_i : i = 1, \ldots, n\}$, the *RBF interpolant* is given by

$$s(\boldsymbol{\xi}) = \sum_{j=1}^{n} a_j \, \Phi(\|\boldsymbol{\xi} - \boldsymbol{\xi}_j\|), \tag{10}$$

where $\Phi(h),\ h \geq 0$ is some radial function. The coefficients a_j are determined from the interpolation conditions

$$s(\boldsymbol{\xi}_j) = y_j, \quad j = 1, \ldots, n, \tag{11}$$

which leads to the following symmetric linear system:

$$A\,\mathbf{a} = \mathbf{y}. \tag{12}$$

The entries of the matrix A are given by

$$A_{ij} = \Phi(\|\boldsymbol{\xi}_i - \boldsymbol{\xi}_j\|). \tag{13}$$

As in Kriging we have a choice of metric and φ. In RBF interpolation a common choice for $\varphi(h)$ is exactly

$$\varphi(h) = \exp\left\{-\theta r^2\right\}$$

given which A is guaranteed to be non-singular for any choice of $\boldsymbol{\xi}_i$, and therefore a unique interpolant exists. Compare (10), (12) with the Kriging posterior mean (6) for $\boldsymbol{\mu}_x = \mathbf{0}$.

3.2 Gradient Enhanced Kriging (GEK)

Kriging is useful, but as the dimension of the design space increases, the number of sample required increases dramatically. In order to delay the point at which building a surrogate becomes impractical we aim to exploit function derivate information in the construction. Derivatives in d-directions may be available at constant computational cost for even complex simulation codes thanks to adjoint methods. Therefore in higher dimensions we should get "more information" for free. The flexibility of Bayesian inference allows us to construct a surrogate with gradients, almost as easily as without.

In simple Kriging we assumed that the hyperparameters – the mean, variance, and correlation range contained in the prior – are known Cressie (1993); Chiles and Delfiner (1999). A Bayesian derivation of simple Kriging has been made at the start of this section. We extend this derivation to *simple GEK*, where we express the relation between the values and gradients in the prior covariance matrix. We later extend the method to *ordinary GEK* by estimating the hyperparameters in Section 3.3.

In the case of GEK we are interested in the $n(d+1)$ *compiled* values

$$\mathbf{x}_c = \left[\mathbf{x}, \frac{\partial \mathbf{x}}{\partial \xi_1}, \ldots, \frac{\partial \mathbf{x}}{\partial \xi_d}\right].$$

If we define some compiled prior mean $\boldsymbol{\mu}_{x,c}$ and covariance matrix P_c, then exactly as before we find the Kriging posterior mean:

$$\hat{\boldsymbol{\mu}}_c := \mathbb{E}(\mathbf{x}_c|\mathbf{y}_c) = \boldsymbol{\mu}_c + K_c(\mathbf{y}_c - H_c\boldsymbol{\mu}_c), \tag{14}$$

the posterior covariance:

$$\hat{\Sigma}_c := \mathrm{Cov}(\mathbf{x}_c|\mathbf{y}_c) = (I - K_c H_c)P_c, \tag{15}$$

with gain matrix

$$K_c := P_c H_c^T (R_c + H_c P_c H_c^T)^{-1}. \tag{16}$$

These are defined for the compiled variables, indicating that we are estimating the values and gradients simultaneously and both include variance information. Missing from the above derivation is the relationship between values and gradients. This will be enforced via the prior. In the remaining discussion of GEK we consider only the 1d case for simplicity.

One objective of this derivation was to include a concept of noise in value and gradient errors separately. The Bayesian perspective makes clear that these belong in the likelihood. Specifically, the matrix:

$$R_c = \begin{pmatrix} \epsilon_1^2 & & & & & \\ & \ddots & & & 0 & \\ & & \epsilon_n^2 & & & \\ & & & \nu_1^2 & & \\ & 0 & & & \ddots & \\ & & & & & \nu_n^2 \end{pmatrix}, \tag{17}$$

allows specification of observation errors ϵ_i and ν_i, for the values and the gradients individually.

The covariance relationship between values and gradients Here we follow Chung and Alonso (2002): so far we have been treating the values and the gradients as independent quantities. We will now see what relation we would actually like them to have and express this relation in the prior covariance. Our approach is to assume that there exists a function f, which provides the expected value of an observation:

$$\begin{aligned} f(\xi_i) &= \mathbb{E}\left[x_i - \mu_x\right], \\ \frac{\partial f(\xi_i)}{\partial \xi} &= \mathbb{E}\left[\frac{\partial x_i}{\partial \xi}\right]. \end{aligned}$$

In some cases, the function f might regarded as the "true process", although the existence of such a process is not always obvious. It is important to note that we do not need exact knowledge of this function f, we will only apply some of its properties. In GEK, the prior covariance matrix:

$$P_c = \begin{bmatrix} P_{00} & P_{10} \\ P_{01} & P_{11} \end{bmatrix}, \tag{18}$$

contains the following submatrices: P_{00} – the covariance matrix of the values, P_{11} – the covariance matrix of the gradients, and P_{01}, P_{10} – the cross-covariances. Using the definition of covariance, and the interchangeability of differentiation and expectation we have:

$$\begin{aligned} P_{00}^{ij} &:= \mathbb{E}\left[f(\xi_i) f(\xi_j)\right], \\ P_{10}^{ij} &:= \mathbb{E}\left[\frac{\partial f(\xi_i)}{\partial \xi_i} f(\xi_j)\right] = \frac{\partial}{\partial \xi_i}\mathbb{E}\left[f(\xi_i) f(\xi_j)\right], \\ P_{01}^{ij} &:= \mathbb{E}\left[f(\xi_i)\frac{\partial f(\xi_j)}{\partial \xi_j}\right] = \frac{\partial}{\partial \xi_j}\mathbb{E}\left[f(\xi_i) f(\xi_j)\right], \\ P_{11}^{ij} &:= \mathbb{E}\left[\frac{\partial f(\xi_i)}{\partial \xi_i}\frac{\partial f(\xi_j)}{\partial \xi_j}\right] = \frac{\partial^2}{\partial \xi_i \partial \xi_j}\mathbb{E}\left[f(\xi_i) f(\xi_j)\right]. \end{aligned} \tag{19}$$

Although these expressions are precise, we can not evaluate them as we do not have exact knowledge of f. In the following subsection we will approximate $\mathbb{E}\left[f(\xi_i) f(\xi_j)\right]$ with an approximate covariance function.

Approximate covariance function Instead of defining the covariances through $f(\xi)$ we will use an approximate covariance function. Consider the lag $h_{ij} := \xi_j - \xi_i$ and approximate

$$P_{00}^{ij} := f(\xi_i)f(\xi_j) \approx r(h_{ij}). \tag{20}$$

where we apply the same Gaussian covariance function is in Kriging:

$$r(h_{ij}) = \sigma^2 \exp\left(-\theta h_{ij}^2\right), \tag{21}$$

with correlation range θ and variance σ^2. Again, this is only an example of a possible covariance function, and that the choice of squared-exponential is not related to the normal distributions of the likelihood or the prior. This covariance function might in fact not be the most suitable for every application Stein (1999); Webster and Oliver (2007).

The approximation of P_{00} in (20) implies approximations for the other sub-matrices. Starting from (19) and using the chain rule we have:

$$\begin{aligned}
P_{00}^{ij} &= \mathbb{E}\left[f(\xi_i)f(\xi_j)\right] \approx r(h_{ij}),\\
P_{10}^{ij} &= \frac{\partial}{\partial \xi_i}\mathbb{E}\left[f(\xi_i)f(\xi_j)\right] \approx -\frac{\partial}{\partial h}r(h_{ij}),\\
P_{01}^{ij} &= \frac{\partial}{\partial \xi_j}\mathbb{E}\left[f(\xi_i)f(\xi_j)\right] \approx \frac{\partial}{\partial h}r(h_{ij}),\\
P_{11}^{ij} &= \frac{\partial^2}{\partial \xi_i \partial \xi_j}\mathbb{E}\left[f(\xi_i)f(\xi_j)\right] \approx -\frac{\partial^2}{\partial h^2}r(h_{ij}).
\end{aligned}$$

With these approximations we can construct the prior covariance matrix, given in (18), which completes our Bayesian derivation of simple GEK.

Example: GEK with observation errors Consider the following simple test function:

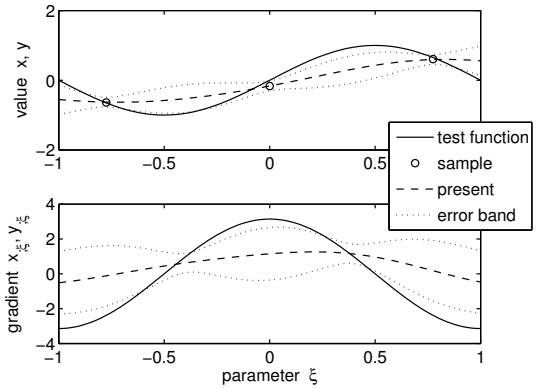
$$\begin{aligned}
x(\xi) &= \sin(\pi\xi),\\
\frac{\partial x}{\partial \xi} &= \pi\cos(\pi\xi).
\end{aligned}$$

on $\xi \in [-1, 1]$. We assume that the prior mean $\mu = 0$, the prior variance $\sigma^2 = 0.5$, and the prior correlation range $\theta = 0.5$ are known. We sample the test function at three nodes, with an observation error of $\epsilon = 0.2$ and $\nu = 0.2$.

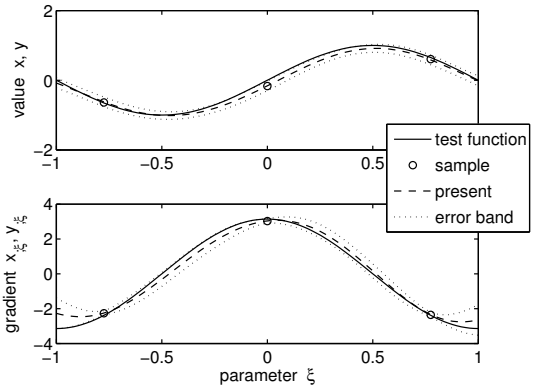
If we only sample the function values, we find the surrogate model shown in Figure 7.1. Note that due to the observation error the estimated variance in the value is not zero at the sample locations. If we sample both the values and the gradients, we find the surrogate model in Figure 7.2. We see that after including the gradient information the surrogate model resembles the function more closely, while the error band is reduced significantly.

Further applications GEK are shown in Figures 8, 9 an 10. Compare with the corresponding results for Kriging.

Most of the extra work in implementing GEK (compared to `kriging()`) is in bookkeeping: setting up the same vectors and matrices as before, extended with gradients. Given these extended objects the application of Bayes is identical to the gradient-free case. This implementation is correct for d-dimensions.



7.1:



7.2:

Figure 7: Simple Kriging and GEK surrogate models, obtained by sampling a test function. In (a) we only sample the values, while in (b) we sample both the values and the gradients.

```
def drdxi_i(xi_i, xi_j, theta):
    """Derivative of r(.) with respect to xi_i.  Return (d x 1)-vector."""
    flat = -2.*theta*(xi_i - xi_j) * np.exp(-theta * np.sum((xi_i - xi_j)**2))
    return np.array([flat]).T

def drdxi_j(xi_i, xi_j, theta):
    """Derivative of r(.) with respect to xi_j.  Return (1 x d)-vector."""
    flat = 2.*theta*(xi_i - xi_j) * np.exp(-theta * np.sum((xi_i - xi_j)**2))
    return np.array([flat])

def d2rdxi_idxi_j(xi_i, xi_j, theta):
    """2nd derivative of r(.) with respect to xi_i and xi_j.  Return (d x d) matrix."""
    d = xi_i.shape[0]
    outer = np.outer(xi_i - xi_j, xi_i - xi_j)
    return 2.*theta*(np.identity(d) - 2.*theta*outer) * \
           np.exp(-theta * np.sum((xi_i - xi_j)**2))

def gek(xi, x, dx, observed, theta, sigma_d, sigma_dg, mu):
    """
    Gradient-Enhanced Kriging (GEK) in d-dimensions for a single
    variable - following the Bayesian derivation and notation.

    Assumptions (as for kriging() and...):
      - Gradients observations collocated with value observations.
      - Gradients in all d directions observed at all locations.
      - Constant gradient error for all locations and directions.

    Constant regression at given mean mu, mean gradient assumed zero.
      d        - Dimension
      xi       - Sample locations (both observations and predictions).
                 Array (n x d).
      x        - Sample values (values not at observation locations are not used).
                 Array (n).
      dx       - Sample gradients.  Array (n x d).
      observed - Bool array specifying which values are observed.  Array (n).
      theta    - Correlation coefficient in each direction.  Scalar.
      sigma_d  - Standard-deviation of observation error.  Scalar.
      sigma_dg - Standard-deviations of observation error.  Scalar.
      mu       - Mean of the approximated function.  Scalar.
      Return   - Dictionary of prior and posterior statistics
    """
    ### Create extended variable vectors containing values then
```

```
### gradients.  The ordering used is:
### (x_1, x_2, ... x_n, dx_1/dxi_1, ... dx_1/dxi_d, dx_2/dxi_1, ..., dx_n/dxi_d)
### The total size is n*(d+1).
n, d = xi.shape                         # Number of samples, design-space
                                        # dimension
xc = copy.copy(x)                       # Extended sample values
for i in range(n): xc = np.hstack((xc, dx[i,:]))
observedc = copy.copy(observed)         # Extended observed array
for i in range(n): observedc = np.hstack((observedc, observed[i] *
                                          np.ones(d, dtype=bool)))
Hc = np.identity((d+1)*n)[observedc]    # Extended observation operator
yc = np.dot(Hc, xc)                     # Extended observation vector
m = yc.size // (d+1)                     # Number of observation locations

### Extended observation error covar matrix
Rc = np.diag( np.hstack((np.ones(m)   * sigma_d,
                         np.ones(m*d) * sigma_dg)) )

### Prior mean and covariance at the sample locations.
mu_prior = np.zeros((d+1)*n)  # Assume zero gradient mean
mu_prior[:n] = mu
Pc00 = np.zeros((n,  n))      # Top-left sub-matrix
Pc01 = np.zeros((n,  d*n))    # Top-right sub-matrix
Pc10 = np.zeros((d*n,n))      # ...
Pc11 = np.zeros((d*n,d*n))

for i in range(n):
    for j in range(n):
        Pc00[i,j]                   = r            (xi[i], xi[j], theta)
        Pc01[i,j*d:j*d+d]           = drdxi_j      (xi[i], xi[j], theta)
        Pc10[i*d:i*d+d,j:j+1]       = drdxi_i      (xi[i], xi[j], theta)
        Pc11[i*d:i*d+d,j*d:j*d+d]   = d2rdxi_idxi_j(xi[i], xi[j], theta)

### Build prior covariance matrix P from sub-matrices.
Pc = np.zeros(((d+1)*n, (d+1)*n))
Pc[:n,:n], Pc[:n,n:], Pc[n:,:n], Pc[n:,n:] = Pc00, Pc01, Pc10, Pc11
Pc *= std**2

### Now everything is exactly as before, with the extended vectors:
### The Kalman gain matrix K.
Ac = Rc + np.dot(Hc, np.dot(Pc, Hc.T))
Kc = np.dot(Pc, np.dot(Hc.T, np.linalg.inv(Ac)))

### Posterior mean and covariance (prediction):
#   E(xc|yc) ("predictor")
mu_post  = mu_prior + np.dot(Kc, yc - np.dot(Hc, mu_prior))
#   Cov(xc|yc) ("mean-squared error estimator")
cov_post = np.dot(np.identity(n*(d+1)) - np.dot(Kc, Hc), Pc)

### Return all this statistical information.
return {'mu_prior': mu_prior, 'cov_prior': Pc,        # Prior
        'mu_post': mu_post,   'cov_post': cov_post}  # Posterior
```

3.3 Estimating the correlation length θ, μ_x and σ_x

We will now extend our derivation to *ordinary GEK* following the standard approach Kitanidis and Lane (1985); Kitanidis (1986); Mardia (1989); Dwight and Han (2009). In this case we find the hyperparameters μ_x, σ_x, and θ by making *maximum a posteriori* estimates of these values, based on the observations $\mathbf{y}_c$. Assuming a 1d problem, define the $2n \times 2n$ matrix A_c as

$$A_c := R_c + H_c P_c H_c^T,$$

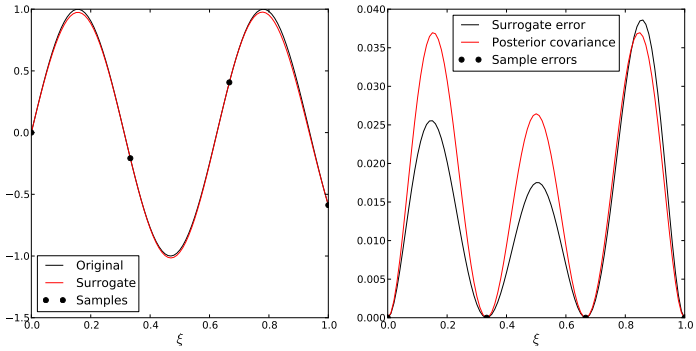


Figure 8: As for Figure 4 but with GEK and 4 samples of values and gradients.

and $A_0 = \sigma^{-2} A_c$. We estimate the hyperparameters by applying Bayes theorem with non-informative priors to identify μ_x, σ_x given $\mathbf{y}_c$ and θ. An analytic expression for the MAP estimator is possible in both cases, namely:

$$\begin{aligned} \mu_x &= \frac{\mathbf{1}_c^T A_0^{-1} \mathbf{y}_c}{\mathbf{1}_c^T A_0^{-1} \mathbf{1}_c}, \\ \sigma_x^2 &= \frac{(\mathbf{y}_c - \boldsymbol{\mu}_c)^T A_0^{-1} (\mathbf{y}_c - \boldsymbol{\mu}_c)}{2n-1}, \end{aligned} \tag{22}$$

where in the case of GEK the "drift" vector $\mathbf{1}_c$ has n entries equal to one followed by n entries equal to zero.

Unfortunately no similar analytic expression exists for the MAP estimate of θ. With a uniform prior belief

$$\theta \sim \mathcal{U}(\theta_{\min}, \theta_{\max})$$

we maximize the log likelihood

$$\ln p(\mathbf{y}_c | \mu, \sigma, \theta) = \frac{-2n - \ln(2\pi) - \ln|A_c| - (\mathbf{y}_c - \boldsymbol{\mu}_c)^T A_c^{-1} (\mathbf{y}_c - \boldsymbol{\mu}_c)}{2} + \text{constant}, \tag{23}$$

on the interval $[\theta_{\min}, \theta_{\max}]$ which is equivalent to minimising:

$$L(\theta) = \ln|A_c| + (\mathbf{y}_c - \boldsymbol{\mu}_c)^T A_c^{-1} (\mathbf{y}_c - \boldsymbol{\mu}_c), \tag{24}$$

subject to $\theta \in [\theta_{\min}, \theta_{\max}]$. This optimization problem must be solved numerically. Interestingly We need a global optimization algorithm to find the best correlation length. So in order to build our global optimization algorithm in dimension d, we need a global optimization algorithm in dimension 1.

Due to the estimating hyperparameters with the observations $\mathbf{y}$, the posterior given in (14) and (15) must be modified. For ordinary GEK the posterior at a location ξ changes from a normal distribution to a Student's t-distribution. The expression for the posterior mean remains unchanged, while the posterior variance should be multiplied with the factor Kitanidis and Lane (1985):

$$\frac{2n-q}{2n-q-2} = \frac{2n-1}{2n-3}, \tag{25}$$

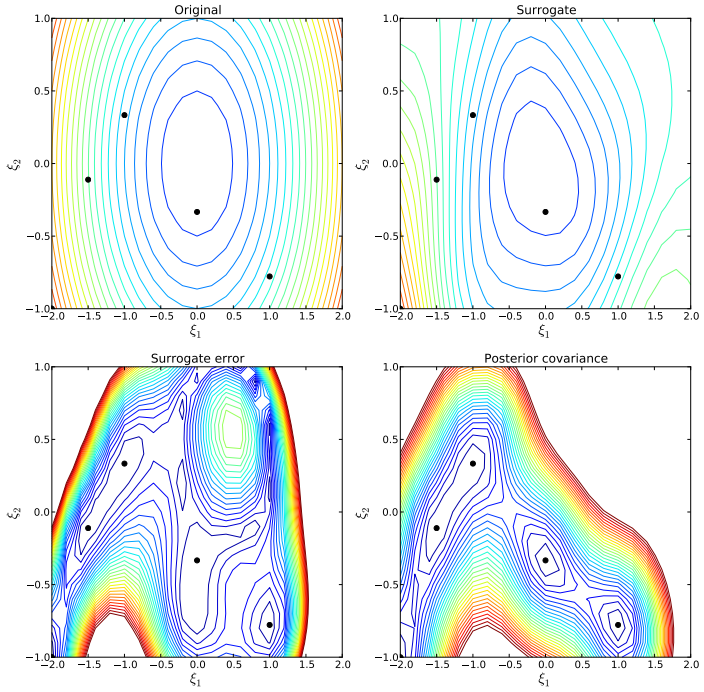


Figure 9: As for Figure 5 but with GEK and 4 samples of values and gradients.

where for ordinary GEK the number of drift coefficients $q = 1$, as μ_x is a scalar. We observe that these alterations are quite significant for a small number of observations n. Note that these alterations should also be accounted for in ordinary Kriging without gradients, otherwise for small n the variance of the prediction will be underestimated Kitanidis and Lane (1985); Bayarri et al. (2007).

4 Non-adaptive and Adaptive Sampling Techniques

Non-adaptive implies no exploitation of knowledge of the solution. Choose all samples in advance. Embarrassingly parallel.

Adaptive implies we have some *a posteriori* (depending on solution) estimate of surrogate error, some method of reducing it. Relation to spatial mesh adaptation. We can choose our error measure to estimate accurately whatever Quantity of Interest (QoI) we are interested in.

4.1 Non-adaptive sampling

Non-adaptive sampling is the practice of choosing support points for the surrogate model in the absence of any specific information on the function being approximated. It may be used in itself, or as the first step of an adaptive sampling scheme. The choice of support points is part of the study of *Design of Experiments* Hinkelmann and Kempthorne (2008), which is concerned with designing information-gathering exercises which are optimal with respect to knowledge gained. Here we consider sampling techniques in the the context of

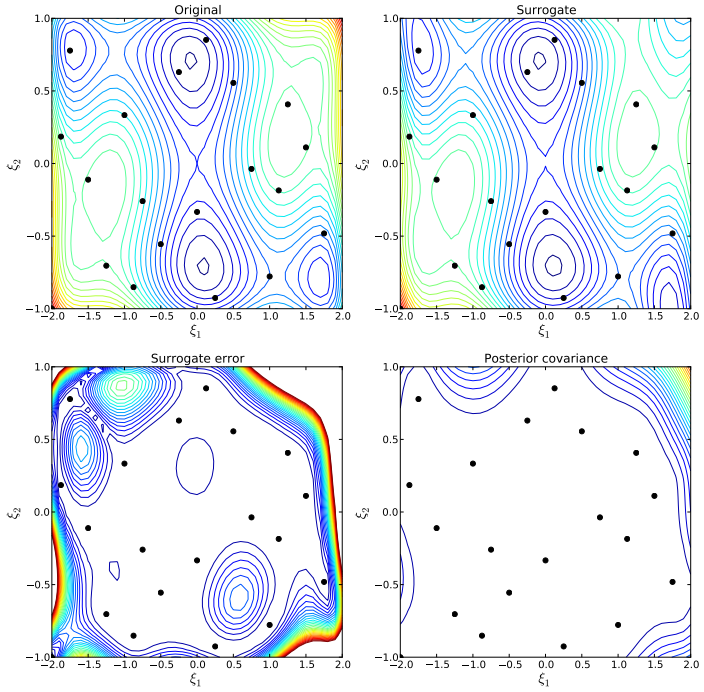


Figure 10: As for Figure 6 but with GEK and 20 samples of values and gradients. Posterior covariance curve shows 1σ.

Monte-Carlo simulation, which provides a useful framework for the discussion. Kriging does not restrict the locations of the samples, so we can choose a sampling scheme that is most informative.

The fundamental issue is that naive approaches to space sampling, e.g. tensor-product grids, or one-at-a-time sampling (where one variable is varied and all others held constant), do not minimize variance in approximations of rate-of-change of the output quantity with respect to multiple inputs Hinkelmann and Kempthorne (2008). Furthermore in d-dimensional spaces tensor-product grids require a minimum of 2^d support points, which may be too large to be practical. One-at-a-time sampling completely misses interactions between parameters. Random sampling therefore becomes more attractive than it might initially appear.

When discussing random and quasi-random sampling it is natural to examine their effectiveness in terms of Monte-Carlo (MC) and Quasi Monte-Carlo (QMC) methods[7]. We consider the case in which the domain of interest is the d-dimensional unit hypercube $\mathcal{H}$. Sampling on other regular and infinite domains can be obtained by transformation of variables. The simplest approach is to use samples drawn from a uniform distribution:

```
import numpy as np
def random_uniform(N, d):
    """Return N random coordinates on unit d-dimensional hypercube."""
    return np.random.random((N, d))
```

In this code the call `np.random.random((N, d))` returns an array of dimension `N` by `d` with each entry a pseudo-random number on the interval $[0, 1]$. This point selection can

[7]Which are nothing other than Monte-Carlo methods with samples drawn from a quasi-random sequence.

be regarded as Monte-Carlo sampling from a uniform random variable on a d-dimensional unit hypercube. If $\mathbf{x}_i \in [0,1]^d =: \mathcal{H}$ are the samples then:

$$\mathbb{E}_{\mathcal{H}} f := \int_{\mathcal{H}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \frac{1}{N}\sum_{i=0}^{N} f(\mathbf{x}_i) \tag{26}$$

with an error proportional to $\sigma/\sqrt{N}$, where σ is the standard-deviation of f over the domain $\mathcal{H}$:

$$\sigma^2 := \mathbb{E}_{\mathcal{H}}\left[f - \mathbb{E}_{\mathcal{H}} f\right],$$

independently of d. This latter property is unusual as it implies the method is immune to the curse of dimensionality. We reason that a sampling scheme that is able to approximate $\mathbb{E}f$ well, must also sample well information relevant to constructing a surrogate for f over all of $\mathcal{H}$ (assuming sufficient smoothness of f, and independently of a particular surrogate model). Therefore we propose to judge sampling schemes on the accuracy of their MC approximation of $\mathbb{E}f$. An example of the output of the above code for $d = 2$ is shown in Figure 11.

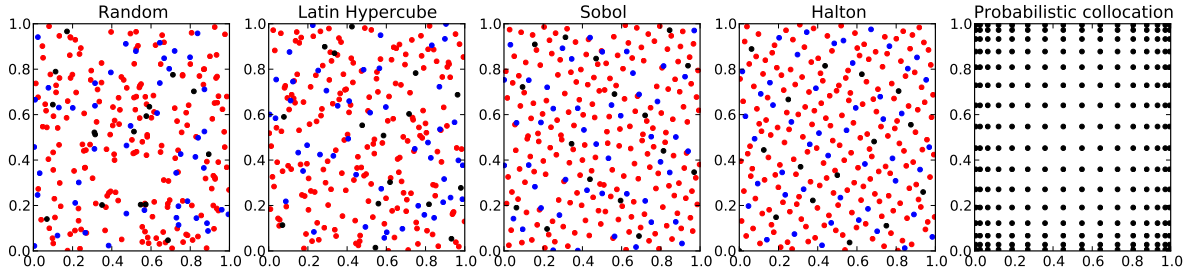


Figure 11: 256 samples on the unit square with 5 different techniques. Key: black, first 16; blue, first 64; red, remaining samples. Probabilistic collocation samples are not uniform on $\mathcal{H}$ and have no order.

Given the convergence result for random sampling, we would suppose random sampling is effective in an arbitrary dimensions — however in modest dimensions, as in Figure 11, we see that for finite sample sizes large gaps may be present, and suppose that our sampling would be better if it was less gappy. This motivates *Latin-Hypercube* sampling (LHS).

A Latin square is an $N \times N$ matrix with integer entries from the set $\{1, \ldots, N\}$, filled such that in each row and column of the matrix, each integer occurs exactly once. We consider related objects: $N \times N$ squares in which the entries come from $\{0, 1\}$, and where we require that each row and column has exactly one 1, the remaining entries being zero. There are $(N!)^2$ possible such squares, and we choose one at random with equal probability. The 1s corresponds to a sample at that location, thus we will obtain samples regularly across the full domain in each direction. To get the actual samples we divide the square $\mathcal{H}_2$ into an $N \times N$ uniform grid, and place a sample in a grid-cell iff the corresponding matrix entry is 1. The exact location of the sample within the cell is chosen pseudorandomly with a uniform distribution. Latin hypercube sampling is the natural generalisation of this strategy to d-dimensions. Note this scheme allows any choice of N and d, in particular there is no minimum N for a given d. It is not gappy in the sense that a 1-dimensional projection of the samples onto any of the principle directions will result in a roughly uniform sampling. A simple implementation of this scheme is as follows:

```python
def latin_hypercube(N, d):
    """Return N Latin Hypercube samples from a uniform distribution
    on the unit d-dimensional hypercube."""
    samples = np.zeros((N, d))
    for i in range(d):
        id = range(N)
        np.random.shuffle(id)
        for j in range(N):
            samples[j][i] = (np.random.random() + id[j]) / N
    return samples
```

In the code above the call `np.random.shuffle(id)` randomises the order of the entries in the array `id` in place, such that every one of the $N!$ possible orderings is equally likely. An example of the output of the above code for $d = 2$ is shown in Figure 11. The convergence of the MC integral approximation (26) is identical to random sampling $\mathcal{O}(\sigma/\sqrt{N})$, independently of dimension. However the constant of proportionality is usually more favorable for LHS. We still don't have a visually very "uniform" or "ungappy" sampling however.

We can formalize the requirement of reducing gappiness in sequences with the idea of discrepancy. The discrepancy of a sequence $s_N = (x_0, x_1, x_2, \ldots, x_N)$ on $[0, 1)$ (one-d) is defined as

$$D(s_N) := \sup_{0 \le a \le b \le 1} \left| \frac{1}{N} \left| \{x_0, \cdots, x_N\} \cap [a, b] \right| - (b - a) \right|,$$

where $|\{x_0, \cdots, x_N\} \cap [a, b]|$ counts the number of entries of s_N in the interval $[a, b]$. The discrepancy will be small provided we can not find any large, and mostly empty interval $[a, b]$, which would correspond to a gap. If

$$\lim_{N \to \infty} D(s_N) = 0,$$

then s_N is said to be equidistributed on $[0, 1)$, that is that samples occur within any interval in proportion to the interval width — as though we are performing uniform random sampling on $[0, 1)$. The important point is that we are not requiring any randomness of s_N, only the definition of the sequence for arbitrary N.

From this definition we begin look for quasi-random or *low discrepancy* sequences, which informally speaking are "less random" than pseudorandom sampling, but also "more uniform", and therefore more suitable for exploring $\mathcal{H}$. They will all have the property that the discrepancy tends to zero more rapidly than pseudorandom or LHS sampling, while maintaining the property expressed in (26), with a different rate of convergence. In fact a sequence is termed *low discrepancy* if

$$D(s_N) \le C \frac{(\log N)^d}{N}.$$

for some finite C. MC with samples taken as a low discrepancy sequence is called QMC, and the error in the approximation of the integral in (26) is

$$\mathcal{O}\left(\sigma \frac{(\log N)^d}{N}\right).$$

This result implies that QMC will be superior to MC for $N > e^d$ Some sequences with these properties are Faure, Halton, Niederreiter and Sobol sequences. For the Hammersley

sequence h_N there is slightly better convergence:

$$D(h_N) \leq C\frac{(\log N)^{d-1}}{N}.$$

As an example we give an implementation of the Halton low discrepancy sequence, which is in fact totally deterministic, so the popular term quasi-random can be misleading.

```
def halton_1d(idx, base):
    """Generate the idx-th entry in the Halton sequence with given (prime) base."""
    out, f = 0., 1./base
    i = idx
    while i > 0:
        out = out + f * (i % base)
        i = np.floor(i / base)
        f = f / base
    return out

def halton(N, d):
    """Return Halton sequence in d-dimensions of length N."""
    primes = np.array([2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67])
    assert d <= len(primes)
    s = np.zeros((N, d))
    for i in range(d):
        p = primes[i]
        for j in range(N):
            s[j,i] = halton_1d(j, p)
    return s
```

4.2 Adaptive sampling

Given that our surrogate comes equipped with a natural error estimate in the form of the posterior covariance, it seems logical to devise adaptive sampling strategies based on this estimate. For example a simple approach might be:

1. Choose an initial set of samples non-adaptively.
2. Build a Kriging surrogate on these samples.
3. Termination criteria: if integral of $\hat{\sigma}(\xi)$ over $\mathcal{H}$ is sufficiently small, terminate. Here $\hat{\sigma}(\xi)$ is the standard-deviation of the Kriging posterior evaluated at ξ.
4. Solve:
$$\boldsymbol{\xi}_{\text{new}} := \arg\max_{\boldsymbol{\xi}\in\mathcal{H}} \hat{\sigma}(\xi),$$
for the location with the largest estimated error, and add $\boldsymbol{\xi}_{\text{new}}$ to the sample set.
5. Goto 2.

There are a couple of features of the posterior covariance that make this scheme difficult to implement. As we have already seen it is highly oscillatory, multi-modal and non-smooth (for zero value error $\hat{\sigma} = 0$ at the sample points). Therefore not only the integral in Step 3, but also particularly the optimization in Step 4 are difficult. One might almost say that in order to implement this as part pf a global optimization method we need to be able to perform global optimization efficiently! Furthermore the approach is inherently sequential.

However there is a more important point: the posterior covariance depends only on he sample locations - not the function values. This a a feature of the Gaussian, stationarity and linearity assumptions made in Section 3. By assuming a uniform smoothness (via variance and correlation length) of our function f in the prior, we discarded the possibility that there may be regions of f requiring more samples than others. The adaptation procedure above, and any other based on the unweighted posterior covariance, will tend to result in coordinate sequences comparable to the low discrepancy sequences of the previous section, as the adaptation routine simply attempts to "fill in the gaps" regardless of function values. In fact for a given initial sample distribution, correlation length, and design space, these sequences can be pre-calculated — but this is adaptivity not worthy of the name. Therefore another approach is necessary. Nevertheless in Figure 12 we show the progress of such an adaptation strategy.

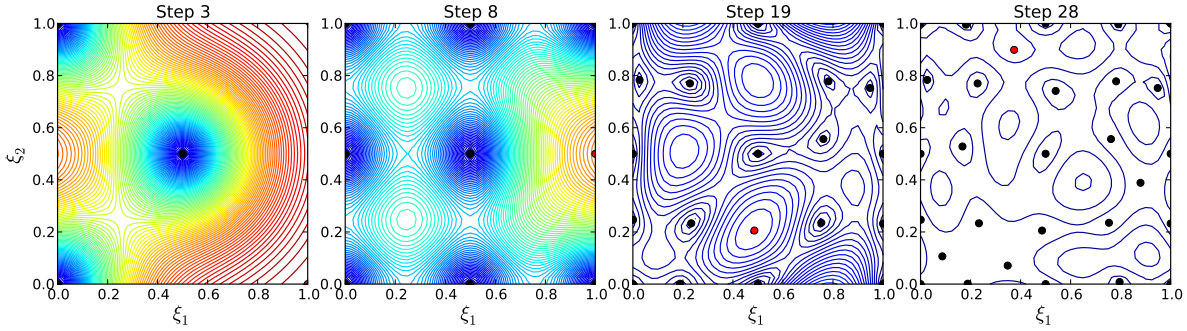


Figure 12: Adaptive sampling on $\mathcal{H}_2$ based on the Kriging posterior covariance. Initial sample: $(\frac{1}{2}, \frac{1}{2})$. Black dots: samples; red dots: next adaptively selected sample; contours: covariance (blue small, red large). Contours have same scale in all plots.

Leave-one-out cross-validation As an alternative we can examine the sensitivity of the surrogate to input conditions. For example we can leave each data point out of the analysis in turn, building m surrogates each based on $m-1$ observations from $\mathbf{y}$. If these surrogates disagree radically in a particular area, then we can presume that area needs more samples. Various error estimates can be designed around this leave-one-out principle. A variant of this approach is the *parameter sensitivity* error estimate. The reasoning goes that if the surrogate is particularly sensitive to choice of μ_x, σ_x, θ or $r(\cdot)$ at a given location, then we should not have much confidence in the surrogate there. Given an error estimate, the corresponding adaptive scheme just adds samples where the estimate is greatest.

Both the leave-one-out and parameter sensitivity adaptive methods could be equally applied to any surrogate model — so it would seem that the posterior covariance is not particularly valuable for adaptation. However this is where Efficient Global Optimization (EGO) comes in, which uses the probabilistic nature of the error estimate to adapt in regions in which the *expected* improvement of a minimum is maximised.

5 Efficient Global Optimization (EGO)

Maximising the expected improvement of the minimum as a criteria for adaptation was popularised by Jones *et al.* Jones et al. (1998). They called the algorithm *Efficient Global Optimization* (EGO). Locatelli Locatelli (1997) proved that under mild assumptions the iterates generated by the method are dense, meaning that eventually all points in the design space will be sampled, and the method is guaranteed converge to the global optimum (if necessary by sampling everywhere).

To calculate the expected improvement, we proceed as follows (referring to Figure 13): given p samples points, the minimum of f the sample set is identified

$$f_{\min} = \min(x_1, ..., x_p).$$

The Kriging surface using these samples is interpreted as a normally distributed random variable with mean and variance given by (6) and (7) respectively. For example, at the point $\xi = 8$ in the figure, a normal density function is plotted. Clearly there is a non-zero probability that the true value of f at $\xi = 8$ is an improvement over the current best value $f_{\min}$. This is true for any ξ with non-zero variance. If the improvement on $f_{\min}$ is defined as

$$I = \max(f_{\min} - x(\xi), 0) \tag{27}$$

then the expected improvement is obtained by taking the expectation of (27):

$$\mathbb{E}[I(x)] = \mathbb{E}[\max(f_{\min} - x(\xi), 0)] \tag{28}$$

It turns out that (28) can be expressed in closed form:

$$\mathbb{E}[I(x)] = (f_{\min} - \hat{\mu}(\xi)) \cdot \Phi\left(\frac{f_{\min} - \hat{\mu}(\xi)}{\hat{\sigma}(\xi)}\right) + \hat{\sigma}(\xi) \cdot \phi\left(\frac{f_{\min} - \hat{\mu}(\xi)}{\hat{\sigma}(\xi)}\right) \tag{29}$$

where $\hat{\mu}(\xi)$ and $\hat{\sigma}(\xi)$ are the mean and standard-deviation of the Kriging posterior evaluated at ξ, and where $\Phi(\cdot)$ and $\phi(\cdot)$ are the standard normal *cumulative* and *probability* distribution functions, respectively. Looking at (29) we note some reasonable properties:

1. The expected improvement will be zero at a sample point assuming zero observation noise $\sigma_y = 0$. Thus the second term in (29) becomes zero. The associated surrogate value at ξ_i is x_i, with $f_{\min} \leq x_i$. The argument in $\Phi(\cdot)$ is therefore effectively $-\infty$, which renders it equal to 0. The fact that the expected improvement will be 0 at a sample point guarantees that no re-sampling will occur. As already noted the iterates are dense in $\mathcal{H}_d$. If $\sigma_y > 0$ it is actually reasonable to resample, as we might obtain a better value on a second attempt!;

2. The first term in (29) is the difference between the current minimum and the predicted value multiplied by the probability that the function value will be smaller than $f_{\min}$. It is therefore large where the prediction is likely to be smaller than $f_{\min}$. The second term tends to be large where there is high uncertainty about whether or not the prediction will be better than $f_{\min}$, i.e. unexplored regions. The utility function can therefore be thought of as providing a trade-off between local and global search;

Figure 15 shows the method implemented on the same 1D function as was given in Figure 14. In addition to the true function and the Kriging surface, the plots also show the expected improvement. This is the objective function maximised in the background. At the first iteration, the infill point is the same as for method 2, i.e. the minimum on the response surface. This is because the expected improvement there is at its maximum. Note that this is the only maximum at the considered interval. At the second iteration, the expected improvement has three maxima. Two of these are in the vicinity of the global minimum of the response surface. However, the global maximum of the expected improvement occurs at a different point, where the design space has not been explored extensively yet. For this reason, the infill point is at that position. When the point is evaluated on the real function, it is found that the function value found is considerably better than the already sampled points. The new function value found becomes the new $f_{\min}$. In the next three iterations, the infill points are all in the vicinity of the global minimum of the response surface (around $x = 0.76$). In the sixth iteration however, the method samples at a different point as the response surface has been sampled extensively around $x = 0.76$ and the expected improvement there has therefore diminished. So, when examining the infill points chosen it can clearly be seen that the method tries to strike a balance between local and global search, which confirms point 2 stated above. Point 1 can be confirmed by examining the 'double bumps' at Figures 15.2 and 15.3. They can be thought of as being one bump 'pinched' to zero in the middle by the presence of the sample point there. Also, note how the maximum expected improvement starts out around 6E-1 and in the sixth iteration has dropped to 1E-11.

Since EGO tries to balance local and global search, the sampling of the space can be quite erratic. To define a stopping criterion for the method it is therefore undesirable to monitor the convergence of the objective function value or the parameter values. The most intuitive stopping criterion in that case would be the expected improvement value. If it drops below a certain threshold, the method can be stopped. To illustrate this, a two-dimensional test function is considered, as given in Figure 16.1. The test function has four minima. The global minimum is denoted by the cross. Figure 16.2 gives the initial response surface built using 10 sample points distributed by an Optimal Latin Hypercube algorithm. 20 iterations are carried out using Method 2 and EGO. Figures 16.3 and 16.5 give the iterates at iteration 10 and 20 for Method 2, respectively. Note that the figures are identical. This is because the method converged after 9 iterations. Subsequent iterates were duplicate points so the method stalled; it converged to one of the *local* minima. Figures 16.4 and 16.6 give the iterates at iterations 10 and 20 for EGO, respectively. After 10 iterations it seems to be converging to the same local minimum as Method 2 but it clearly escapes getting stuck as at iteration 20 it is seen to have located the global minimum.

References

Bayarri, M., Berger, J., Paulo, R., Sacks, J., Cafeo, J., Cavendish, J., Lin, C.-H., and Tu, J. (2007). A framework for validation of computer models. *Technometrics*, 49(2):138–154.

Chiles, J.-P. and Delfiner, P. (1999). *Geostatistics, Modeling Spatial uncertainty*. Wiley.

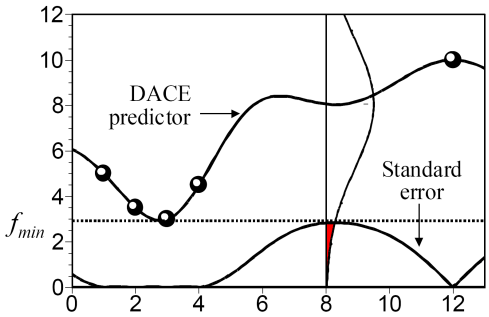


Figure 13: The uncertainty about the function's value at a point (such as $x = 8$ above) can be treated as if the value there was a realisation of a normal random variable with mean and standard deviation given by the Kriging predictor and its standard error. (From Jones *et al.* Jones et al. (1998)).

Chung, H.-S. and Alonso, J. J. (2002). Using gradients to construct cokriging approximation models for high-dimensional design optimization problems. *AIAA 40th Aerospace Sciences Meeting and Exhibit*.

Cressie, N. (1990). The origins of kriging. *Mathematical Geology*, 22(3):239–252.

Cressie, N. (1993). *Statistics for spatial data*. Wiley.

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

Dwight, R. P. and Han, Z.-H. (2009). Efficient uncertainty quantification using gradient-enhanced kriging. *11th AIAA Non-Deterministic Approaches Conference*.

Fukumizu, K., Bach, F. R., and Jordan, M. I. (2009). Kernel dimension reduction in regression. *The Annals of Statistics*, 37(4):1871–9105.

Gandin, L. (1965). *Objective analysis of meteorological fields: Gidrometeorologicheskoe Izdatel'stvo (GIMIZ), Leningrad*. Translated by Israel Program for Scientific Translations, Jerusalem.

Handcock, M. S. and Stein, M. L. (1993). A Bayesian analysis of Kriging. *Technometrics*, 35(4):pp. 403–410.

Hinkelmann, K. and Kempthorne, O. (2008). *Design and Analysis of Experiments. I and II*. Wiley.

Jones, D., Schonlau, M., and Welch, W. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45.

Kennedy, M. C. and O'Hagan, A. (2000). Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B*, 63:425–464.

Kitanidis, P. K. (1986). Parameter uncertainty in estimation of spatial functions: Bayesian analysis. *Water Resources Research*, 22(4):499–507.

Kitanidis, P. K. and Lane, R. W. (1985). Maximum likelihood parameter estimation of hydrologic spatial processes by the Gauss-Newton method. *Journal of Hydrology*, 79(1-2):53 – 71.

Laurenceau, J. and Sagaut, P. (2008). Building efficient response surfaces of aerodynamic functions with kriging and cokriging. *AIAA Journal*, 46(2):498–507.

Locatelli, M. (1997). Bayesian algorithms for one-dimensional global optimization. *Journal of Global Optimization*, 10:57–76.

Lophaven, S., Nielsen, H., and Søndergaard, J. (2002). DACE − A Matlab Kriging Toolbox. Report IMM-REP-2002-12, Informatics and Mathematical Modelling, Technical University of Denmark, 34 pages.

Mardia, K. V. (1989). Maximum likelihood estimation for spatial models. *Proceedings from the symposium on Spatial Statistics: Past, Present, and Future*, pages 203–253.

Matheron, G. (1963). Principles of Geostatistics. *Economic Geology*, 58:1246–1266.

Mira, J. and Sánchez, M. J. (2002). Analytical results for a Bayesian bivariate Cokriging model. *Statistics Probability Letters*, 58(1):97 – 109.

Morris, M. D., Mitchell, T. J., and Ylvisaker, D. (1993). Bayesian design and analysis of computer experiments: Use of derivatives in surface prediction. *Technometrics*, 35(3):pp. 243–255.

Omre, H. and Halvorsen, K. (1989). The Bayesian bridge between Simple and Universal Kriging. *Mathematical Geology*, 21(7):767–786.

Philip, G. and Watson, D. (1986). Matheronian Geostatistics – Quo vadis? *Mathematical Geology*, 18(1):93–117.

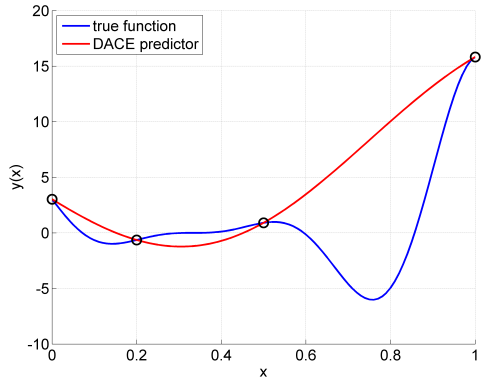
Srivastava, R. (1986). Philip and Watson – Quo vadunt? *Mathematical Geology*, 18(1):141–146.

Stein, M. L. (1999). *Interpolation of spatial data, some theory for Kriging*. Springer.

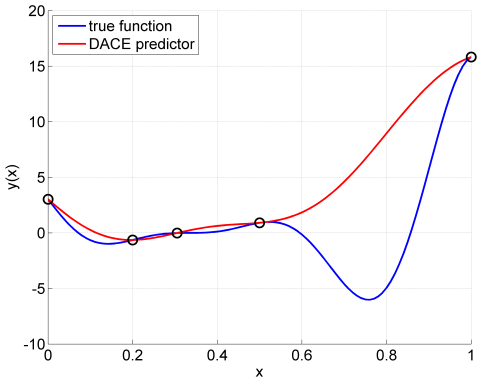
Webster, R. and Oliver, M. A. (2007). *Geostatistics for environmental scientists 2nd edition*. Wiley.

Wikle, C. K. and Berliner, L. M. (2007). A Bayesian tutorial for data assimilation. *Physica D: Nonlinear Phenomena*, 230(1-2):1 – 16.

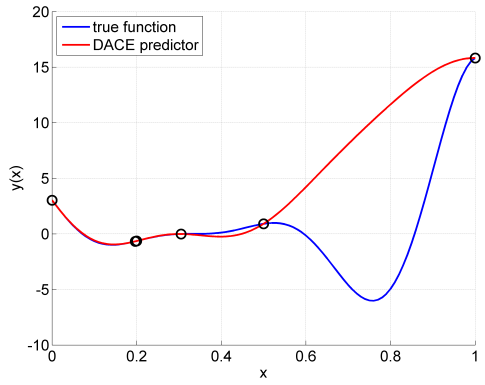
Zhang, Y. and Oliver, D. (2011). Evaluation and error analysis: Kalman gain regularization versus covariance regularization. *Computational Geosciences*, 15(3):1–20.



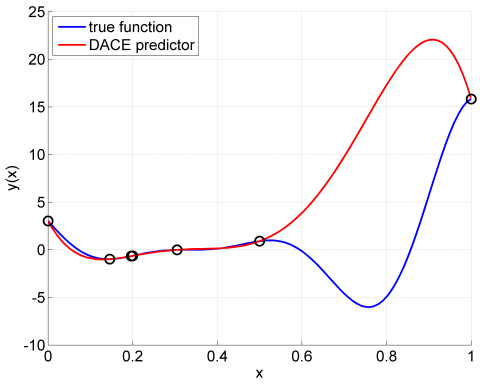
14.1:



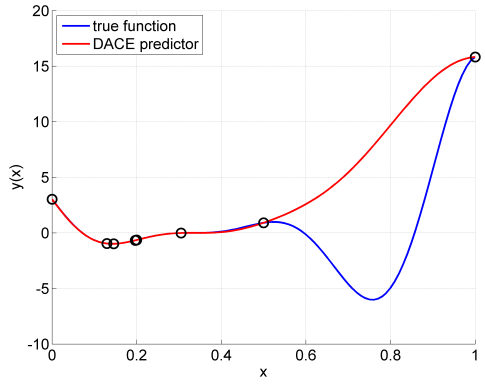
14.2:



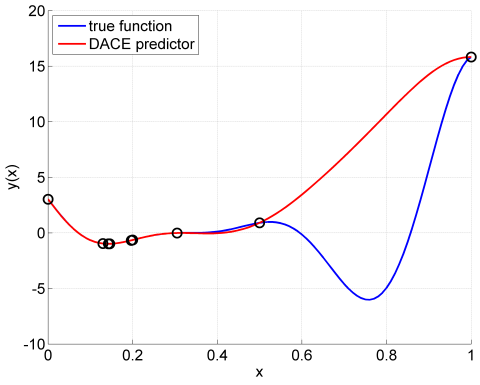
14.3:



14.4:

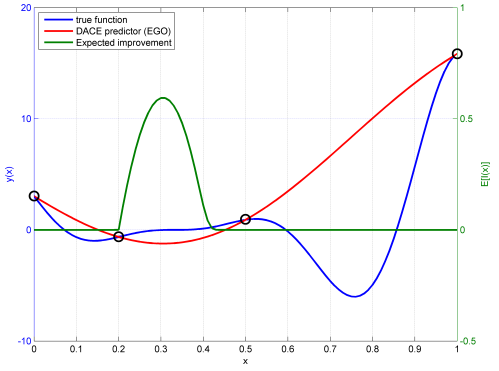


14.5:

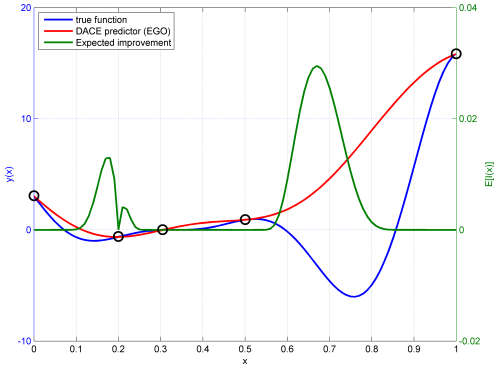


14.6:

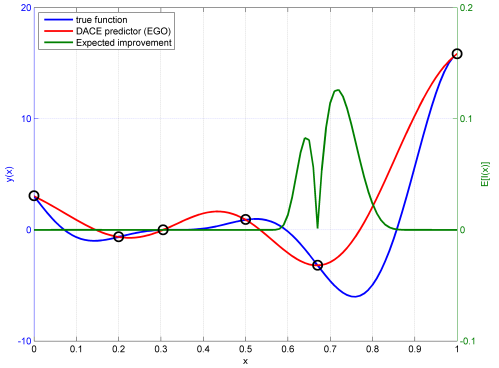
Figure 14: Optimization of the function $y(x) = (6x-2)^2 \sin(12x-4)$ using the global minimum on the response surface as the infill point.



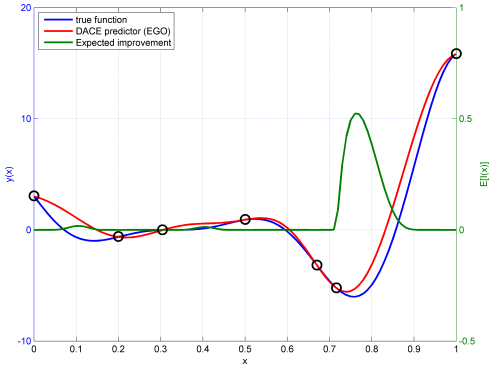
15.1:



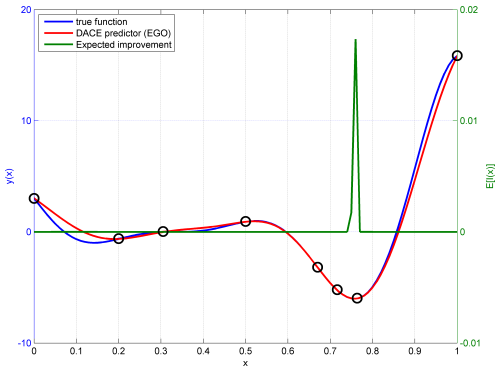
15.2:



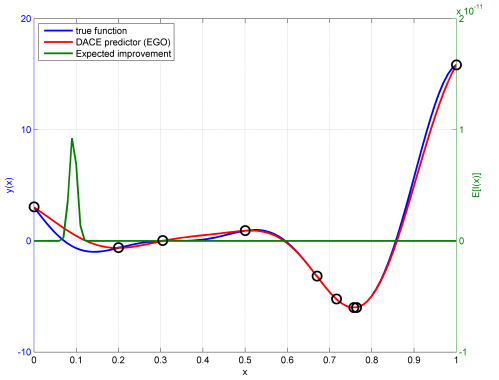
15.3:



15.4:

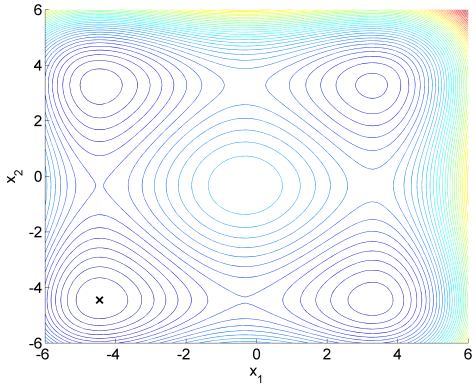


15.5:

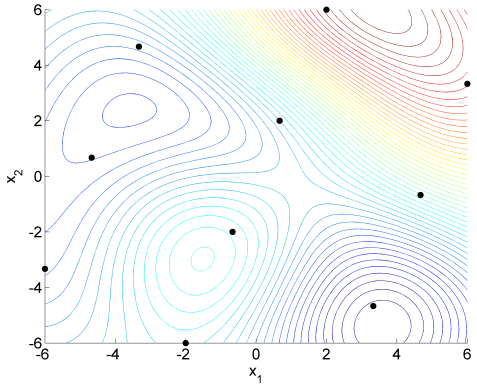


15.6:

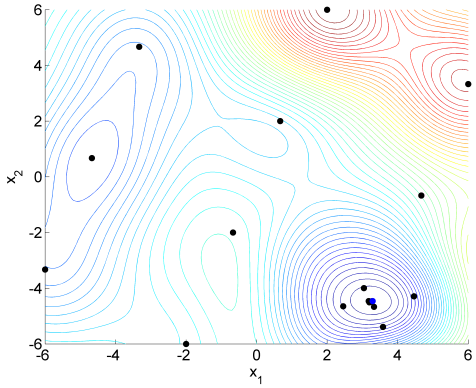
Figure 15: Optimization of the function $y(x) = (6x-2)^2 \sin(12x-4)$ using the maximum of the expected improvement as the infill point (EGO).



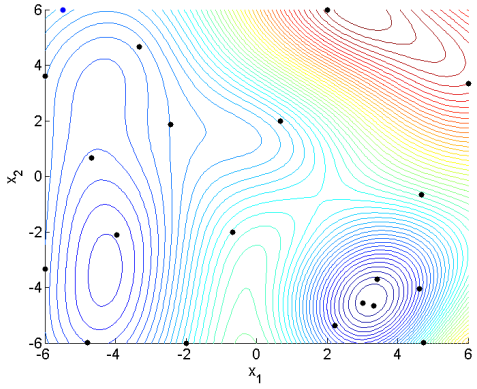
16.1:



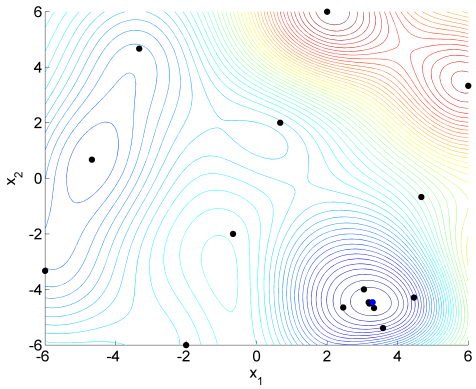
16.2:



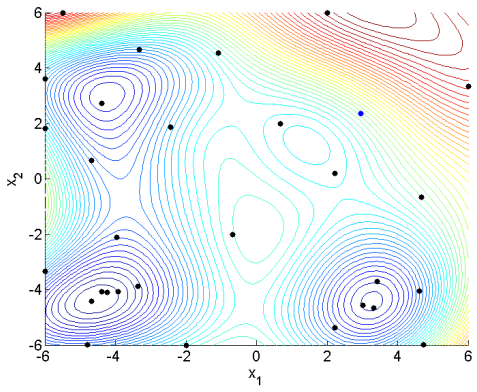
16.3:



16.4:



16.5:



16.6:

Figure 16: Multimodal two dimensional test function $y(\mathbf{x}) = \sum_{j=1}^{2} 0.01\left[(x_j+0.5)^4 - 30x_j^2 - 20x_j\right]$ with the global minimum at the black cross (a); the initial response surface using 10 sample points (b); response surfaces after 10 (c) and 20 (e) iterations of adding a sample at the surrogate minimum; response surfaces after 10 (d)) and 20 (f)) iterations of EGO.

A Background of Kriging

Kriging was proposed independently in the field of geology by Matheron (1963) and in the field of meteorology by Gandin (1965), as a means of spatially interpolating sparsely sampled observations. It is equally well-suited to interpolation in general parameter spaces, in which context it is useful for optimization, and as a general-purpose surrogate model. A detailed account on the origins of Kriging is given by Cressie (1990), while a complete overview is provided by Cressie (1993) and Stein (1999). A popular implementation of Kriging is provided by the Matlab Toolbox `DACE` Lophaven et al. (2002). A recent development for extremely large data sets is Fixed Rank Kriging Cressie and Johannesson (2008).

In Kriging's original application as an interpolation tool for potentially noisy experimental data or data with discontinuous spatial correlation, it was undesirable to exactly fit observed values. Two techniques were used to regress the data according to the level of observational noise or discontinuous correlation: (a) altering the covariance matrix with a correlation function $r(\cdot)$ with $r(0) \neq 1$ Gandin (1965), or (b) introducing a "nugget-effect" in the semi-variogram Matheron (1963); Cressie (1993), which is often considered to be equivalent to adding a "nugget" to the diagonal of the covariance matrix – although we note that this equivalence is generally limited to the assumption that a small amount of observational noise is equivalent to discontinuous spatial correlation. With the application of Kriging to the output of deterministic computer models, regression is generally considered to be undesirable, and the method is constructed as if the observational noise were zero Dwight and Han (2009). Nevertheless a small nugget must always be added for numerical reasons Lophaven et al. (2002); Dwight and Han (2009). In this work we advocate regressing also the output of computer models, which are often also noisy. In general we claim that primary interest typically lies, not in a perfect surrogate of an imperfect computer model, but in prediction (with error estimates) of a real process. The modifications of Kriging suitable for modelling experimental noise, are also suitable for modelling discretization error and modelling discrepancy. Although these errors are not random like experimental noise (an example of aleatory uncertainty), they are unknown (epistemic uncertainty), and our lack-of-knowledge can be represented as a probability distribution.

Gradient information can be incorporated into the method in two ways: 1) assuming gradient observations and value observations are co-located, one can construct new artificial value observations by linear extrapolation over a distance δ_ξ. This is usually done in each coordinate direction at each observation. Standard Kriging is then applied to this extended data-set. This approach is simple, but is sensitive to the choice of δ_ξ, results in very ill-conditioned gain matrices for small δ_ξ, and does not readily include observation errors Chung and Alonso (2002); Laurenceau and Sagaut (2008); Dwight and Han (2009). Therefore we consider the approach examined in Section 3.2 of this work: 2) including the gradient as a co-variable in cokriging.

Combining error estimates and gradient information in cokriging is challenging, as the interpretation of several elements of cokriging is unclear, see the discussion in Philip and Watson (1986) and Srivastava (1986). A critical question is: what is the most appropriate way to represent observation errors? This question can be clarified by formulating the problem in a Bayesian framework. Several Bayesian derivations of Kriging have been

made Kitanidis (1986); Omre and Halvorsen (1989); Handcock and Stein (1993); Mira and Sánchez (2002); Kennedy and O'Hagan (2000); Bayarri et al. (2007), and a particularly clear presentation is that of Wikle and Berliner (2007). In this last reference we find a clear distinction between the semi-variogram and the observation error: the semi variogram is the covariance of the Bayesian prior, while the observation errors are contained in the likelihood. As such, the observation errors are clearly not part of the semi-variogram. With this simple distinction between prior and likelihood the Bayesian framework enables us to treat observation errors in Gradient-Enhanced Kriging (Section 3.2).

In Figure 17 we have classified works on Kriging according to three criteria: the conventional statistical or Bayesian framework, the treatment of gradients, and the treatment of observation errors. To our knowledge there is no existing work that combines a treatment of errors and gradients. Although we presently consider the robustness of Cokriging, we note that the theoretical results developed for the strict positive-definiteness and condition number of A_c also apply to standard Kriging. Furthermore an operation closely related to simple Kriging is the analysis step in a Kalman filter Kalman (1960); Wikle and Berliner (2007), where "Kalman gain regularisation" or "covariance regularisation" can be necessary for matrix inversion Zhang and Oliver (2011). Another related technique is Kernel Dimension Reduction, where a "regularisation term" is required to enable matrix inversion. Importantly Fukumizu et al. (2009) "are not aware of theoretically justified methods of choosing these parameters; this is an important open problem". We observe that in both cases the present approach of estimating the robustness analytically, could readily be applied in order to improve understanding of these regularisation methods.

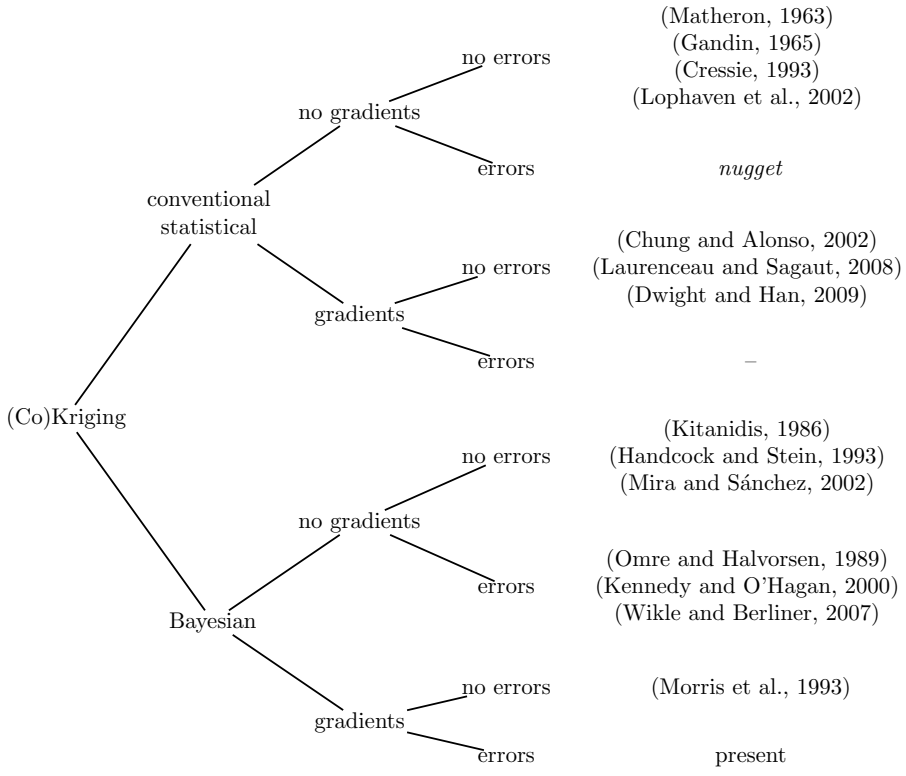


Figure 17: References to literature on different Kriging and Cokriging techniques.

B Vectorized construction of P and P_c in Python

As mentioned in Section 3 the bottleneck in the given implementation of Kriging is in the construction of the prior covariance matrix P due to explicit loops. These loops can be replaced in `kriging()` and `gek()` with single calls to the following `vec_r()`, `vec_drdxi()`, and `vec_d2rdxi2()` which return full covariance (sub-)matrices.

```
def d_diff(xi):
    """
    Return tensor of distances between all pairs of points in d-dimensions.
    I.e. (x_i - x_j)_k for all i,j in {0,...,n-1}, k in {0,...,d-1}.
    """
    n, d = xi.shape
    return np.einsum('ik,j', xi, np.ones(n)) - np.einsum('i,jk', np.ones(n), xi)

def d_dist2(xi):
    """
    Return matrix of squared distances between all pairs of points in d-dimensions.
    I.e. h^2 = |x_i - x_j|^2 for all i,j in {0,...,n-1}.
    """
    return np.sum(d_diff(xi)**2, axis=2)

def vec_r(xi, theta):
    """
    Vectorized version of r() - return full prior covariance matrix P.
      xi    - full coordinate vector, array n x d
      theta - correlation function scale parameter
    """
    return np.exp(-theta * d_dist2(xi))

def vec_drdxi(xi, theta, P00):
    """
    Vectorized version of drdxi_i and drdxi_j - return P_{01} and P_{10}.
      xi    - full coordinate vector, array n x d
      theta - correlation function scale parameter
      P00   - output of vec_r(), save recalculation
    """
    n, d = xi.shape
    tmp0 = 2.*theta* d_diff(xi) * np.einsum('ij,k', P00, np.ones(d))

    out = np.zeros((n, d*n))    ### Flatten the 3-tensor to an (n x d*n) matrix
    for i in range(d): out[:,i::d] = tmp0[:,:,i]
    return out, out.T

def vec_d2rdxi2(xi, theta, P00):
    """
    Vectorized version of d2rdxi_idxi_j - return P_{11}.
      xi    - full coordinate vector, array n x d
      theta - correlation function scale parameter
      P00   - output of vec_r(), save recalculation
    """
    n, d = xi.shape
    tmp0 = d_diff(xi)
    tmp1 = np.einsum('i,j,kl', np.ones(n), np.ones(n), np.identity(d)) \
        - 2.*theta*np.einsum('ijk,ijl->ijkl', tmp0, tmp0)
    tmp2 = 2.*theta * tmp1 * np.einsum('ij,k,l', P00, np.ones(d), np.ones(d))

    P11 = np.zeros((d*n, d*n))   ### Flatten the 4-tensor to an (d*n x d*n) matrix
    for i in range(d):
        for j in range(d):
            P11[i::d,j::d] = tmp2[:,:,i,j]
    return P11
```