How to rewrite multivariate random functions as univariate to do cokriging. A flexible algorithm

Adrián Martínez Vargas

Abstract

Most of geostatistical software have a limited number of kriging models and are not flexible enough to allow to define new or complicated models. With the aim to obtain a flexible algorithm for kriging was used a notation of multivariate regionalized random function (RF) as a univariate RF $Z(x, i)$, with a drift $m(x, i, \mathrm{j})$, were i and j indicate variable identifiers. A geostatistical library named OpenKriging was programed in C++. This library has a function callable from scripts written in Python, able to construct the kriging system of equations of the above mentioned RF, with no restriction in the number of spatial dimensions, variables and monomials in the drift. Two types of drift monomials where implemented, geographical and external drift. The drift monomials can be independent or dependent. The function responsible to build the kriging system of equations was not programed for a predefined model, the model is defined by the user, passing from Python the appropriate data and parameters and modifying from Python the system prior to solve it. The resulting library can handle the most commons models, and also new or uncommon models. The library is easy to modify to increase the functionality. The functionality was shown trough two examples of kriging models: one with a complicated definition, the other can be considered as uncommon.

Key Words

C++, geostatistical software, GPL, kriging, multivariate random functions, n-dimensional, OpenKriging, Python.

Received: January 2009 / Accepted: May 2010

Cómo reescribir funciones aleatorias multivariadas como univariadas para hacer cokrigeage. Un algoritmo flexible

Resumen

La mayoría de los software para geoestadísticas poseen un limitado número de modelos de krigeage y no son lo suficientemente flexibles para definir nuevos o complicados modelos. Con el objetivo de obtener un algoritmo flexible de krigeage se empleó la notación de funciones aleatorias regionalizadas (RF) multivariadas escritas como una RF univariada $Z(x, i)$, con un *drift* $m(x, i, j)$, donde i y j son identificadores de variables. Se programó en C++ una librería geoestadística llamada OpenKriging, con una función que se puede llamar desde código escrito en Python, la cual es capaz de construir el sistema de krigeage de la RF mencionada, sin restricciones en el número de dimensiones espaciales, variables y monomios en el *drift*. Se implementaron dos tipos de *drift*: los geográficos y los *drift* externos, estos pueden ser independientes o no. La función responsable de construir el sistema de ecuaciones krigeage no se programó para un modelo predefinido, el modelo es definido por el usuario pasando desde el código en Python los datos y argumentos apropiados y modificando el sistema antes de resolverlo. La librería resultante puede asimilar muchos de los modelos conocidos de krigeage y también modelos nuevos o poco comunes; esta es fácil de modificar para incrementar su funcionalidad, la cual se mostró a través de dos ejemplos de modelos de krigeage: uno con definición complicada y el otro poco común.

Palabras clave

C++, funciones aleatorias multivariadas, GPL, krigeage, n-dimensional, OpenKriging, Python, software para geoestadística.

Recibido: enero 2009 / Aceptado: mayo 2010

INTRODUCTION

Earliest implementations of kriging in geostatistical libraries were limited to a set of bi-variate regionalized random functions (RF) $Z_i(x)$, up to three spatial dimensions in coordinates vector x and a reduced set of kriging models. In the legendary Geostatistical Library (GSLIB) by Deutsch, Clayton and Journel (1998) kriging options were implemented in the packages: 2D Kriging (kb2d), 3-D Kriging Program (kt3d), 3-D Cokriging Program (cokb3d) and the Indicator Kriging Program (ik3d). The prefix 3d and 2d indicate that those packages are designed for two and three dimensions only. In GSLIB cokriging was implemented for only two variables.

This tendency remains in some modern geostatistical packages, for example, in the popular GsTL: the Geostatistics Template Library, presented and explained by Remy (2001, p. 26) the cokriging method is implemented for only two variables, and only up to three dimensions are allowed. In the software SGeMS, the graphical front end of GsTL (Remy *et al. 2006*) the set of kriging models implemented is limited to: the univariate versions of simple kriging, ordinary kriging and kriging with trend, this last one limiting the drift definition to a total of nine possible monomials (X, X^2, Y, Y^2, Z, Z^2, XY, XZ, YZ). In the bi-variate kriging models the options are: simple cokriging, ordinary cokriging and two variants of Markov model. Remy *et. al.* (2006) also gave an implementation of indicator kriging.

The commercial software ISATIS has the largest number of geostatistical techniques implemented. This software includes a wide set of external drifts, also covers from the classical implementation of simple kriging to disjunctive kriging, some special models, for example: kriging several variables linked through partial derivatives and kriging with measurement error (Bleines *et al. 2007*). Despite the potential of this software the number of dimensions of the coordinate space is limited up to three, and the implementation of new models is limited or impossible because is not open source.

A different approach was introduced by a new generation of libraries that run over scripting programming languages as Matlab, Octave, or R. Gebbers R. (Trauth 2006, p. 177) introduced the use of geostatistics in Matlab throw a pure Matlab code, with a deep vectorized but intuitive approach. The code presented by Gebbers R. only covers the ordinary kriging, but can be modified to be extended to any model. Another example was shown by Diggle & Ribeiro-Jr (2007), they presented two packages for R: geoR and geoRglm (Ribeiro & Diggle 2001, Christensen & Ribeiro 2002), both for model based geostatistics, including likelihood-based and Bayesian methods, but

they are only implemented for problems defined in two dimensions, and up to two variables. Another R package for geostatistics is gstat, by Pebesma (2004). Multivariate geostatistics in this package can handle more than two variables distributed in a three-dimensional space.

Script languages are slow. To increase the speed in computations the packages running over scripting languages are often programmed as shared libraries written in C, C++ or Fortran. The script works as an interface from where data is passed to functions inside a compiled library, as reference in memory or as pointers. Compiled libraries are in charge to do the heavy work: looping over large data, obtain numerical solutions, etc.

Most of kriging algorithms implemented in open source geostatistical libraries are designed for a fixed number of variables, dimensions and kriging models. To modify those algorithms, with the aim to increase the number of dimensions, variables and kriging models, is not an efficient idea because usually large transformations of the original code are required.

A list of some kriging models (Bleines *et al. 2007*): can be ordinary kriging, simple kriging, kriging in the IRF-k case, kriging drift estimator, kriging with external drift, kriging for filtering model components, factorial kriging, block kriging, polygon kriging, kriging gradient estimator, kriging for several variables linked through partial derivatives, kriging with inequalities, kriging with measurement error, lognormal kriging, cokriging, extended collocated cokriging, indicator kriging, disjunctive kriging, uniform conditioning and service variables, and other models not mentioned here. Notice that most of those kriging models can be combined to obtain hybrid kriging models. To programming a suitable algorithm for all base models and hybrids can be difficult.

Trough comparing kriging systems of equations for different models it can be notice that they do not exist too many differences between many of them. For example the kriging system of equations for drift estimation is the same to the universal kriging one but with all covariances in the right hand side of the equations equal to zero; to perform IRF-k kriging instead to use covariances or variograms we use generalized covariances; to perform disjunctive kriging we have to estimate by simple kriging the Hermite coefficients. The differences between the kriging systems can be summarized in two main types: 1) variation relate with the definition of the kriging system in term of: spatial variability functions, number of variables and drifts, 2) variation in the nature of the data we use as input.

In the first part of this paper the idea to rewrite a multivariate set of random functions (RF) $Z_i(x)$ into a single univariate RF $Z(x,i)$ with a drift $m(x,i,\mathrm{j})$ was discussed (Martínez-Vargas 2006). The resulting system of equations is equivalent to the system obtained by the classical multivariate approach; in addition the method can handle most of the kriging models mentioned above and simplify the construction of the system of cokriging equations.

The aim of this paper is to present flexible algorithms to kriging RFs $Z(x,i)$, without limitations in the number of dimension, variables, drift size and types, able to handle the most commons models, and also easy to modify to implement new or uncommon models.

METHODOLOGY

A geostatistical library name OpenKriging was created. It is a shared library that has a function named *k_system*. This function is the responsible to construct the kriging system of equations for a single target v, given a RF $Z(x,i)$, with a drift $m(x,i,\mathrm{j})$ and a specific model of spatial variability (Stage A). This function can be called from a scripting code written in Python. The flexibility of this kriging algorithm lays in the freedom that has the user to define the kriging model to be used, passing to the function *k_system* the appropriate data and options. The main data that the user has to pass to the function are: data locations and variables $z(x,i)$, the drift definition $m(x,i,\mathrm{j})$ and the type of spatial variability. No restrictions were imposed in the number of variables i and dimensions in the location vectors x, neither in the number of monomials and their types. In addition the user can modify the kriging system of equation from the Python script, before to solve it.

The shared library was created to experiment existing and new models in a single testing target (Figure 1), but can be extended to apply kriging in many targets, looping in each target and finding neighborhood data with a neighborhood searching function available in the library. This approach can be implemented directly in the shared library (Figure 3) or from Python scripts (Figure 2), because the neighborhood searching function is also available for external applications. This one is the stage B of development of the library but it is not disused in this paper.

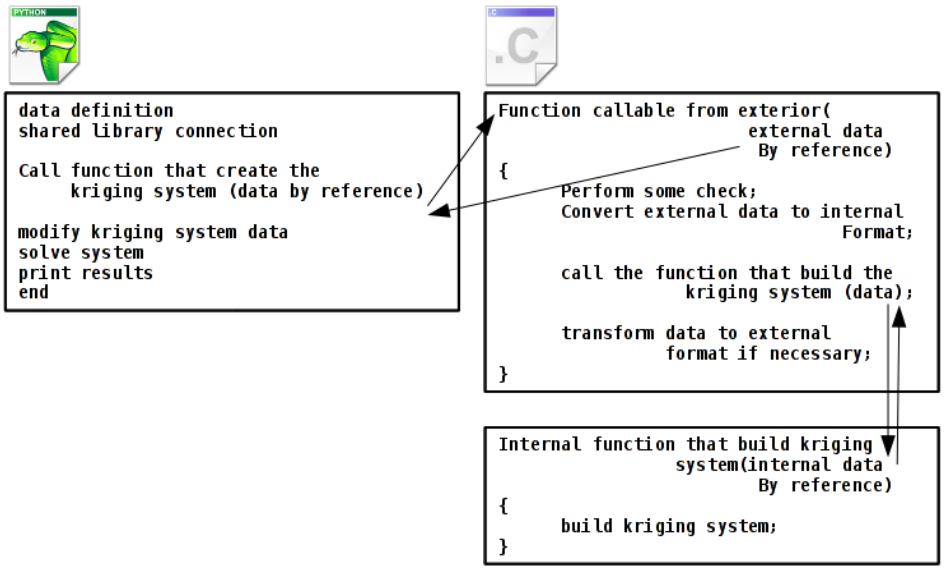

Figure 1. Schematic implementation in pseudo-code of the kriging algorithm for a single target.

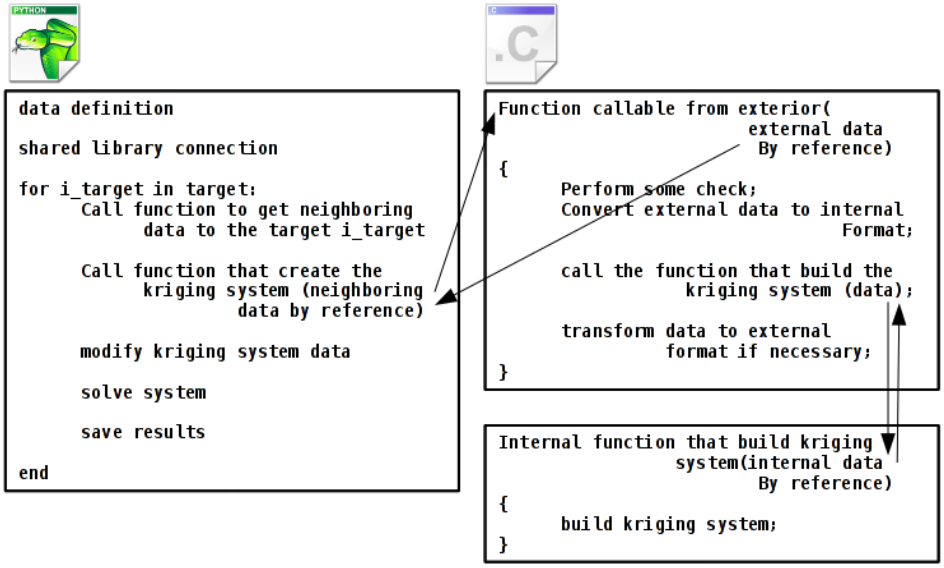

Figure 2. Schematic implementation in pseudo-code of the kriging algorithm for n targets, looping in Python.

Others auxiliary functions in the library are callable from external applications, for example: functions to perform test of communication, transference of the data and to obtain the kriging matrix definition in OpenOffice Math format. It is also available a function to calculate the mean variogram and covariance in a block, which is necessary to calculate the standard deviation of kriging errors from Python.

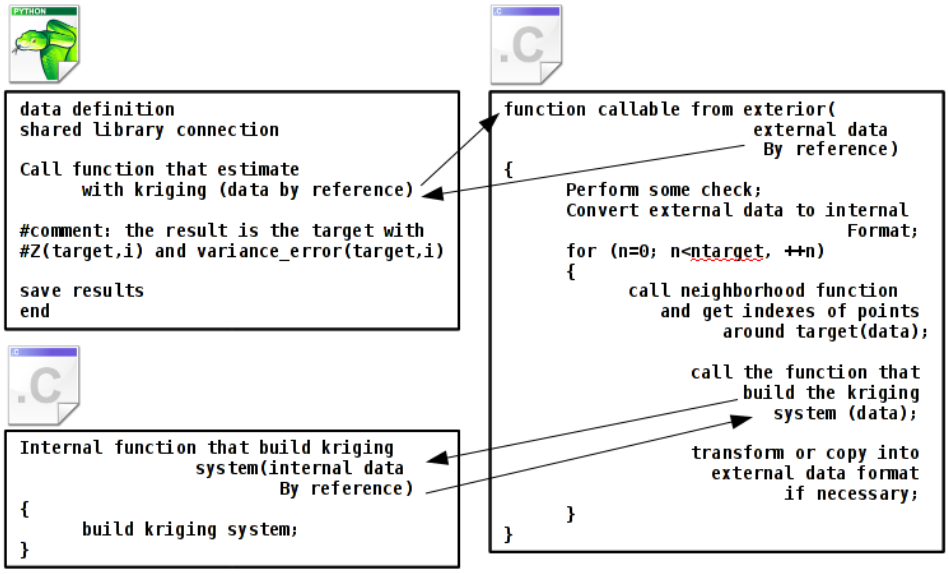

Figure 3. Schematic implementation in pseudo-code of the kriging algorithm for n targets, looping inside the C++ library.

To create the shared library five principles were followed:

1. The source code is open, to allow any modification by the users
2. A procedural programming approach is preferable, to make as clear as possible the code for non experimented programmers
3. The communication with external applications is important, specially with scripting languages
4. The speed is not the main objective, but is important
5. The library is portable to common operating systems, for example Linux and Windows.

To ensure a free access to source code the license GPL from GNU was adopted (Free Software Foundation 2007). In principle the idea of the shared library start by a flexible code that allows to do as many operation possible from the scripting language, but many modifications will be necessary to fix bugs, to add more functionality, or to implement new kriging models in the development Stage B.

The library was programed in C++, but with C procedural philosophy (Brokken 2008). To solve main tasks and sub-tasks a conglomerate of functions was created, communicating each one to the others through arguments and variables. This approach is very intuitive for many non expert programmers. Was avoided the use of classes, instead were used structures. Was also avoided the definition of templates functions. With this approach only a minimum of knowledge about C++ is required to modify the library source code. To know about concepts related to object oriented programing and generic programming is not necessary. Confusing

concepts as pointers were avoided as mush as possible, instead were used reference variables defined in function's arguments as T & variable where T is the variable type (Brokken 2008).

The matrices and vectors manipulations were implemented with TNT version 3.0.12 (Pozo-Roldan 2004). In addition the numerical library JAMA/C++ (Monsalve-Tobon Juan Esteban 2005) was used to solve numerical problems. Some abstract containers from the Standard Template Library (STL) (Brokken 2008) as *vector* and *string* were occasionally used as extra tools. TNT and JAMA/C++ were redistributed with the geostatistical library sources (Appendix A). The use of extra libraries was avoided, to make easy the portability to different operative systems.

The shared library was compiled in with GNU G++ for the operative system Windows and Linux, and distributed as binary files. To call those files from Python 2.x in Windows and Linux they have to be copied inside system directories (Appendix A).

The communication with external applications was performed through the prototype:

```
extern "C"
{
   // C-declarations go in here
}
```

This prototype allows to call the external functions in the library from scripting languages as Python or R, also from compiled softwares, for example in C++, C# or Visual Basic.

To make an universal transference of data a set of external structures was declared in the file “ext_globals.cc”; those structures define the data and models that we transfer:

1. The spatial variability model
2. The search neighbor
3. The data, including the external drift in the data
4. The target, including the external drift in the target

The arrays coming from external applications are passed to the library by reference, as classic C uni-dimensional arrays, defined with the pointer operator as T **variable*. Most of those arrays are reconverted to TNT arrays as Array1D (int *n*, T **a*) or Array2D (int m, int n, T *a), where *m* and *n* are the number of columns and rows in the arrays, T **a* is an uni-dimensional array of type T with C order, often referred

T *a is an uni-dimensional array of type T with C order, often referred to as "row-major ordering" (Pozo-Roldan 2004). This method take the reference in memory of *variable saving running time and memory, also allowing appropriate transference of data between the library and external applications.

The selected scripting language to communicate with the library was Python 2.x, the communication was performed trough the modules *ctypes* version 1.0.3, and *numpy* version 1.3.0. In the case of the module *numpy* a version equal or higher to 1.1.0 is recommended, because allows to reading matrices from files (NumPy community 2009). Python was selected because is easy, intuitive, cross platform and there are a large number modules that make easy the data manipulation, graphical user interface (GUI) implementation and plotting. In some tests were used the Python modules: *numpy* for data manipulation and scientific calculation; *Visual Python* for 3D plotting and *Matplotlib* for 2D plotting.

To create the kriging algorithm three main objective were set:

1. they are not limitations in the number of dimensions,
2. neither in the number of variables and drift monomials,
3. the algorithm must be flexible enough to permit modifications.

Avoiding limitation on the number of dimensions

To avoid limitation on the number of dimensions non special algorithms are necessary to define the kriging matrix, but to calculate the spatial variability and to search neighboring points to a target with an (hyper)ellipsoid searching function. For example, to estimate the variogram value between two points the anisotropy per structure must be considered; if the anisotropy exist it is necessary to translate, rotate and then rescale the anisotropic distance vector to obtain an isotropic scalar distance h.

In systems with one dimension the rotation is not performed. For two or more dimensions the idea of plane rotation (also called Givens rotations) was implemented as $\mathrm{P}\mathrm{x} = \mathrm{x}'$, where P is the rotation matrix, x, the original coordinates vector and x' the rotated vector. The rotation in two and three dimensions are special case of the plane rotation (Meyer 2000). In the library we only rotate in the three first dimensions, that does not affect the others, as shows Figure 4.

Figure 4. Definition of the plane rotation matrix in an n-dimensional space, in the (i, j)-plane. The entries *c* and *s* are cosine and sine, respectively. The rotation of a vector x is performed by matrix multiplication (x'=P•x). Notice that if *i* and *j* are lower than four; the elements of the vector x corresponding to dimensions greater than three are not affected by the rotation (taken from Meyer (2000)).

Actually rotations are performed for a vector *vect* centered at 0,0 a angle *alfa* (in radians) anticlockwise, with the functions:

```
Array1D<double> rota_3D(Array1D<double> &vect, Array1D<double> &alf)
Array1D<double> rota_2D(Array1D<double> &vect, Array1D<double> &alf)
```

The rotation matrix for 2D, along a normal axis to the 2D plane is:

$$\mathrm{P}=\begin{pmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{pmatrix}$$

The rotations in three dimensions are applied first trough Z axis, second trough Y, and finally trough X. The matrices for each rotation axis are defined as shows Figure 5.

Avoiding limitations in the number of variables

To build kriging systems with an arbitrary number of variables the methodology explained in the first part of this paper was used (Martinez-Vargas 2006). We write a multivariate RF as $Z(x,i)$ with drift $m(x,i,\mathrm{j})$. To implement the methodology we pass to the function void k_system(...), which is responsible to build the kriging system, the locations of all data points x as a TNT Array2D with a row per point and a column per dimension. We also pass the values $Z_i(x)$ as a TNT Array2D, with a row per point and a column per variable. The notation $Z(x,i)$ is achieved with a mapping function defined as a TNT Array2D named *index*, with a row per each tuple (x,i), one column for the index of the row k, where is stored the point with vector coordinate x_k, and a column with identifiers of variables i (Figure 6).

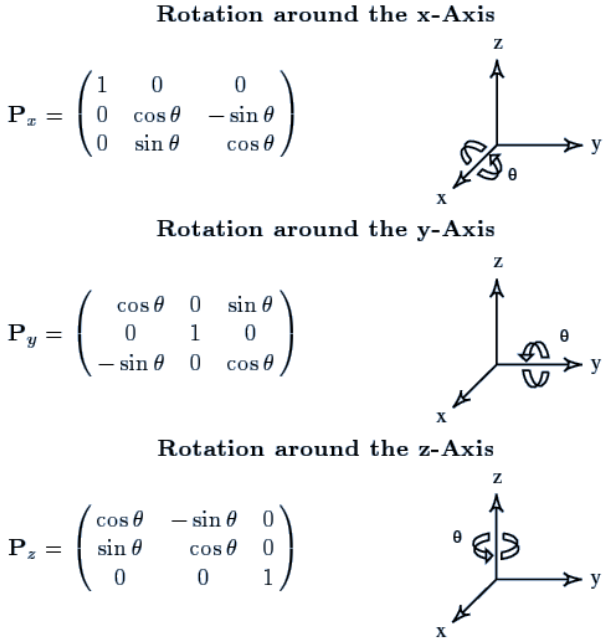

Figure 5. Definition of the rotation matrices trough the axis *x*, *y* and *z* (taken from Meyer (2000))

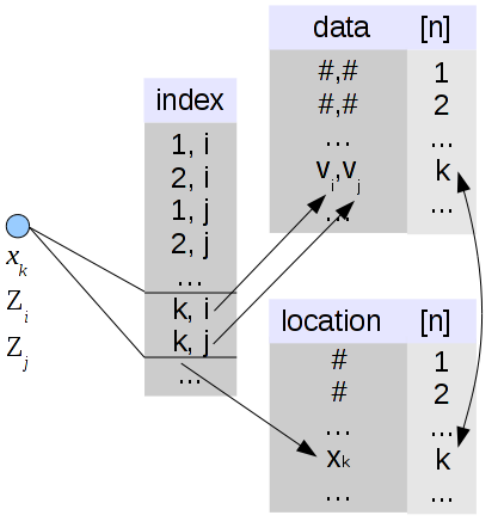

Figure 6. Schematic construction of the mapping function *index* from a multivariate data in a multidimensional space with location vectors x_k .

What about the drift?

The drift $m(x,i,\mathrm{j}) = \sum a_l m_l(x,i,j_l)$, is defined trough a set of n monomials $m_l(x,i,j_l)$ with two possible types: geographic monomials, and external drift monomials, where j is the vector that define the variable j_l assigned to a monomial m_l . The geographic drift monomials are defined in term of coordinates x as:

$$f_l(x,i,j_l) = \mathbf{1}(i,j_l)\prod x_t^{p_t}$$

for each dimension t, and a power p. The external drift is defined as:

$$e_l(x,i,j_l)=\mathbf{1}(i,j_l)e(x,j_l)$$

where the external deterministic function $e(x,j_l)$ is defined in any point x and is stored in a TNT Array2D, with a row per each point x and a column per each external drift variable. We can define without limit an arbitrary number of external and geographical drifts.

A monomial for a variable i can share or not coefficients with monomials of another variables j_l. When monomials share coefficients they are dependent and the indicator function is $\mathbf{1}(i,j_l)=1\,\forall i,j$. If a monomial is independent then $\mathbf{1}(i,j_l)=1\forall i=j_l$, and $\mathbf{1}(i,j_l)=0\forall i\neq j_l$.

Monomials definitions are stored in a C++ structure named *monomial*, with an identifier of the type of drift, an identifier for the variable j_l associated, and an identifier of independence status. For geographical monomials there is an array with dimensions identifiers t and another one with the power p_t. For external drift it is defined an identifier j_l of the variable of the external drift. The drift is stored as a TNT Array1D of monomials.

The definition of the kriging system of equations

The system of kriging equations is composed by the kriging matrix, the *m* solution vectors and the left *m* side vectors. Those vectors are stored in a matrix with *m* columns; each column is associated to a variable, assuming that we want to retrieve the full solution set for each one of the *m* variables in the system.

The kriging system incorporate a set of values of spatial variability, and a set of values of monomials that compose the drift. To define the spatial variability we want to calculate we pass to the function *k_system*(...) an identifier specifying if the system is retrieved in term of variograms, covariances or generalized covariances, but this last one is not implemented yet. Additionally we pass a spatial variability model as an array of structures (named *var_model*) and other necessary variables (Appendix A). To define the drift we pass the drift definition in an array of monomials, and also a TNT arrays with external drift values in the data and target locations.

The kriging matrices and vectors are constructed in two separate steep, first the spatial variability part, second the drift. The results are retrieved by reference as matrices with numerical values of spatial variability and drift, the type of functions used, the variables i and j, and the locations indexes α and β (or the monomial index l instead β for

the elements that belong to the drift part). This detailed information is necessary if the user want to modify the kriging system of equation in the script.

The implementation of the kriging algorithm is as follow:

1. To create empty kriging matrices, with size *n*=(*index.dim1*()+ *drift.dim1*()); where *index.dim1*() is the number of tuples (x,i) in the mapping function *index*. This mapping function can be filled with the search neighborhood function.
2. To create the empty kriging vectors with size (*n* x *nvar*), where *nvar* is the number of variables in the system. Notice that additionally we pass a TNT Array1D named *nv* with the number of points per variable. If some of the variables has zero points then we rise an error because the cokriging system can get unstable in some kriging models; this checking is optional, but is activated by default, it can be deactivated from Python, passing extra arguments. The array *nv* can be automatically filled by the search neighborhood function existing in the library.
3. To fill the spatial variability part of the kriging matrix with a double for:

```
for (i=0; i<index.dim1(); ++i)
{
   li=index[i][0];
   vi=index[i][1];
   for (ii=i; ii<index.dim1(); ++ii)
   {
      lii=index[ii][0];
      vii=index[ii][1];
      K_matrix[i][ii]=calc_kovar(...);
      K_matrix[ii][i]=K_matrix[i][ii];
      ...
   }
}
```

   where li and lii are the coordinates indexes at points x_α and x_β , vi and vii are the variable identifiers i and j of $Z(x_\alpha,i)$ and $Z(x_\beta,j)$; calc_kovar(...) is a function that returns the bi-variate spatial variability between locations (x_α,i) and (x_β,j).
4. To fill the drift part of the kriging matrix with the double for:

```
for (i=n-drift.dim1(); i<n; ++i)
    for(ii=0; ii<n-drift.dim1(); ++ii)
    {
```

```
            K_matrix[i][ii]=drift_calculator(...);
            K_matrix[ii][i]=K_matrix[i][ii];
            ...
        }
```

where drift_calculator(...) is a function to calculate the drift for each point (x_α,i), n is the matrix size. The function drift_calculator(...) takes as parameters the drift definition, the index l of the monomial m_l, the coordinates (x) of data points, the index α of the points (x_α,i), the values of the external drift $e(x_\alpha,t)$, the index t for the external drift variable to be used, and two variable index i and j_l, obtained from the point (x_α,i) and this one from the monomial $m_l(x_\alpha,i,j_l)$.

5. To fill the spatial variability part of vectors with the double for:

```
for(vii=0; vii<mv_data.dim2(); ++vii)
   for (i=0; i<index.dim1(); ++i)
   {
      li=index[i][0];
      vi=index[i][1];
      K_vector[i][vii]= calc_kovar_vect(...);
      ...
   }
```

where the function calc_kovar_vect(...) is similar to calc_kovar(...) but takes also in to account if the target is in block support and returns the mean spatial variability between a data point and the discretization points of blocks.

6. To fill the drift part of the vectors with the double for:

```
for (i=n-drift.dim1(); i<n; ++i)
      for(ii=0; ii<nvar; ++ii)
      {
            K_vector[i][ii]=drift_calculator(...);
            ...
      }
```

Others functions are defined in the library to solve various problems. For example, the functions string do_K_system_equation_formula(...) and string do_K_system_equation_result(...) return the kriging system as a string with OpenOffice Math format. Others functions located in the source file utils.cc perform diverse utilities, as rotate a vector, test if a variogram is semi-definite positive, etc.. The description of the functions and other symbols are given in the library source code (Appendix A).

RESULTS

The full C++ source and also the compiled shared library for Windows an Linux can be downloaded from the library website (Appendix A). Additionally the Python interface for functions and data definition is provided, as well the Python code for two examples designed to shows the functionality of the library. The first example was designed to show how a complicated model, defined as a mix of common kriging models, can be implemented in the library. The second example was designed to show how an uncommon models can be handled by the library.

Example one: a complicated kriging system

Given a random function $Z(x,i);\ i=0,1,2$ with drift $m(x,i,\mathrm{j})$ defined by three local independent monomials $m_l = \mathbf{1}(i,j_l)x^0$ with $l=0,1,2$ and $i,j_l=0,1,2$, and also two dependent external drifts $e_3(x)$ and $e_4(x)$, we want to estimate $Z(v,0)$ and $Z(v,2)$ in a target location $v=x_1$ with ordinary cokriging with extended collocation of $Z(x,1)$, assuming that $Z(x,1)$ is known at any point of a domain D. Figure 1 shows the spatial arrangement of the data and target.

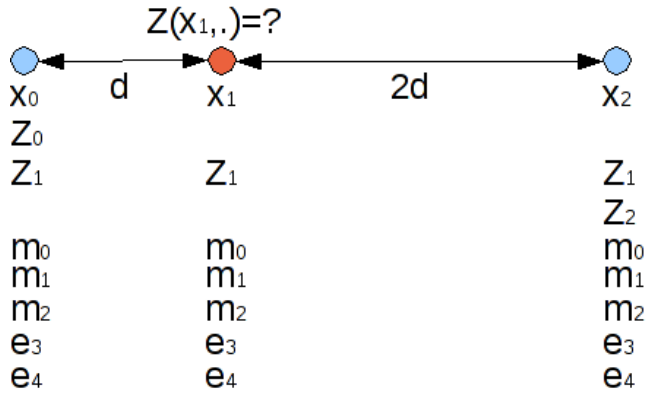

Figure 6. Spatial arrangement of data (x_0, x_1, x_2) and target ($v=x_1$) for the example 1. The distance $\mathrm{d}=1$. The empty spaces below location points correspond to missing data.

We passed to the function that construct the kriging system an uni-dimensional array with data locations $\mathrm{x}=[x_0=-1, x_1=0, x_2=2]$ with data defined as:

$$Z = \begin{bmatrix} 1 & 10 & NaN \\ NaN & 20 & NaN \\ NaN & 30 & 100 \end{bmatrix}$$

where NaN values are “Not a Number” and are regarded as undefined data by the library.

The mapping function "Index" was defined as:

$$Index\left(x\_id, i\_id\right) = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 2 & 2 \end{bmatrix}$$

where x_id and i_id are the indexes that identify the row and the column of the matrices x and Z respectively.

Similarly the external drift is passed as an array, with drift values defined for all points x.

$$e = \begin{bmatrix} 10 & 150 \\ 20 & 210 \\ 30 & 100 \end{bmatrix}$$

The variogram model was defined as an isotropic spherical with range $a = d = 1$

$$\gamma_{sph}\left(h,i,j\right) = \mathbf{B_{i,j}} \begin{cases} \frac{3}{2}\left(\frac{h}{a}\right) - \frac{1}{2}\left(\frac{h}{a}\right)^3 & \text{if } h < a \\ 1 & \text{if } h \geq a \end{cases}$$

where

$$\mathbf{B}_{\mathbf{i,j}} = \begin{bmatrix} 1{,}0 & 0{,}7 & 0{,}8 \\ 0{,}7 & 2{,}0 & 1{,}2 \\ 0{,}8 & 1{,}2 & 3 \end{bmatrix}$$

the model is semi(definite) positive since the eigenvalues of the matrix $\mathbf{B_{i,j}}$ are:

$$eig = \begin{bmatrix} 0{,}62 \\ 1{,}23 \\ 4{,}15 \end{bmatrix}$$

The resulting kriging matrices were automatically generated by the library in OpenOffice Math format as:

$$\begin{bmatrix} \gamma_{0,0}^{0,0} & \gamma_{0,0}^{0,1} & \gamma_{0,1}^{0,1} & \gamma_{0,2}^{0,1} & \gamma_{0,2}^{0,2} & f_{0,0}^{0,0} & f_{0,1}^{0,1} & f_{0,2}^{0,2} & e_{0,3} & e_{0,4} \\ \gamma_{0,0}^{1,0} & \gamma_{0,0}^{1,1} & \gamma_{0,1}^{1,1} & \gamma_{0,2}^{1,1} & \gamma_{0,2}^{1,2} & f_{0,0}^{1,0} & f_{0,1}^{1,1} & f_{0,2}^{1,2} & e_{0,3} & e_{0,4} \\ \gamma_{1,0}^{1,0} & \gamma_{1,0}^{1,1} & \gamma_{1,1}^{1,1} & \gamma_{1,2}^{1,1} & \gamma_{1,2}^{1,2} & f_{1,0}^{1,0} & f_{1,1}^{1,1} & f_{1,2}^{1,2} & e_{1,3} & e_{1,4} \\ \gamma_{2,0}^{1,0} & \gamma_{2,0}^{1,1} & \gamma_{2,1}^{1,1} & \gamma_{2,2}^{1,1} & \gamma_{2,2}^{1,2} & f_{2,0}^{1,0} & f_{2,1}^{1,1} & f_{2,2}^{1,2} & e_{2,3} & e_{2,4} \\ \gamma_{2,0}^{2,0} & \gamma_{2,0}^{2,1} & \gamma_{2,1}^{2,1} & \gamma_{2,2}^{2,1} & \gamma_{2,2}^{2,2} & f_{2,0}^{2,0} & f_{2,1}^{2,1} & f_{2,2}^{2,2} & e_{2,3} & e_{2,4} \\ f_{0,0}^{0,0} & f_{0,0}^{1,0} & f_{1,0}^{1,0} & f_{2,0}^{1,0} & f_{2,0}^{2,0} & 0 & 0 & 0 & 0 & 0 \\ f_{0,1}^{0,1} & f_{0,1}^{1,1} & f_{1,1}^{1,1} & f_{2,1}^{1,1} & f_{2,1}^{2,1} & 0 & 0 & 0 & 0 & 0 \\ f_{0,2}^{0,2} & f_{0,2}^{1,2} & f_{1,2}^{1,2} & f_{2,2}^{1,2} & f_{2,2}^{2,2} & 0 & 0 & 0 & 0 & 0 \\ e_{0,3} & e_{0,3} & e_{1,3} & e_{2,3} & e_{2,3} & 0 & 0 & 0 & 0 & 0 \\ e_{0,4} & e_{0,4} & e_{1,4} & e_{2,4} & e_{2,4} & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_0^0 & \lambda_0^0 & \lambda_0^0 \\ \lambda_0^1 & \lambda_0^1 & \lambda_0^1 \\ \lambda_1^1 & \lambda_1^1 & \lambda_1^1 \\ \lambda_2^1 & \lambda_2^1 & \lambda_2^1 \\ \lambda_2^2 & \lambda_2^2 & \lambda_2^2 \\ \mu_0 & \mu_0 & \mu_0 \\ \mu_1 & \mu_1 & \mu_1 \\ \mu_2 & \mu_2 & \mu_2 \\ \mu_3 & \mu_3 & \mu_3 \\ \mu_4 & \mu_4 & \mu_4 \end{bmatrix}_{V_0} \begin{bmatrix} \lambda_0^0 \\ \lambda_0^1 \\ \lambda_1^1 \\ \lambda_2^1 \\ \lambda_2^2 \\ \mu_0 \\ \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \end{bmatrix}_{V_1} \begin{bmatrix} \lambda_0^0 \\ \lambda_0^1 \\ \lambda_1^1 \\ \lambda_2^1 \\ \lambda_2^2 \\ \mu_0 \\ \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \end{bmatrix}_{V_2} = \begin{bmatrix} \overline{\gamma}_{v,0}^{0,0} \\ \overline{\gamma}_{v,0}^{0,1} \\ \overline{\gamma}_{v,1}^{0,1} \\ \overline{\gamma}_{v,2}^{0,1} \\ \overline{\gamma}_{v,2}^{0,2} \\ f_{v,0}^{0,0} \\ f_{v,1}^{0,1} \\ f_{v,2}^{0,2} \\ e_{v,3} \\ e_{v,4} \end{bmatrix}_{B_0} \begin{bmatrix} \overline{\gamma}_{v,0}^{1,0} \\ \overline{\gamma}_{v,0}^{1,1} \\ \overline{\gamma}_{v,1}^{1,1} \\ \overline{\gamma}_{v,2}^{1,1} \\ \overline{\gamma}_{v,2}^{1,2} \\ f_{v,0}^{1,0} \\ f_{v,1}^{1,1} \\ f_{v,2}^{1,2} \\ e_{v,3} \\ e_{v,4} \end{bmatrix}_{B_1} \begin{bmatrix} \overline{\gamma}_{v,0}^{2,0} \\ \overline{\gamma}_{v,0}^{2,1} \\ \overline{\gamma}_{v,1}^{2,1} \\ \overline{\gamma}_{v,2}^{2,1} \\ \overline{\gamma}_{v,2}^{2,2} \\ f_{v,0}^{2,0} \\ f_{v,1}^{2,1} \\ f_{v,2}^{2,2} \\ e_{v,3} \\ e_{v,4} \end{bmatrix}_{B_2}$$

$$\begin{bmatrix} 0.0 & 0.0 & 0.7 & 0.7 & 0.8 & 1.0 & 0.0 & 0.0 & 10.0 & 150.0 \\ 0.0 & 0.0 & 2.0 & 2.0 & 1.2 & 0.0 & 1.0 & 0.0 & 10.0 & 150.0 \\ 0.7 & 2.0 & 0.0 & 2.0 & 1.2 & 0.0 & 1.0 & 0.0 & 20.0 & 210.0 \\ 0.7 & 2.0 & 2.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 30.0 & 100.0 \\ 0.8 & 1.2 & 1.2 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 30.0 & 100.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 1.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 10.0 & 10.0 & 20.0 & 30.0 & 30.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 150.0 & 150.0 & 210.0 & 100.0 & 100.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \begin{bmatrix} \lambda_0^0 & \lambda_0^0 & \lambda_0^0 \\ \lambda_0^1 & \lambda_0^1 & \lambda_0^1 \\ \lambda_1^1 & \lambda_1^1 & \lambda_1^1 \\ \lambda_2^1 & \lambda_2^1 & \lambda_2^1 \\ \lambda_2^2 & \lambda_2^2 & \lambda_2^2 \\ \mu_0 & \mu_0 & \mu_0 \\ \mu_1 & \mu_1 & \mu_1 \\ \mu_2 & \mu_2 & \mu_2 \\ \mu_3 & \mu_3 & \mu_3 \\ \mu_4 & \mu_4 & \mu_4 \end{bmatrix}_{V_0} \begin{bmatrix} \lambda_0^0 \\ \lambda_0^1 \\ \lambda_1^1 \\ \lambda_2^1 \\ \lambda_2^2 \\ \mu_0 \\ \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \end{bmatrix}_{V_1} \begin{bmatrix} \lambda_0^0 \\ \lambda_0^1 \\ \lambda_1^1 \\ \lambda_2^1 \\ \lambda_2^2 \\ \mu_0 \\ \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \end{bmatrix}_{V_2} = \begin{bmatrix} 1.0 \\ 0.7 \\ 0.0 \\ 0.7 \\ 0.8 \\ 1.0 \\ 0.0 \\ 0.0 \\ 20.0 \\ 200.0 \end{bmatrix}_{B_0} \begin{bmatrix} 0.7 \\ 2.0 \\ 0.0 \\ 2.0 \\ 1.2 \\ 0.0 \\ 1.0 \\ 0.0 \\ 20.0 \\ 200.0 \end{bmatrix}_{B_1} \begin{bmatrix} 0.8 \\ 1.2 \\ 0.0 \\ 0.0 \\ 3.0 \\ 0.0 \\ 0.0 \\ 1.0 \\ 20.0 \\ 200.0 \end{bmatrix}_{B_2}$$

where $\gamma_{\alpha,\beta}^{i,j}$ is the variogram between the points $Z(x_\alpha, i)$ and $Z(x_\beta, j)$; $f_{\alpha,l}^{j,i}$ is the geographical monomial l, of the drift $m(x_\alpha, i, \mathrm{j})$; $e_{\alpha,l}$ is the external drift monomial l at point x_α. The external drifts $e_{\alpha,l}$ were written without coefficients i, j to highlight that are dependent monomials. The target location is represented by v; λ_α^i are the weights associated to $Z(x_\alpha, i)$ and μ_l the Lagrange multiplier associated to the monomial l.

The resulting matrix also were passed as arrays of doubles. Solving the system from the Python, with the module *numpy*, was obtained:

$$\begin{bmatrix} \lambda_0^0 & \lambda_0^0 & \lambda_0^0 \\ \lambda_0^1 & \lambda_0^1 & \lambda_0^1 \\ \lambda_1^1 & \lambda_1^1 & \lambda_1^1 \\ \lambda_2^1 & \lambda_2^1 & \lambda_2^1 \\ \lambda_2^2 & \lambda_2^2 & \lambda_2^2 \\ \mu_0 & \mu_0 & \mu_0 \\ \mu_1 & \mu_1 & \mu_1 \\ \mu_2 & \mu_2 & \mu_2 \\ \mu_3 & \mu_3 & \mu_3 \\ \mu_4 & \mu_4 & \mu_4 \end{bmatrix}_{V_1} \begin{bmatrix} \lambda_0^0 \\ \lambda_0^1 \\ \lambda_1^1 \\ \lambda_2^1 \\ \lambda_2^2 \\ \mu_0 \\ \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \end{bmatrix}_{V_2} \begin{bmatrix} \lambda_0^0 \\ \lambda_0^1 \\ \lambda_1^1 \\ \lambda_2^1 \\ \lambda_2^2 \\ \mu_0 \\ \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \end{bmatrix}_{V_3} = \begin{bmatrix} 1.00 \\ -0.94 \\ 0.88 \\ 0.06 \\ 0.00 \\ -3.60 \\ -5.12 \\ -5.17 \\ 0.11 \\ 0.02 \end{bmatrix} \begin{bmatrix} 0.00 \\ 0.06 \\ 0.88 \\ 0.06 \\ 0.00 \\ 0.77 \\ 0.84 \\ 0.80 \\ -0.01 \\ 0.00 \end{bmatrix} \begin{bmatrix} 0.00 \\ 0.06 \\ 0.88 \\ -0.94 \\ 1.00 \\ -1.30 \\ -1.23 \\ 1.33 \\ -0.02 \\ 0.01 \end{bmatrix}$$

Notice that also the result for $Z(v = x_{1,}1)$ is given, and $\lambda_1^1 \neq 1$ because:

$$\sum_{(\alpha,i)} \lambda_\alpha^i f_{\alpha,l}^{j,i} = f_{v,l}^{k,i}.$$

If the external drifts monomials are removed the solution of the system is as follow:

$$\begin{bmatrix} \lambda_0^0 \\ \lambda_0^1 \\ \lambda_1^1 \\ \lambda_2^1 \\ \lambda_2^2 \\ \mu_0 \\ \mu_1 \\ \mu_2 \end{bmatrix}_{V_1} \begin{bmatrix} \lambda_0^0 \\ \lambda_0^1 \\ \lambda_1^1 \\ \lambda_2^1 \\ \lambda_2^2 \\ \mu_0 \\ \mu_1 \\ \mu_2 \end{bmatrix}_{V_2} \begin{bmatrix} \lambda_0^0 \\ \lambda_0^1 \\ \lambda_1^1 \\ \lambda_2^1 \\ \lambda_2^2 \\ \mu_0 \\ \mu_1 \\ \mu_2 \end{bmatrix}_{V_3} = \begin{bmatrix} 1.00 \\ -0.35 \\ 0.35 \\ 0.00 \\ 0.00 \\ 0.76 \\ 0.00 \\ 0.00 \end{bmatrix} \begin{bmatrix} 0.00 \\ 0.00 \\ 1.00 \\ 0.00 \\ 0.00 \\ 0.00 \\ 0.00 \\ 0.00 \end{bmatrix} \begin{bmatrix} 0.00 \\ 0.00 \\ 0.60 \\ -0.60 \\ 1.00 \\ 0.00 \\ 0.00 \\ 2.28 \end{bmatrix}$$

Notice that the kriging matrix can be singular if the values of the external drift monomials are collinear for example if:

$$e = \begin{bmatrix} 10 & 100 \\ 20 & 200 \\ 30 & 300 \end{bmatrix}$$

If the values are not collinear but almost, then we can have serious stability problems.

It is important to highlight that the library can construct many different kriging systems, but if the model that underline this kriging system is correct or not is not tested by the library. The general shape of the kriging system must be deduced analytically for new models, in order to prove that it is correct and to underline the exception of validity.

Example two: cokriging grades measured in two different exploration campaigns

This example was designed to show that the library can handle non-conventional model of kriging. The problem was stated as:

> In a mineral deposit grades z are measured in two different exploration campaigns: campaign 0 and campaign 1. The samples in the campaign 0 were

measured with high precision, let say with measurement error equal to zero. The samples in campaign 1 were measured with a quick field method, with an unknown but non systematic measurement error.

A suitable model in this case is to consider two random functions $Z(x,0)$ and $Z(x,1)$ for grades z in the campaigns 0 and 1. It is evident that the mean $m_z(x)$ for both random functions is the same, then if we do ordinary cokriging the drift coefficient for both RF are shared, that means that we have a drift with a unique dependent monomial $m_0(x,i,j_0)=1 \forall i,j_0$. A solution to this problem was given by Chilès & Delfiner (1999, p. 313), but is not a common method, then, at the moment is not implemented in any software we know.

In this problem the data is not coincident, let say that samples from different campaign are not in the same location. That means that we are dealing with pure heterotopy. If we suppose we know that $\gamma^{11}(h)=\gamma^{00}(h)+\gamma_e(0)$, where $\gamma_e(0)$ is a nugget model associate to the error with sill equal to two, and $\gamma^{00}(h)$ is an spherical $\gamma_{sph}(h)$ with sill equal to one and range $d=1$. If we also assume none spatial correlation between the error in $Z(x,1)$ and $Z(x,0)$ then the variogram is:

$$\gamma(h,i,j)=\mathbf{B}^{\mathbf{0}}_{\mathbf{i,j}}\gamma_{sph}(h)+\mathbf{B}^{\mathbf{1}}_{\mathbf{i,j}}\gamma_e(h)$$

with matrices

$$\mathbf{B}^{\mathbf{0}}_{\mathbf{i,j}}=\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \qquad \mathbf{B}^{\mathbf{1}}_{\mathbf{i,j}}=\begin{bmatrix}0 & 0\\ 0 & 2\end{bmatrix}$$

with eigenvalues $eig(\mathbf{B}^{\mathbf{0}}_{\mathbf{i,j}})=[2 \quad 0]$ and $eig(\mathbf{B}^{\mathbf{1}}_{\mathbf{i,j}})=[0 \quad 2]$ respectively.

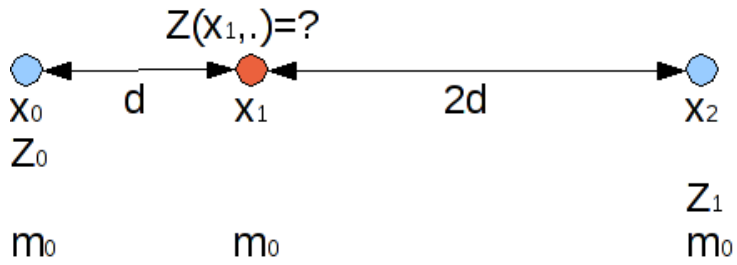

Figure 7. Spatial arrangement of data points at ($x_{0,}x_2$) and target ($v=x_1$) for the example 2. The distance d=1. The empty spaces below location points correspond to missing data.

Given the spatial distribution of the data, as shows Figure 7, the resulting kriging system is:

$$\begin{bmatrix} \gamma_{0,0}^{0,0} & \gamma_{0,1}^{0,1} & f_{0,0} \\ \gamma_{1,0}^{1,0} & \gamma_{1,1}^{1,1} & f_{1,0} \\ f_{0,0} & f_{1,0} & 0 \end{bmatrix} \begin{bmatrix} \lambda_0^0 \\ \lambda_1^1 \\ \mu_0 \end{bmatrix}_{V_0} \begin{bmatrix} \lambda_0^0 \\ \lambda_1^1 \\ \mu_0 \end{bmatrix}_{V_1} = \begin{bmatrix} \gamma_{v,0}^{-0,0} \\ \gamma_{v,1}^{-0,1} \\ f_{v,0} \end{bmatrix}_{B_0} \begin{bmatrix} \gamma_{v,0}^{-1,0} \\ \gamma_{v,1}^{-1,1} \\ f_{v,0} \end{bmatrix}_{B_1}$$

$$\begin{bmatrix} 0.0 & 1.0 & 1.0 \\ 1.0 & 0.0 & 1.0 \\ 1.0 & 1.0 & 0.0 \end{bmatrix} \begin{bmatrix} \lambda_0^0 \\ \lambda_1^1 \\ \mu_0 \end{bmatrix}_{V_0} \begin{bmatrix} \lambda_0^0 \\ \lambda_1^1 \\ \mu_0 \end{bmatrix}_{V_1} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}_{B_0} \begin{bmatrix} 1.0 \\ 3.0 \\ 1.0 \end{bmatrix}_{B_1}$$

Notice that now the index $\alpha = 1$ is associated to the point x_2, because we pass locations as an array with two elements $[x_0 = -1, x_2 = 2]$ and we count array elements from zero.

The solution for this system is as follow:

$$\begin{bmatrix} \lambda_0^0 \\ \lambda_1^1 \\ \mu_0 \end{bmatrix}_{V_0} \begin{bmatrix} \lambda_0^0 \\ \lambda_1^1 \\ \mu_0 \end{bmatrix}_{V_1} = \begin{bmatrix} 0{,}5 \\ 0{,}5 \\ 0{,}5 \end{bmatrix}_{B_0} \begin{bmatrix} 1{,}5 \\ -0{,}5 \\ 1{,}5 \end{bmatrix}_{B_1}$$

Notice that the weight attached for both data points in the estimation of the variable without error ($Z(v,0)$) is the same. Surprisingly the solution vector for $Z(v,1)$ give a reduced power to $Z(x_1,1)$ and a large power to $Z(x_0,0)$, filtering in some way the error.

CONCLUSIONS

The proposed algorithm, based on RFs defined as $Z(x,i)$, with drift $m(x,i,\mathrm{j})$, can handle a large number of kriging models, included new, uncommon and complex models. That is possible because the source code that build the kriging system of equations was not predefined for a set of specific kriging models. Are the users whom define the kriging model, passing the appropriated set of data and parameters to the function responsible to build the kriging system. It is also possible because users have access to the kriging system of equations, which can be modified before be solved.

APPENDIX

The source code of the library can be downloaded from the website http://OpenKriging.webs.com

Some links with modules recommended or necessary to work with the library are:

Numpy http://numpy.scipy.org/
Matplotlib http://matplotlib.sourceforge.net/
Visual Python http://vpython.org/

REFERENCES

CHILÈS JEAN-PAUL, DELFINER PIERRE. 1999: Geostatistics: *Modeling Spatial Uncertainty.* Jonh Wiley & Sons Inc., New York, 695 p.

BLEINÈS CATHERINE, DERAISME JACQUES, FRANÇOIS GEFFROY, JEANNÉE NICOLAS, PERSEVAL SÉBASTIEN, RAMBERT FRÉDÉRIC, RENARD DIDIER, TORRES OLIVER, TOUFFAIT YVES 2007: *Isatis Technical References, version 7.0. GEOVARIANCES, 2007*. 138 p.

BROKKEN FRANK B., 2008: *C++ Annotations Version 7.2.0.* University of Groningen, Netherlands. ISBN 90 367 0470 7. http://www.icce.rug.nl/documents/

CHRISTENSEN O.F., RIBEIRO-JR. P.J. 2002: geoRglm: A package for generalised linear spatial models. *R-News* 2(2): 26-28.

DEUTSCH, CLAYTON V. AND ANDRÉ G. JOURNEL. 1998: "*GSLIB: Geoestatistical software library and User's Guide*". Oxford University Press, New York, 1998, 369 p.

DIGGLE PETER J., RIBEIRO-JR. PAULO J. 2007: *Model-based Geostatistics.* Springer Science+Business Media, LLC, 2007. ISBN-10: 0-387-32907-2. 228 p.

FREE SOFTWARE FOUNDATION. 2007: *GNU General Public License,* Version 3, 29 June 2007. Copyright © 2007 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110, USA. http://www.gnu.org/licenses/gpl.html

MEYER-CARL, D. 2000: Matrix analysis and applied linear algebra. Society for industrial and applied mathematics, 3600 University City Science Center. Philadelphia, PA 19104-2688 ISBN 0-8987|-454-0.

MONSALVE-TOBON JUAN ESTEBAN, 2005. JAMA/C++ Linear Algebra Package, version 1.2. The MathWorks and the National Institute of Standards and Technology (NIST).http://math.nist.gov/tnt/jama_doxygen/index.html, http://math.nist.gov/tnt/jama125.zip

NUMPY COMMUNITY, 2009: NumPy User Guide Release 1.4.0.dev7335. http://www.scipy.org/

OLIPHANT, TRAVIS E. 2006: Guide to NumPy. PhD. 378 pages.

PEBESMA, EDZER J. 2004: Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*. Vol 30, ISSN: 0098-3004, p. 683-691.

POZO, ROLDAN. 2004: The *Template Numerical Toolkit* (TNT). Mathematical and Computational Sciences Division, National Institute of Technology, Gaithersburg, MD USA. http://math.nist.gov/tnt/index.html, http://math.nist.gov/tnt/tnt_3_0_12.zip

REMY, NICOLAS. 2001: GsTl: The geostatistical template library in C++. Department of petroleum engineering, Stanford University. [Master of science these]142 pages.

REMY, NICOLAS; BOUCHER, ALEXANDRE & WU, JIANBING. 2006: SGeMS User's Guide. 128 pages.

RIBEIRO-JR., P.J.; DIGGLE, P.J. 2001: geoR: A package for geostatistical analysis. *R-News* 1(2): 15-18.

TRAUTH, MARTIN H. 2006: MATLAB® Recipes for Earth Sciences. Springer-Verlag Berlin Heidelberg New York, 237 p.

Adrian Martínez Vargas adriangeologo@yahoo.es
Doctor en Ciencias Geológicas.
Profesor Asistente. Departamento de Geología.
Instituto Superior Minero Metalúrgico de Moa, Cuba.