GSTL: the geostatistical template library in C++

Nicolas Remya,*, Arben Shtuka, Bruno Levy, Jef Caers

a Department of Geological and Environmental Sciences, Stanford University, Braun Hall, Building 320, Stanford, CA 94305-2115, USA
b Ecole Superieure de Geologie, A.S.G.A.-Gocad, Batiment G, BP 40- rue du Doyen Marcel Roubault, 54501 Vandoeuvre-les-Nancy, Cedex, France
c Institut de Recherche en Informatique et en Automatique, INRIA Lorraine/Loria, Technopole de Nancy Brabois-Campus scientifique, 615, rue du Jardin Botanique, 54602 Villers-les-Nancy Cedex, France
d Department of Petroleum Engineering, Stanford University, Stanford, CA 94305-2220, USA

Received 3 October 2001; received in revised form 18 March 2002; accepted 20 March 2002

Abstract

The development of geostatistics has been mostly accomplished by application-oriented engineers in the past 20 years. The focus on concrete applications gave birth to many algorithms and computer programs designed to address different issues, such as estimating or simulating a variable while possibly accounting for secondary information such as seismic data, or integrating geological and geometrical data. At the core of any geostatistical data integration methodology is a well-designed algorithm. Yet, despite their obvious differences, all these algorithms share many commonalities on which to build a geostatistics programming library, lest the resulting library is poorly reusable and difficult to expand.

Building on this observation, we design a comprehensive, yet flexible and easily reusable library of geostatistics algorithms in C++. The recent advent of the generic programming paradigm allows us elegantly to express the commonalities of the geostatistical algorithms into computer code. Generic programming, also referred to as “programming with concepts”, provides a high level of abstraction without loss of efficiency. This last point is a major gain over object-oriented programming which often trades efficiency for abstraction. It is not enough for a numerical library to be reusable, it also has to be fast.

Because generic programming is “programming with concepts”, the essential step in the library design is the careful identification and thorough definition of these concepts shared by most geostatistical algorithms. Building on these definitions, a generic and expandable code can be developed.

To show the advantages of such a generic library, we use GSTL to build two sequential simulation programs working on two different types of grids—a surface with faults and an unstructured grid—without requiring any change to the GSTL code. © 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Generic programming; Programming library; Stochastic simulation; Estimation; C++

1. Introduction

The development of geostatistics has been mostly accomplished by application-oriented engineers over the past 20 years. The focus on concrete applications gave birth to a diversity of algorithms designed to address different issues, such as estimating or simulating a variable, at the same time as, for example, accounting...
for secondary information such as seismic data, or integrating geological and geometrical data.

In order to test and then apply these algorithms to real examples, they must be coded into a programming language. Making a computer executable publicly available plays a vital role in popularizing an algorithm. However, despite the essential place of programmed algorithms in geostatistics, no programming library that implements the basic tools and algorithms of geostatistics exists (at least no such library is publicly available).

The main programming effort in geostatistics made publicly available is GSLIB (Deutsch and Journel, 1992), the Geostatistical Software Library. GSLIB, as its name suggests, is a collection of software, not a programming library: it provides a variety of computer executables which implement a broad family of algorithms, but it does not provide an optimal framework for programming and adding new software.

The purpose of this work is to propose a genuine programming library of geostatistical tools and algorithms, in C++. C++ was chosen because it is a high-level programming language whose usage is now widespread.

The design of the library was governed by three main goals:

- As for any scientific computing library, performance is a key concern.
- The new library should allow an efficient reuse of existing code.
- It should be easily extensible.

In order to achieve these goals, we propose to base the library design on Generic Programming. Generic Programming is a new programming paradigm that enables the attainment of a high level of abstraction without sacrificing efficiency. Instead of working directly with actual data types (“classes” in C++), a generic algorithm works on abstractions or concepts, which are assumed to have precise properties. The key step in the design of a generic library for geostatistics is thus the precise identification of the concepts required by the geostatistical algorithms and the definition of the minimal set of properties these concepts must possess.

2. Generic programming

Consider the example of the sequential Gaussian simulation algorithm for a Gaussian variable (Ripley, 1987; Journel, 1989; Isaaks, 1990). The core “idea” of sequential Gaussian simulation is to simulate a series of values by sequential drawing from Gaussian distributions whose parameters are determined through kriging. It can be summarized as follows:

1. define a path visiting all the nodes of the simulation grid;
2. for each node \( u \) in the path;
   a. find the node’s informed neighbors. The neighbors can be nodes from the original data set \( n \), or nodes simulated at previous iterations \( l \).
   b. estimate the Gaussian cumulative distribution \( G^*(u; y \mid (n + l)) \) at \( u \) conditional to the neighbors \( (n + l) \) by solving a kriging system. The mean of \( G^*(u; y \mid (n + l)) \) is the kriging estimate and its variance is the kriging variance.
   c. draw a realization from \( G^*(u; y \mid (n + l)) \) by Monte-Carlo simulation, and assign the simulated value to the node.

An implementation of this algorithm for a Cartesian grid would unnecessarily restrict its potential domain of application. The sequential Gaussian simulation algorithm does not require the grid to be Cartesian. As long as a path through all the grid nodes can be defined, this algorithm can be applied to any type of grid, be it Cartesian or unstructured, 1D, 3D or nD.

Similarly, the path defined at the beginning of the algorithm is usually taken to be random in practical applications. However this is not imposed by the algorithm, and one could choose a path that visits preferentially nodes close to the original set of data.

A truly generic implementation of the sequential simulation Gaussian algorithm should, therefore, be independent of the type of the grid or the type of the path.

In modern computing, the most popular method of achieving this aim is to use object-oriented programming. In object-oriented programming, the generic character of the implementation is provided through the use of inheritance and dynamic binding. The algorithm is written for abstract types (or objects), e.g. an “AbstractGrid”, an “AbstractPath”, and will work on objects that represent particular instances of these abstract objects: the algorithm would be defined in terms of “AbstractGrid”; but would be used on “CartesianGrid” or “UnstructuredGrid” which are particular types of grid that inherit from “AbstractGrid”.

This approach is most useful when the relevant entities are similar but are not identical, i.e. when they can be grouped into object hierarchies. If this is not the situation, then forcing an object-oriented approach, i.e. forcing a taxonomy of the entities, leads to an awkward design. The use of inheritance and dynamic binding also has a major drawback in scientific programming: it induces non-negligible run-time overhead which can badly affect the overall performance of the program.

Object-Oriented is not the only way of achieving a high level of abstraction. Generic programming is a relatively new programming paradigm that allows the
program implementation to be separated or abstracted in an elegant way from any unnecessary information. Instead of working directly with actual data types ("classes" in C++), a generic algorithm works on abstractions (often called concepts) which are assumed to have precise properties (the fewer the assumed properties, the more generic the implementation). A generic algorithm is thus made of two parts: an actual program code, and a list of all the assumed properties of the abstractions used. This list of properties is not C++ code, yet it is an integral part of the algorithm. These properties are the hypotheses of the algorithm. Omitting them is as damaging as omitting to state the hypotheses of a mathematical theorem.

To illustrate how this works, consider the simple example of finding the maximum of a set of elements. The set could be an array, a linked list, ..., and its elements real numbers, strings, cars, ... . To find the maximum of this set, only the following is required:

1. a method to go from one element of the set to another;
2. an order relation is defined on the elements of the set, and given two elements, a method to compare them.

The algorithm would then be implemented as follows:

```cpp
template < class iterator, class comparator >
iterator find_maximum(iterator first,
                        iterator last,
                        comparator greater) {
  // initialize iterator max_position, the
  // iterator that points to the largest element
  // found so far
  iterator max_position = first;
  // iterate through the container
  for(iterator current = first ++ ; current != last; current++)
    { if ( greater(*max_position ,
                   *current) )
      max_position = current;
    }
  return max_position;
}
```

The first line indicates that algorithm `find_maximum` refers to two concepts: `iterator` and `comparator`. The algorithm assumes these two concepts have the following properties:

**iterator:** It is the device used to go through the set. An iterator is a classical way to make the code independent of the container (set of elements) to which it is applied. Different kinds of iterators are detailed in Austern (1999). The `find_maximum` algorithm assumes that an iterator has the following properties:

- an iterator can be assigned to another (line 8: `max_position = first`);
- two iterators can be compared using `!=` (line 11: `current != last`);
- operator `++` can be applied to an iterator, and it will move the iterator to the next position in the set of elements (line 11: `current++`);
- operator `*` can be applied to an iterator, and it will return the element to which the iterator is pointing (line 13: `*current`).

**comparator:**

- a comparator has an operator `()` which takes two objects as arguments and returns a type convertible to bool.
- For example: `greater(*max_position ,*current)` (line 12).
- It returns “true” if the first argument is greater than the second.

The previous C++ code and its two sets of requirements form the generic `find_maximum` algorithm. Any C++ object that fulfills the four requirements of concept `iterator` is an eligible iterator for the algorithm and can be an input of `find_maximum`. Such an object is called a model of concept `iterator`. On the other hand, trying to use as an `iterator` an object which does not meet the four requirements of `iterator` will result in a compile-time or link-time error.

Type `double*` is a valid model of `iterator` because it has the four properties required by concept `iterator`. A call to

```cpp
double* an_array;
find_maximum(an_array,
             an_array+10,
             greater_doubles() )
```

will then find the maximum of the array `an_array` which contains 10 elements of type `double`. Here `greater_doubles` is a model of concept `comparator`, i.e. it takes two doubles as argument and returns a type convertible to boolean. Similarly, `find_maximum` can be
applied to a STL list of characters without any change to its implementation, because the STL type list<
char>::iterator has the four properties of iterator. The comparator could be greater, char which
would take two char as argument and return a type convertible to boolean.
Given a set of requirements, one can write any model of the concepts and use them in any generic algorithm
that needs these concepts, without requiring any change to the implementation of the algorithm.
A generic algorithm could be compared to a theorem: the C++ code would be the statement of the theorem,
and the concepts requirements the hypotheses.
How may Generic Programming be used to implement a programming library for geostatistics? The first
step in the design of GSSTL is to analyze the algorithms of least-square regression algorithms (Goovaerts, 1997).

The aim is to:

- estimation, i.e. the mapping of a spatially and/or time dependent variable $z$, through regression techniques;
- simulation, used to assess the uncertainty on a spatially and/or time dependent variable $z$, quantified
through a series of numbers or possible outcomes, allowing risk quantification.

These two applications of geostatistics are reviewed and detailed in the following sections with the purpose
of identifying the key concepts of geostatistics.

3. Overview of the main algorithms of geostatistics

The goal of geostatistics is to study and characterize phenomena that vary in space (and/or time). Geostatis-
tics has two principal applications:

- estimation, i.e. the mapping of a spatially and/or time dependent variable $z$, through regression techniques;
- simulation, used to assess the uncertainty on a spatially and/or time dependent variable $z$, quantified
through a series of numbers or possible outcomes, allowing risk quantification.

3.1. Estimation

Consider a set $U$ of locations in space or time. In practical applications, $U$ is finite, of size $N$. Suppose that
the value of $z$ is known on a subset of $U$. The aim is to estimate the values of $z$, interpreted as the realization of
a regionalized random variable $Z(u)$, at any location $u$ in $U$ given the known $z$-values \{$z(u_x)$, $x = 1, \ldots, n$\}.

For a given loss function $L$, the best estimate $Z^*(u)$ of unknown value $Z(u)$ is the estimate that minimizes the
expected loss:

$$Z^*(u) = \arg\min_z E\{L(z, Z(u))\}.$$  

Kriging is the name of a family of generalized linear least-square regression algorithms (Goovaerts, 1997).
The estimate $Z^*(u)$ is modeled as a linear combination of the known $z$-values \{$z(u_x)$\}:

$$Z^*(u) - m(u) = \sum_{x=1}^n \lambda_x [Z(u_x) - m(u_x)],$$  \tag{1}

where $m(u)$ and $m(u_x)$ are the expected values of $Z(u)$ and $Z(u_x)$.

Under the unbiasedness constraint

$$E(Z^*(u) - Z(u)) = 0$$

minimizing the expected loss amounts to minimizing the error variance

$$\sigma^2_k(u) = \text{Var}(Z^*(u) - Z(u)).$$  \tag{2}

Substituting $Z^*(u)$ in Eq. (2) by its expression Eq. (1) and setting to zero all the derivatives $\partial \sigma^2_k(u)/\partial \lambda_x$ yields
a system of linear equations whose solution are the weights $\lambda_x$, $x = 1, \ldots, n$. The system is of the form

$$
\begin{pmatrix}
(C(u_1, u_1) & \ldots & C(u_1, u_n) \\
\vdots & \ddots & \vdots \\
C(u_n, u_1) & \ldots & C(u_n, u_n)
\end{pmatrix}
\begin{pmatrix}
\lambda_1 \\
\vdots \\
\lambda_n
\end{pmatrix} = 
\begin{pmatrix}
(C(u, u_1) \\
\vdots \\
C(u, u_n)
\end{pmatrix},
$$

where $C(u_i, u_j)$ is the covariance between $Z(u_i)$ and $Z(u_j)$.

Combining the weights $\lambda_1, \ldots, \lambda_n$ according to Eq. (1) provides the best linear least-squares estimate $Z^*(u)$.

To this base (“simple”) kriging system, various linear constraints on the weights $\lambda_1, \ldots, \lambda_n$ can be added: in
kriging with a trend, the mean is unknown and varies smoothly with location:

$$m(u) = \sum_{k=0}^K a_k f_k(u),$$

where $a_k$ are unknown but locally constant and $f_k$ are known functions of $u$. The kriging system at location $u$ is
then given by

$$\text{Var}\left(\sum_{x=1}^n \lambda_x [Z(u_x) - m(u_x)] - [Z(u) - m(u)]\right)$$

is minimum,

$$\sum_{x=1}^n \lambda_x = 1,$$

$$\sum_{x=1}^n \lambda_x f_k(u_x) = f_k(u) \quad \forall k \in [1, K].$$

Kriging can also be made to account for secondary information by extending Eq. (1). Suppose $n_s$ secondary
variables $S_i(u)$, $i = 1, \ldots, n_s$ are to be accounted for,
Eq. (1) becomes
\[ Z^*(u) - m(u) = \sum_{i=1}^{n} \lambda_i [Z(u_i) - m(u_i)] \]
\[ + \sum_{i=1}^{m_1} \sum_{j=1}^{m_2} \lambda_{ij} [S_i(u_{ij}) - m_i(u_{ij})] \]
where \( m_i(u) \) is the expected value of \( S_i(u) \). This version of kriging is called cokriging.

The kriging weights are obtained by minimizing the error variance as defined in Eq. (2). As in the situation of a single variable, different linear constraints on the kriging weights can be added to the "simple" system.

From an algorithmic point of view, kriging and its variants can be decomposed into two parts:

- **a weighting system** which to location \( u \), neighborhood \( V(u) \) and set of covariance functions \( C_{ij} = \text{Cov}(Z(u_i), Z(u_j)) \) (\( Z_i \) can either be \( Z \) or one of the secondary variables \( S_j, k = 1, \ldots, n_e \)) associates a set of kriging weights and a kriging variance (the kriging variance is independent of the values \( z(u) \)):

\[ (u, V(u), \{C_{ij}\}) \mapsto \{\lambda_i\}_{1 \leq i \leq n(u)}, \sigma^2(u). \]

The weighting system consists, in the most general situation, of two parts: a first part accounts for the correlation and the redundancy between the data through the covariance functions, whereas a second part implements the additional constraint equations.

- **a combiner**, which from the previous weights and an a priori mean, computes the kriging estimate:

\[ (\{\lambda_i\}, \{z(u)\}, \mu) \mapsto Z^*(u) \quad 1 \leq z \leq n(u), \]

where \( \mu \) is the a priori mean.

The combiner is a linear combination
\[ \sum_{z=1}^{n(u)} \lambda_z z(u) + \left(1 - \sum_{z=1}^{n(u)} \lambda_z\right) \mu. \]

Note that other types of kriging have been developed, such as block-kriging (see Cressie, 1993; Rivoirard, 1994; Chiles and Delfiner, 1999), which are not covered in the previous overview of kriging, but still follow the previously described scheme. Chapter 6 of Remy (2001) details how kriging can be constrained to a block average value by using the G3TL cokriging algorithm.

**3.2. Simulation**

The aim of simulation is to find a function \( \mathbb{U} \rightarrow \mathbb{E}^N \),
\[ (u_i)_{1 \leq i \leq N} \mapsto z(u_i) \]
such that the sequence of values \( z(u_i), i = 1, \ldots, N, \) honors a set of constraints (\( \mathbb{E} \) is the space in which \( z \) is valued). The constraints can be of various types:

- **Local equality constraints**, or data conditioning: the value of the variable is known at a subset of locations \( (u_j), j = 1, \ldots, K < N. \)
- **Inequality constraints**: the values of the variable must be lesser or greater than a given threshold \( t(u) \) at a subset of locations \( (u_j), j = 1, \ldots, K < N. \)
- **Correlation constraint**: the values of the variable must honor a given model of correlation. Most often a variogram is imposed, but more complicated models, which involve the correlation between more than two locations at a time, could be chosen.
- **Histogram constraints**: the values must match a given histogram which could for example reflect some prior knowledge of variable \( z \).

Because the set of constraints does usually not suffice to characterize fully the sequence \( z(u) \), many solutions exist. Different solutions, termed realizations, provide a model of the uncertainty about the unknown \( Z(u) \).

Four main types of simulation algorithms can be distinguished:

**Sequential simulation.** (Isaaks, 1990; Goméz-Hernández, and Cassiraga, 1994) A path visiting all locations is defined and each location is simulated sequentially. The variable to be simulated is interpreted as a location-dependent random variable \( Z(u) \). At each location \( u \) the cumulative distribution function (cdf) \( F(u, Z|n) \) conditional to some information \( n \), is estimated and sampled. Monte-Carlo Markov Chains (MCMC) simulation (Hastings, 1970), although different from sequential simulation from a theoretical point of view, follow a similar scheme and hence could be implemented the same way. In MCMC simulation, the set of locations to be simulated is initialized with some arbitrary values (random for example). This set of values is then sequentially modified, until it honors the constraints: at a randomly selected location (the same location can be visited multiple times), a conditional cdf (cdf) is estimated and sampled. The new sample value can either be accepted and replace the former value at that location, or be rejected, in which case the location’s value is unchanged.

**P-field.** (Srivastava, 1992) The p-field simulation is divided into two parts: first a cdf \( F(u, Z|n_0) \) conditional to only the original data \( (n_0) \) is estimated at each location \( u \) to be simulated \( (u') \) depends on \( u \) if only the closest original data are retained at each location \( u \). The
family of ccdfs \((F(u, Z | (n_u)))_{u \in U}\) is then sampled using a field of correlated probability values (p-field). The generation of the p-field can be made fast by using methods based on the Fast Fourier Transform (FFT), hence yielding a computationally efficient class of simulation algorithms.

**Boolean simulation.** (Holden et al., 1998) The aim of boolean techniques is to reproduce shapes described by specific parameterizations, which honor the original data \((n')\). For example, it can be used to simulate channels of given sinuosities and extent, or ellipses parameterized by their dimensions and orientations.

**Optimization techniques.** Instead of approaching the simulation problem from a statistical point of view, i.e. interpreting the variable to be simulated as a location-dependent random variable, simulation can be envisioned as an optimization problem: the satisfaction of the constraints is measured through an objective function which must be minimized. Deutsch (1992) proposed to use simulated annealing (Geman and Geman, 1984) to minimize the objective function.

This first release of GsTL focuses on sequential simulation and p-field simulation. These two simulation paradigms interpret the sequence of values \(z(u), i = 1, ..., N\), to be simulated as an outcome of the sequence of random variables \(Z(u), i = 1, ..., N\). The two simulation algorithms proceed as follows:

1. Define a partition \(I = (P_j)_{1 \leq j \leq J}\) of \(\{1, ..., N\}\):
   \[
   \bigcup_{1 \leq j \leq J} P_j = \{1, ..., N\}, \quad \forall j \neq j' \quad P_j \cap P_j = \emptyset.
   \]

2. For each \(P_j\), visited in a pre-defined order,
   (a) for every \(i \in P_j\), estimate the cumulative distribution of \(Z(u)\) conditional to some neighboring data \(V(u)\):
   
   \[
   (u, V(u)) \mapsto F(u, Z | (n(u))),
   \]

   (b) for every \(i \in P_j\), draw a realization from \(F(u, Z | (n(u)))\):
   
   \[
   F(u, Z | (n(u))) \mapsto z(u).
   \]

If the \(P_j\) are singletons, the algorithm described is sequential simulation. If \(I = \{1, ..., N\}\), the algorithm described belongs to the p-field family.

Changing (1) the order of visit of the \(P_j\), (2) the way the cumulative distributions are estimated, and (3) the way new values are deduced from the cdf’s, provide a broad family of algorithms.

### 3.2.1. Order of visit of the \(P_j\)

In p-field simulation, there is only one set of indices \(P_1 (J = 1)\), hence there is no order to decide.

In practical applications of sequential simulation, a random path is usually chosen. However, other types of path could be used, for example, a path that would preferentially visit locations close to the original data, so as to increase the weight of the original data and possibly improve the data conditioning.

### 3.2.2. Estimation of the conditional cdf’s

Two approaches can be distinguished:

- A first possibility is to build the cdf from estimated values. If the variable \(Z(u)\) is multi-Gaussian, all cdf \(F(u, Z | (n_u))\) are also Gaussian, and it suffices to estimate its mean and variance. When no Gaussian assumption is made, the cdf is estimated for given \(z\)-values \(z_1, ..., z_k\) and an interpolation of these \(F^*_Z(u, z_i | (n))\) yield a model of the function \(z \mapsto F_Z(u, z | (n))\).

- Most simulation algorithms estimate these values by kriging. In the situation of a Gaussian cdf, the mean is the kriging estimate, and the variance the kriging variance. In the non-parametric situation, the probabilities \(F_Z(u, z_i | (n))\) are estimated by kriging the indicator random variable \(I(u, z_i)\) defined as follows:

   \[
   I(u, z_i) = \begin{cases} 
   1, & \text{if } z(u) \leq z_i, \\
   0, & \text{otherwise}.
   \end{cases}
   \]

   The conditional probability \(F(u, z_i | (n))\) is indeed equal to the conditional expectation of \(I(u, z_i)\):

   \[
   F_Z(u, z_i | (n)) = E[I(u, z_i) | (n)]
   \]

   and the least-squares estimate of the indicator \(I(u, z_i)\) is also the kriging (least-squares) estimate of its conditional expectation (Luenberger, 1969).

- A second possibility is to infer the ccdf directly from the neighboring information, i.e. no estimation of parameters of a ccdf is required. The cdf can for example be read from a table whose entries are the conditioning data values and geometry. It is the method used in the sequential normal equation simulation (SNESSIM) algorithm (Strebelle, 2000). The ccdf can also be inferred by a classification algorithm such as a neural network (Caers and Journel, 1998).

### 3.3.3. Drawing new values

The new simulated value is usually obtained by drawing a value from the cdf, using uncorrelated random probabilities. This is the technique used in sequential Gaussian simulation, sequential indicator simulation or SNESSIM. However, it is not the sole option. The p-field technique uses a field of correlated “random” probabilities to draw from the cdf’s. Another possibility would be to use a Metropolis–Hastings sampling scheme: a new value is drawn from a cdf using uncorrelated random probabilities, but it does not...
4. Concepts and algorithms

From the previous brief description of the different families of geostatistics algorithms, certain concepts common to most, if not all algorithms emerge:

- A location: coordinates in space or time.
- A geo-value: a location plus a single property value.
- A geo-value iteratator: the device that controls the path through the set of geo-values to be simulated or estimated. It is the interface between the algorithm and the grid of geo-values.
- A neighborhood: most generally, only the data closest to the location of interest are taken into account in order to decrease the computation cost. However, if speed is not an issue, the neighborhood can be made large enough always to include all the available data.
- A cdf. It is either parametric (Gaussian, …) or non-parametric, i.e. defined by a finite set of values $F_Z(Z_i)$ at thresholds $Z_i: (z_i, F_Z(z_i))$.
- A cdf-estimator: to provide an estimate of the cdf, be it marginal or conditional. An estimator can either directly estimate a cdf given a node and its neighborhood as in SNESIM (in SNESIM, the cdf is read from a table whose entries are the neighborhood geometry and the neighboring data values), or builds the cdf from estimated values, using kriging for example, as in sequential Gaussian or indicator simulation.
- A sampler: determines the new simulated value given a cdf.

These concepts, along with others more specific to certain algorithms, are thoroughly described in Remy (2001) and are available on the web.¹

Building on these key concepts, the following algorithms are implemented:

- Kriging weights: computes the kriging weights at a given location. The same algorithm allows simple kriging, ordinary kriging or kriging with trend to be performed, depending on the set of linear constraints input to the algorithm.
- Cokriging: computes the cokriging weights at a given location (Wackernagel, 1995). The same algorithm allows simple kriging or ordinary kriging to be performed, in each situation using the full cokriging system, hence requiring all the covariances and cross-covariances between all the variables, or using collocated cokriging and Markov models MM1 and MM2 (see, Rivoirard, 1994; Goovaerts, 1997) to reduce the number of cross-covariances to be inferred.
- Sequential simulation: this algorithm allows a variable on a set of locations to be simulated sequentially. Depending on the Cdf Estimator given as an argument to the algorithm, the algorithm can perform sequential Gaussian simulation, sequential indicator simulation or single normal equation simulation (multiple-point statistics-based algorithm).
- p-field simulation.
- Miscellaneous algorithms, such as a cdf transform algorithm which transforms a set of values so that their final cdf is a given target cdf.

These algorithms are fully documented in Remy (2001) and on the web (see footnote 1). The source code is available from the SCRF (Stanford Center for Reservoir Forecasting) web site.²

5. Application: simulation on complex geometries in the gOcad software

In order to illustrate the generic character of the library, GSTL algorithms are applied on grids implemented outside the GSTL framework.

Because the GSTL algorithms are, in particular, dissociated from the actual structure of the grid to which they might be applied, they can be used with any type of objects that satisfy their requirements. If some objects do not meet the requirements of an algorithm, “wrapper” classes must be implemented, which modify the former behavior of the object to make it compliant with the GSTL requirements.

Sequential Gaussian simulation is performed on a triangulated surface (see Fig. 1A) using the GSTL. Two types of neighborhoods are given as argument to the simulation algorithm:

- neighborhood A: location v is a neighbor of u if $d(u,v) < R$, where $d$ is the variogram distance.
- neighborhood B: location v is a neighbor of u if $d(u,v) < R$ and there is no fault between u and v. With this type of neighborhood, the simulated property will be discontinuous across the faults.

Fig. 1B shows the result with neighborhood A. The results obtained with neighborhood B (all other parameters being the same) are shown on Fig. 1C.

Using the exact same sequential simulation algorithm of G5TL, a Gaussian simulation is performed on a gOcad T-solid, i.e. an unstructured grid with polyhedra cells. Fig. 2A and B shows the results obtained, respectively, with neighborhoods A and B.

Working directly on these complex grids, hence, allows some important geometrical features (e.g. faults) to be incorporated into the model, which was not feasible with the traditional approach. In the traditional approach, the properties are simulated or estimated on a Cartesian grid and then transported to a complex grid. Such a methodology does not allow geometrical constraints like faults to be modelled.

It should be stressed that in all examples shown, exactly the same simulation C++ code is used, and no modifications are required.

6. Conclusion

G5TL is a C++ library of geostatistical algorithms and tools. It is flexible and covers the most frequently used geostatistical algorithms. The library has three major components: the source code of the geostatistical algorithms, the detailed description of the requirements on the concepts used by the algorithms, and a collection of ready-to-use models of the concepts, i.e. actual C++ objects.

In contrast to the two other components, the description of the concepts is not C++ code. It is simply a textual description of the assumptions made by the G5TL algorithms, yet it is an essential part of the library. These descriptions are the analogue of the hypotheses of a theorem: the statement of a theorem has little value if the hypotheses are omitted.

This similarity with theorems makes the use of the generic algorithms intuitive, particularly in geostatistics. It allows a geostatistician to develop algorithms without being concerned about specific software (e.g. gridding) implementation; hence, it enables a widespread use and re-use of his/her code.

In the context of statistical/geostatistical algorithms, generic programming provides a more intuitive approach to coding than the object-oriented approach. The latter requires the library user to have a detailed understanding of the class hierarchies before being able to efficiently use the library.

It must be stressed that G5TL is a library of programming components, not a collection of software elements. Its aim is to provide tools and building blocks for quickly implementing new geostatistics algorithms.

An extension of this work would be to implement a set of geostatistical routines, in the style of GSLIB (Deutsch and Journel, 1992), based on G5TL.
Fig. 2. Sequential Gaussian simulation on a T-solid.

References


