Available online at www.sciencedirect.com
ScienceDirect
SoftwareX 5 (2016) 127–133
SoftwareX
www.elsevier.com/locate/softx

MPSLIB: A C++ class for sequential simulation of multiple-point statistical models

Thomas Mejer Hansen[a,*], Le Thanh Vu[b], Torben Bach[b]

[a] Niels Bohr Institute, University of Copenhagen, Denmark
[b] I-GIS, Aarhus, Denmark

Received 12 May 2016; received in revised form 7 July 2016; accepted 8 July 2016

Abstract

Geostatistical simulation methods allow simulation of spatial structures and patterns based on a choice of statistical model. In the last few decades multiple-point statistics (MPS) has been developed that allows inferring the statistical model from a training image. This allows for a simpler quantification of the statistical model, and simulation of more realistic (Earth) structures. A number of different algorithms for MPS based simulation have been proposed, each associated with a unique set of pros or cons. MPSLIB is a C++ class that provides a framework for implementing most of the currently proposed multiple-point simulation methods based on sequential simulation. A number of the most widely used methods are provided as an example. The single normal equation simulation (SNESIM) method is implemented using both a tree and a list structure. A new generalized ENESIM (GENESIM) algorithm is proposed that can act as (in one extreme) the ENESIM algorithm, and (in another extreme) similar to the direct sampling algorithm. MPSLIB aims to be easy to compile on most platforms (standard C++11 is the only requirement) and is released under the Open Source LGPLv3 License to encourage reuse and further development.

Keywords: Simulation; Geostatistics; Multiple-point statistics

Code metadata

Current code version	1.1
Permanent link to code/repository	https://github.com/ElsevierSoftwareX/SOFTX-D-16-00042
Legal Code License	LGPLv3
Code versioning system used	git
Software code languages, tools,	C++
Compilation requirements	C++11
Link to developer documentation/manual	https://github.com/ElsevierSoftwareX/SOFTX-D-16-00042/blob/master/README.md
Support email for questions	ltv@i-gis.dk, tmeha@nbi.ku.dk

1. Motivation and significance

Geostatistics is a type of statistics with a focus on describing geo-spatial (Earth) structures in a probabilistic framework through a probability distribution. A ‘geostatistical model’ refer to a selection of a probability distribution to reflect a specific Earth model. Geostatistical models are typically used to characterize subsurface reservoir models used for groundwater, hydrocarbon or heat storage, for both exploration, exploitation, and management. A geostatistical model describes infinitely many single Earth models, consistent with the chosen probability function. The variability of these Earth models reflect the uncertainty related to the choice of statistical model. Such uncertainty quantification is the key advantage of using

* Corresponding author.
E-mail address: tmeha@nbi.ku.dk (T.M. Hansen).

geostatistical models, as opposed to considering only one, in some sense, optimal model.

Geostatistical modeling is a two-step process. First a statistical model must be selected or described, typically through a probability distribution $f(\mathbf{m})$. Once a model has been established, actual realizations from the probability distribution is generated using 'simulation algorithms', which is the topic of this manuscript.

Traditionally, geostatistical simulation algorithms are based on Gaussian statistics, typically based on statistics only between pairs of model parameters, and hence referred to as 2-point statistics. These methods have been, and are still, widely used [1]. However, 2-point based statistical models are limited with respect to the spatial structures they can represent. With the introduction of multiple-point statistical (MPS) models, more geologically realistic spatial structures can be quantified [2,3]. This has led to the development of a number of simulation algorithms for multiple-point simulation, e.g. [2–5]. For MPS based statistical models, there is usually no closed form analytical expression of $f(\mathbf{m})$. Instead, a 'training image' or a 'sample model' is assumed available from which multiple-point statistical events can be inferred. The goal of MPS methods is to allow sampling from the unknown $f(\mathbf{m})$ such that realizations are consistent with the statistics that can be inferred from the training image. For an overview of MPS based simulation algorithms see [6].

Many of the proposed MPS algorithms are implemented in various forms. However, some of these codes are either not maintained [7,8], not available on multiple platforms [8], not available under an open source license [9–11], or does only implement one specific type of MPS algorithm [12].

Here a C++ library, MPSLIB, designed specifically for multiple-point simulation is presented, that is released under the GPLv3 license. MPSLIB implements the core functionalities needed to implement any multiple-point simulation algorithm, based on sequential simulation [13]. Note that this does not include methods based on pattern matching [14] and image-synthesis [15]. Implementations of the 'extended normal equations simulation' (ENESIM) [2] and the 'single normal equation simulation' (SNESIM) [3,4] algorithms are provided as examples. Further, a new algorithm GENESIM, based on the ENESIM algorithm, is proposed that can act similar to both the ENESIM and the direct sampling algorithms [5].

2. Software description

MPSLIB is written in C++ [16] using c++11 [17]. It consists of a C++ class that provides the framework for applying sequential simulation to sample from multiple-point statistical models. It also contains a number of algorithms implemented using MPSLIB.

At the core of the library is an implementation of sequential simulation, that can be briefly described as follows: Say a probability distribution describes the relation between M model parameters through $f(\mathbf{m}) = f(m_1, m_2, \ldots, m_M)$ that are typically ordered by some nodes on a grid. Then a realization of $f(\mathbf{m})$ can be generated using sequential simulation as follows:

1. Visit a random node, say m_1, and generate a realization of $f(m_1)$ as m_1^*.
2. Move to another node, say m_2, and generate a realization of $f(m_2|m_1^*)$ as m_2^*.
3. Move to another node, say m_3, and generate a realization of $f(m_3|m_1^*, m_2^*)$ as m_3^*.
4. ...
5. Move to the last node, m_M, and generate a realization of $f(m_M|m_1^*, m_2^*, \ldots, m_{M-1}^*)$ as m_M^*.

This will generate a realization of $\mathbf{m}^* = [m_1^*, m_2^*, \ldots, m_M^*,]$ from $f(\mathbf{m})$. See more details in e.g. [18].

The major challenge implementing the sequential simulation algorithm is to generate a realization from the conditional distribution at each iteration. If the conditional data are given by $\mathbf{m}_c^*$, then the conditional distribution can in general be given by

$$f(m_i|\mathbf{m}_c^*). \tag{1}$$

If uncertain co-located information about m_i is available as $f(m_{i\,soft})$ (often referred to as 'soft' data in geostatistical literature) then it is accounted for by drawing from

$$f(m_i|\mathbf{m}_c^*)\; f(m_{i\,soft}). \tag{2}$$

For MPS based models there will, in general, be no analytical form of $f(m_i|\mathbf{m}_c^*)$ available. Instead, the key idea utilized in most simulation algorithms is to infer information about the conditional distribution, by scanning the training image for the conditional data event, $\mathbf{m}_c^*$. There are (at least) two different approaches for sampling from the conditional distribution in Eq. (1). In one approach (ENESIM type algorithms), the training image is scanned for matching conditional data events at each iteration of the sequential simulation algorithm, from which the conditional distribution can be estimated [2], or a realization of the conditional distribution directly obtained [5]. In another approach, the training image is scanned prior to running the simulation, and the conditional statistics for a number of predefined data templates are stored in memory. The conditional distribution, Eq. (1), can then be retrieved from memory during simulation [3,4]. Usually, storing conditional statistics in memory requires the use of so-called multiple-grids [3] in order to allow simulation of structures with correlations over longer distances, while at the same time imposing manageable memory and CPU requirements. Using multiple-grids, simulation is performed starting in a coarse grid that is refined until simulation is performed on the finest, and requested, grid size. The use of multiple-grids poses a challenge to conditional simulation (when the value of some model parameters are known before simulation starts). One approach to handle this is to make use of data-relocation. When simulating on coarser grids, conditional data on finer grids are relocated to the closest node in the coarse grid. When conditional simulation has been performed on a coarser grid, re-located data on the coarse grid are removed, and simulation is then performed on a finer grid. For details on multiple-grids and the use of data-relocation see [19,3,20].

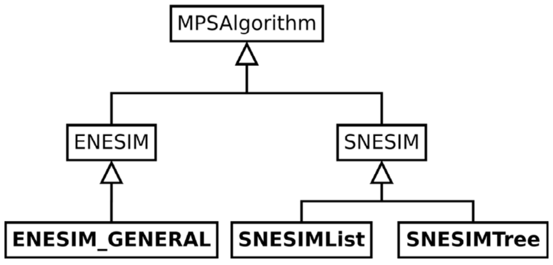


Fig. 1. MPS general architecture.

2.1. Software architecture

MPS is a namespace that contains different classes. The main class is the MPSalgorithm class, which implements the sequential simulation algorithm, methods for reading and writing 3D gridded data, methods for reading known data values (known as hard and soft data), and methods for establishing a data neighborhood, and controlling the simulation path. In addition, MPSAlgorithm allows multiple grids, including handling of conditional data. Specifically, grid re-location of hard data, as proposed in [3], is implemented.

Also, two abstract member functions are defined, but not implemented: MPSAlgorithm::readConfig and MPS-Algorithm::simulate

That is, in order to use the MPS class, a new C++ class that inherits the MPSAlgorithm class needs to be defined, that implements (at least) the two member functions. MPSAlgorithm::readConfig should set all the parameters needed to run the simulation, for example by reading from a parameter file. MPSAlgorithm::simulate should implement a method that allows generating a realization from Eq. (1). As discussed above, the main difference between proposed multiple-point simulation algorithms is related to how this function is implemented, and can be divided into two families of algorithms, those that are similar to ENESIM and those that are similar to SNESIM. Therefore, two general classes, MPSEnesim and MPSSnesim, have been implemented to handle operations specific to ENESIM and SNESIM type simulation.

ENESIM. The ENESIM subclass extends the main MPSAlgorithm class with methods and functions specifically designed to use in case the training image is scanned at each iteration. This includes methods for scanning the whole training image in order to estimate the conditional distribution, Eq. (1), from which a realization can be drawn. It also includes methods for scanning only parts of the training image, and optionally only scan the training image until a maximum number of conditional data events has been found.

SNESIM. The SNESIM subclass extends the main MPS class with methods and functions specifically designed in case the statistics from the training images is scanned prior to running the simulation algorithm and stored in memory.

The SNESIM and ENESIM subclasses form the base of the proposed C++ framework. A key idea is that it should be possible to implement any MPS based sequential simulation algorithm using the C++ Class. Fig. 1 illustrates the architecture of MPSLIB.

2.2. MPS simulation algorithms

To demonstrate the versatility of MPSLIB a number of the most well-known multiple-point simulation algorithms, as well as a new variant, based on sequential simulation has been implemented.

mps_genesim. The mps_genesim algorithm is a generalized implementation of the MPS algorithm named ENESIM proposed by [2].

In the simplest form of the ENESIM algorithm, the whole training image is scanned to establish the conditional distribution in Eq. (1) at each iteration. This is relatively easy to implement, and is void of problems related to the use of multiple grids. However, it is also computationally extremely demanding, to the point where it is not practical to use. The reason is due to the fact the full training image is scanned at each iteration, in order to obtain a conditional distribution from which a realization can be drawn. In order to alleviate this we suggest to scan the training image only for a maximum number N_{max} of conditional data events, which means the conditional distribution will only be based on (at the most) N_{max} counts.

MPSAlgorithm::simulate is implemented by scanning the training image for a maximum of N_{max} replicates, from a random starting point. This provides an approximation to the full conditional distribution, from which a realization is generated.

At one extreme, $N_{max} = \infty$, this algorithm will produce the same results as the classical ENESIM algorithm.

At another extreme, $N_{max} = 1$, this algorithm will provide essentially the same results as the direct sampling algorithm, with the specific difference that using mps_genesim the conditional distribution is in fact estimated (even if based on only one count), from which a realization is generated. Using direct sampling the conditional distribution is never obtained, and instead a realization from Eq. (1) is taken directly from the training image as the first matching conditional data event[5]. In this case, when the conditional distribution is based on only one observed conditional event, conditioning to uncertain data, as in Eq. (2), cannot be done.

For $N_{max} > 1$ and $N_{max} < \infty$ mps_genesim leads to an algorithm that can approximately account for uncertain data using (2), while at the same avoiding the high CPU requirements associated to scanning the full training image.

mps_snesim_tree. The main contribution of SNESIM is that conditional patterns are scanned from the training image prior to simulation, and stored in a search tree [3]. Therefore a member function is implemented that allows scanning a training image for a set of predefined data templates. The corresponding conditional statistics are stored in a binary search tree. MPSAlgorithm::simulate is implemented such that the conditional distribution in Eq. (1) is obtained by searching through the binary search tree. Then a realization is generated from Eq. (1).

mps_snesim_list. mps_snesim_list is very similar to mps_snesim_snesim. The main difference is that mps_snesim_list stores conditional statistics using a list rather than a search tree.

This leads to less memory requirements, but slower sequential search for the conditional distribution, compared to using a binary search tree [4].

The two SNESIM based algorithms will though not produce exactly the same result as they differ slightly in how a missing data event is handled.

2.3. Future development

The algorithms described above implements the core ideas of the SNESIM, ENESIM, and GENESIM algorithms. Many variations of the methods are available, and many types of post-processing have been presented, some of which may be included in MPSLIB in the future [8,21]. A natural feature to implement next is to allow simulation of non-stationary fields, by using multiple training images and/or scaling and rotation of on training image, which has not yet been considered [8,22]. The use of parallel computing and GPUs will also be a natural development [23,24].

3. Examples

As an example, based on the training image in Fig. 2(a), the implemented algorithms will be used to generate realizations in a 2D 80 × 50 pixel grid, using the hard and soft data shown in Fig. 2(b).

The implemented algorithms are based on the same core functionality, and therefore the parameter files needed to run the algorithms are very similar. The following text file is part of the parameter file for all simulations algorithms:

```
Number of realizations # 3
Random Seed (0 ‘random‘ seed) # 1
Simulation grid size X # 80
Simulation grid size Y # 50
Simulation grid size Z # 1
Simulation grid world/origin X # 0
Simulation grid world/origin Y # 0
Simulation grid world/origin Z # 0
Simulation grid grid cell size X # 1
Simulation grid grid cell size Y # 1
Simulation grid grid cell size Z # 1
Training image file
  (spaces not allowed) # ti.dat
Output folder (spaces in name not allowed) # .
Shuffle Simulation Grid path
  (1 : random, 0 : sequential) # 1
Shuffle Training Image path
  (1 : random, 0 : sequential) # 1
HardData filename  (same size as the simulation
  grid) # conditional.dat
HardData seach radius (world units) # 1
Softdata categories (separated by ;) # 0;1
Soft datafilenames (separated by ; only need
  (number_categories - 1) grids) # soft.dat
Number of threads (minimum 1, maximum 8 -
  depend on your CPU) # 1
Debug mode (2: write to file, 1: show preview,
  0: show counters, -1: no ) # -1
```

The parameter file above defines a 2D regular grid (80 by 50 pixels) where each pixel has a size of 1 × 1 and the coordinates of the first pixel are (0, 0). The training image must be given in an ASCII formatted GSLIB file (a type of simplified Geo-EAS formatted file, see [1]) called ‘ti.dat’, where the first line contains the dimension of the training image. For example, the first line for a 100 × 100 × 10 training image must be ‘100 100 10’. One can optionally make use of hard data (data known with no uncertainty), or soft data (data that reflect uncertain observations), both in GSLIB format. The number of threads is currently not used, but kept to allow the use of multi threading in the future. Finally the debug mode defines the amount of information written to disk and screen during simulation.

3.1. SNESIM

Both `mps_snesim_tree` and `mps_snesim_list` make use of the same additional information in the parameter file:

```
Number of multiple grids # 4
Search template size X # 7
Search template size Y # 7
Search template size Z # 1
Maximum number conditional data (-1: all) # 49
Min Node count (-1: ignore)# 5
```

The first line sets the number of multiple grids used. To disable the use of multiple-grids, “0” should be chosen. The next three lines indicate the search template size, which is here set as 7 × 7 × 1. This means that any conditional data event within a 7 × 7 grid size is scanned, and its associated conditional statistics stored, prior to running the simulation algorithm. The next line sets the maximum number of conditional data within the template to consider. If set to zero all conditional data within the template will be used.

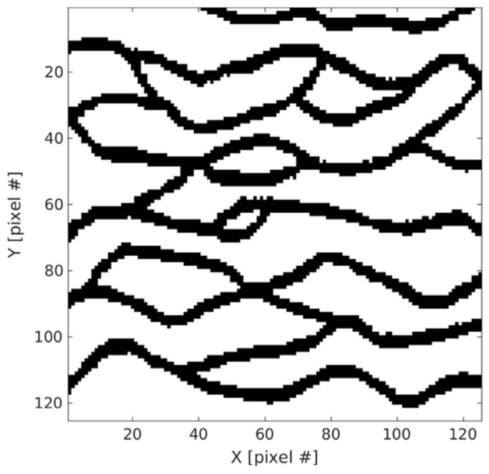
The last line sets the minimum number of replicates needed in order to use an obtained conditional distribution. If the number of conditional data is below this number then one conditional data is discarded, until enough replicates are found.

3.2. GENESIM

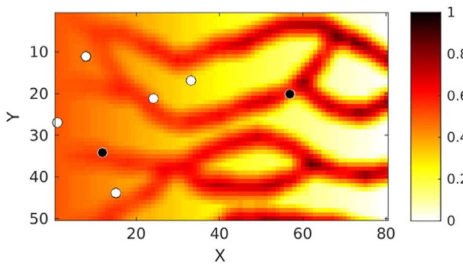
The `mps_genesim` algorithm needs the following additional parameters defined:

```
Max number of conditional point, N_cond # 49
Maximum number of counts for conditional pdf,
  N_max # 1
Max number of iterations, N_max_ite # 10000
```

The first line defines the maximum number of conditional points needed during simulation. This is related to the template size for the SNESIM type algorithms. However, as no multiple-grids are used, conditioning can be effective over ranges spanning the whole simulation grid. The next two lines define properties specifically related to the GENESIM algorithm. Line 2 defines the maximum number of points, N_{max}, used to set up the conditional distribution Eq. (1). However, even if not

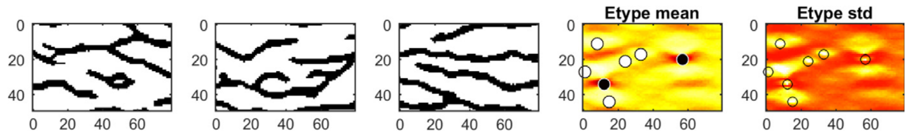


(a) Training image.

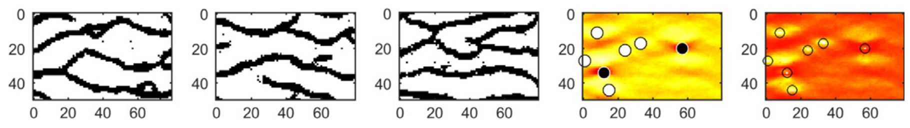


(b) Simulation grid hard and soft data, $P(m_i) = 1$.

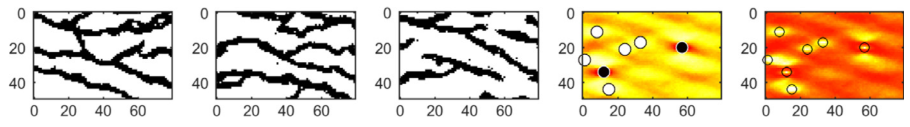
Fig. 2. (a) Training image, (b) Simulation grid with 6 hard (circles) and soft data at all grid nodes. Colorbar indicates the local soft probability. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



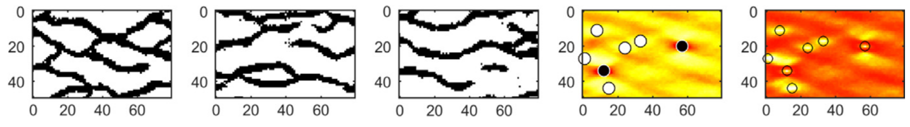
(a) mps_snesim_tree.



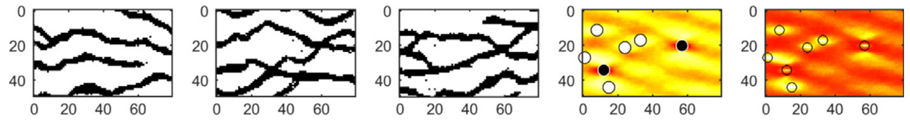
(b) mps_snesim_list.



(c) mps_genesim ($N_{max} = \infty$).



(d) mps_genesim ($N_{max} = 10$).

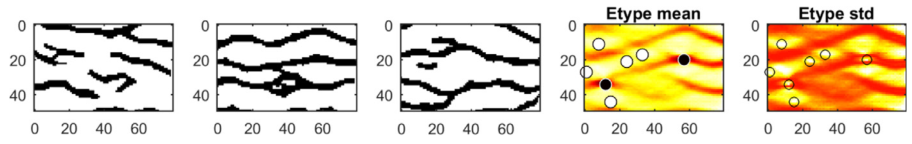


(e) mps_genesim ($N_{max} = 1$).

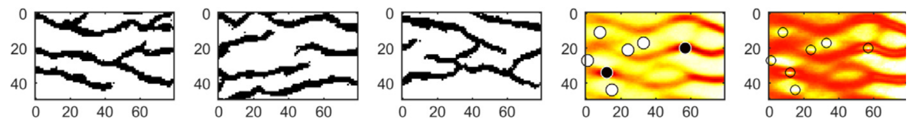
Fig. 3. Simulation conditional to 6 hard data, using (a) mps_snesim_tree, (b) mps_snesim_list, (c) mps_genesim ($N_{max} = \infty$), (d) mps_genesim ($N_{max} = 10$), and (e) mps_genesim ($N_{max} = 1$). Columns 1–3: 3 independent realizations. Column 4: Etype mean (colorbar as in Fig. 2). Column 5: Etype standard deviation (yellow: low, red: high). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

enough conditional points have been obtained after $N_{max\_ite} = 10000$ iterations of scanning for a match (as defined in the last line), scanning is stopped and the conditional distribution used as is.

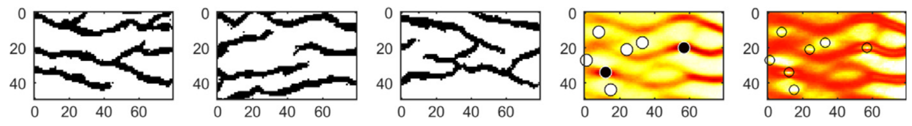
In case $N_{max} = \infty$ and $N_{max\_ite} = \infty$, then `mps_genesim` will behave just as the ENESIM algorithm. $N_{max} = 1$ and $N_{max\_ite} = \infty$ will lead to an algorithm similar direct sampling, in the sense that if needed, the whole training image will



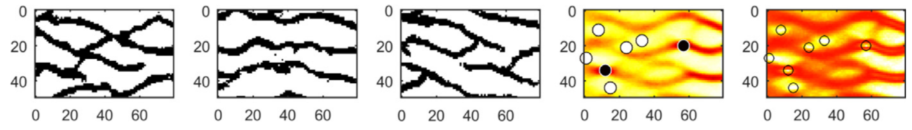
(a) mps_snesim_tree.



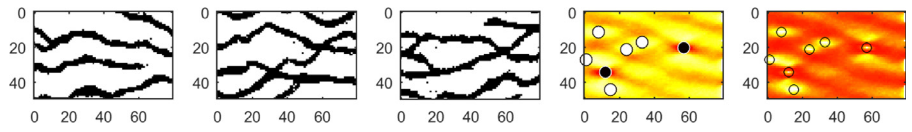
(b) mps_snesim_list.



(c) mps_genesim ($N_{max} = \infty$).



(d) mps_genesim ($N_{max} = 10$).



(e) mps_genesim ($N_{max} = 1$).

Fig. 4. Simulation conditional to 6 hard data and soft data, using (a) mps_snesim_tree, (b) mps_snesim_list, (c) mps_genesim ($N_{max} = \infty$), (d) mps_genesim ($N_{max} = 10$), and (e) mps_genesim ($N_{max} = 1$). Columns 1–3: 3 independent realizations. Column 4: Etype mean (colorbar as in Fig. 2). Column 5: Etype standard deviation (yellow: low, red: high). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

be scanned for a replicate. The original implementation of direct sampling does not allow for conditioning to 'soft' uncertain data in a manner similar to the ENESIM and SNESIM algorithm. However, the GENESIM algorithm, using a finite N_{max_ite} has the potential to decrease the simulation time significantly compared to a traditional ENESIM implementation, while at the same time allowing for conditioning to uncertain data through Eq. (2).

Fig. 3, shows three independent realizations conditional to only the hard data, generated using `mps_snesim_tree`, `mps_snesim_list`, and `mps_snesim_genesim` with $N_{max} = \infty$, $N_{max} = 10$, and $N_{max} = 1$. Fig. 4, shows the same results as Fig. 3 but in case conditioning to both hard and soft data. Note how soft data is not taken into account, as discussed previously, when using GENESIM in style of direct sampling with $N_{max} = 1$, Fig. 4(e). On the other hand, using $N_{max} = 10$, the GENESIM algorithm can take soft data into account, Fig. 4(d).

4. Impact

MPSLIB has been developed to be useful to both beginners and experts using multiple-point geostatistics. It should be useful for both scientific research and commercial application. A goal has been to provide a foundation that allows developing, testing, and comparing the many different approaches for utilizing multiple-point simulation, which are all largely built on the same basic idea.

Perhaps the most widely known and used software for geostatistical estimation and simulation is GSLIB [1]. GSLIB, written in Fortran, provides both a set of building blocks, and numerous examples of how to use these building blocks to implement useful algorithms, for 2-point statistical modeling. The history of GSLIB has shown that availability of free and open software can be a main driver in the development of a research field, benefiting both basic research and commercial application. The design of MPSLIB is chosen with GSLIB in mind.

MPSLIB should be flexible, easy to extend, easy to compile, and applicable for both scientific and commercial purposes. MPSLIB should alleviate the possibility to test out new ideas, without the need to re-implement every aspect of sequential simulation. Currently no such computer code is available, that uses both a permissive open source license, and is maintained.

The development of the generalized ENESIM algorithm is an example on how the existing code can be extended to produce a new type of multiple-point statistical algorithm with some novel features. We hope this example, and the base code, will encourage other developers to extend the code in a similar manner.

The software was developed in a collaboration between researchers from a public institution and a commercial company. The company will use the code as the back end for a user orientated geological model building experience. The researchers will use the code for scientific research related to multiple-point statistical modeling in the future.

5. Conclusions

MPSLIB is a C++ library for multiple-point based statistical models based on sequential simulation. It is intended to provide the building blocks that allow implementation of any multiple-point simulation method based on sequential simulation. As examples, variants of ENESIM and SNESIM type multiple-point algorithms have been implemented and a new algorithm proposed. MPSLIB relies only on standard C++11 and should be easy to compile, modify and extend, and is released under the LGPLv3 license.

Acknowledgments

The code needed to generate Figs. 2–4 is part of the source code, as well as a Matlab interface that allow runn ing the developed algorithms in parallel using the Matlab Parallel Computing toolbox. The development of MPSLIB has been developed as part of the ERGO project, funded by a grant from the Danish High Technology Foundation (grant number 113-2013-1). We thank Giulio Vignoli, GEUS, Denmark, for testing and feedback.

References

[1] Deutsch CV, Journel AG. GSLIB, Geostatistical software library and user's guide. second ed. Applied geostatistics, Oxford University Press; 1998.
[2] Guardiano F, Srivastava R. Multivariate geostatistics: beyond bivariate moments. Geostatistics-Troia 1993;1:133–44.
[3] Strebelle S. Conditional simulation of complex geological structures using multiple-point statistics. Math Geol 2002;34(1):1–20.
[4] Straubhaar J, Renard P, Mariethoz G, Froidevaux R, Besson O. An improved parallel multiple-point algorithm using a list approach. Math Geosci 2011;43(3):305–28.
[5] Mariethoz G, Renard P, Straubhaar J. The Direct Sampling method to perform multiple-point geostatistical simulations. Water Resour Res 2010;46(11).
[6] Mariethoz G, Caers J. Multiple-point geostatistics: Stochastic modeling with training images. John Wiley & Sons; 2014.
[7] Remy N, Shtuka A, Levy B, Caers J. GS TL: the geostatistical template library in C++. Comput Geosci 2002;28(8):971–9.
[8] Remy N, Boucher A, Wu J. Applied geostatistics with SGeMS: a user's guide. Cambridge University Press; 2009.
[9] Ar2Tech, Ar2Gems (2015). http://www.ar2tech.com/products.
[10] Consult E. Impala (2015). URL http://www.ephesia-consult.com/work/impala/.
[11] Consult E. DeeSee (2015). URL http://www.ephesia-consult.com/work/deesse/.
[12] Strebelle S. SNESIM, URL https://github.com/SCRFpublic/snesim-standalone (2015).
[13] Alabert F, et al. Non-Gaussian data expansion in the earth sciences. Terra Nova 1989;1(2):123–34.
[14] Wu J, Boucher A, Zhang T. A SGeMS code for pattern simulation of continuous and categorical variables: FILTERSIM. Comput Geosci 2008;34(12):1863–76.
[15] Mariethoz G, Lefebvre S. Bridges between multiple-point geostatistics and texture synthesis: Review and guidelines for future research. Comput Geosci 2014;66:66–80.
[16] Stroustrup B. The C++ programming language. Pearson Education India; 1986.
[17] ISO, Information technology – Programming languages – C++, ISO ISO/IEC 14882:2011, International Organization for Standardization, Geneva, Switzerland (2011).
[18] Gomez-Hernandez J, Journel A. Joint sequential simulation of multi-Gaussian fields. Geostatistics Troia 1993;92:85–94.
[19] Hernandez JG. A stochastic approach to the simulation of block conductivity fields conditioned upon data measured at a smaller scale [Ph.D. thesis], Stanford University; 1993.
[20] Straubhaar J, Malinverni D. Addressing conditioning data in multiple-point statistics simulation algorithms based on a multiple grid approach. Math Geosci 2014;46(2):187–204.
[21] Zhang T, Pedersen SI, Knudby C, McCormick D. Memory-efficient categorical multi-point statistics algorithms based on compact search trees. Math Geosci 2012;44(7):863–79.
[22] Mariethoz G, Kelly BF. Modeling complex geological structures with elementary training images and transform-invariant distances. Water Resour Res 2011;47(7).
[23] Mariethoz G. A general parallelization strategy for random path based geostatistical simulation methods. Comput Geosci 2010;36(7):953–8.
[24] Tahmasebi P, Sahimi M, Mariethoz G, Hezarkhani A. Accelerating geostatistical simulations using graphics processing units (GPU). Comput Geosci 2012;46:51–9.