User's Guide

January 2019

Software patented by the University of Neuchâtel

Julien Straubhaar – julien.straubhaar@unine.ch

UNIVERSITÉ DE NEUCHÂTEL

Stochastic Hydrogeology and Geostatistics Laboratory, Centre for Hydrogeology and Geothermics (CHYN), University of Neuchâtel, 11 Rue Emile Argand, CH-2000 Neuchâtel, Switzerland.
http://www.unine.ch/chyn

Contents

List of Figures

1 Introduction

DeeSse is a parallel software for multiple point statistics (MPS) simulations. The method is based on the direct sampling of the training image (TI) [Mariethoz et al., 2010] and is patented [Mariethoz et al., 2014]. The code is written in C, and the parallel version used OpenMP and is devoted for shared memory machines.

MPS allows to generate random fields reproducing the structures present in a conceptual model: the TI. This allows the user to implicitly give spatial relationships – within (and between) variable(s) – to be reproduced by DeeSse. In particular, multi-variate simulations can be used to deal with non-stationary TIs.

1.1 DeeSse's main features

- Categorical and continuous variables.
- Simulation for single variable or multiple joint variables.
- Conditioning to punctual data (hard data).
- Conditioning to block data (mean target values on subsets of pixels) [Straubhaar et al., 2016].
- Conditioning to connectivity data.
- Dealing with non-stationarity given by homotheties and rotations.
- Dealing with global and local probability constraints [Mariethoz et al., 2015].
- Dealing with incomplete TIs and training data set.
- Using multiple TIs.
- Multiple scale simulation based on Gaussian pyramids.

Meerschman et al. [2013] proposed a practical guide dealing with basic simulation cases.

Remark 1. *All grids (simulation grids, TI grids, etc.) are defined as 3D regular grids. This allows to perform spatial simulations in 1D, 2D and 3D.*

Part I

Using DEESSE as a console application

2 Grid, image, point set and block data set objects

Grid, image, point set and block data set are objects used in DEESSE. They are defined in the following. The file format used in DEESSE for image and point set are also given.

2.1 Grid object

A grid is a 3D regular box containing nodes (pixels). A grid is defined by the following parameters:

- Nx, Ny, Nz, the number of nodes in each direction;
- Sx, Sy, Sz, the spacing (unit for one node) in each direction;
- Ox, Oy, Oz, the origin of the grid: coordinates of the bottom low left corner of the node at the bottom low left of the grid (bottom, low and left are relative to z, y and x axis direction here).

The grid defined by the parameters above contains $Ng = Nx \cdot Ny \cdot Nz$ nodes; they are numbered from 0 to $Ng-1$, with x index varying first, then y index and finally z index. Thus, the node located at position of indices (ix, iy, iz) (in nodes) in the grid, has the (global) index $ix + Nx \cdot (iy + Ny \cdot iz)$, for $0 \leqslant ix \leqslant Nx-1$, $0 \leqslant iy \leqslant Ny-1$, $0 \leqslant iz \leqslant Nz-1$. The origin localizes in 3D space the bottom low left corner of the 0-th node of the grid. The top up right corner of the $(Ng-1)$-th node of the grid has the spatial coordinates $(Ox + Nx \cdot Sx, Oy + Ny \cdot Sy, Oz + Nz \cdot Sz)$.

Remark 2. *A grid is always defined in 3D space; however, setting Nx and/or Ny and/or Nz to one allows to consider a grid in 1D or 2D space.*

2.2 Image object

An image is a grid with one or more variable(s) attached to each node. An image is given by:

- a grid;
- a number $nvar$ of variable(s);
- the name of each variable;
- $nvar$ vectors $Z1, \ldots, Znvar$ of size Ng (number of nodes in the grid) containing the variable values at each node of the grid: $Zi(j)$ is the value of the i-th variable at the j-th node of the grid, $i = 1, \ldots, nvar$, $j = 0, \ldots, Ng-1$.

An image is given in a file in the following format (based on GSLIB, Geostatistical Software LIBrary, see http://www.gslib.com/).

Image file format

```
Nx Ny Nz [Sx Sy Sz [Ox Oy Oz]]
nvar
name_of_variable_1
...
name_of_variable_nvar
Z1(0)    ... Znvar(0)
...
Z1(Ng-1) ... Znvar(Ng-1)
```

Remark 3. *The parameters Sx, Sy, Sz and Ox, Oy, Oz are optional, the default value are $Sx = Sy = Sz = 1.0$ and $Ox = Oy = Oz = 0.0$.*

2.3 Point set object

A point set is a list of points spatially localized in 3D space with three coordinates, with one or more variable(s) attached to each of these points. A point set is givent by:

- a number *npoint* of point(s);
- a number *nvar* of variable(s);
- the coordinates x, y, z of each point;
- the name of each variable;
- *nvar* vectors $Z1, \ldots, Znvar$ of size *npoint* containing the variable values at each point: $Zi(j)$ is the value of the i-th variable at the j-th point, $i = 1, \ldots, nvar$, $j = 1, \ldots, npoint$.

A point set is given in a file in the following format (based on GSLIB, Geostatistical Software LIBrary, see http://www.gslib.com/).

Point set file format

```
npoint
nvar+3
name_for_x_coordinate
name_for_y_coordinate
name_for_z_coordinate
name_of_variable_1
...
name_of_variable_nvar
x(1)      y(1)      z(1)      Z1(1)      ... Znvar(1)
...
x(npoint) y(npoint) z(npoint) Z1(npoint) ... Znvar(npoint)
```

Remark 4. *One number $(nvar + 3)$ is written on the second line; the name for x, y and z coordinates must begin by x or X, y or Y and z or Z respectively; $x(j)$, $y(j)$ and $z(j)$ are the values of the x, y and z coordinates of the j-point.*

2.4 Block data set object

A block data set is a list of block data where each block data is given by a set of nodes spatially localized in an underlying grid (indeed the simulation grid), a mean value attached to the set of nodes, a tolerance value (around the mean), and a minimal and maximal "activation proportion" indicating when the block data constraint is activated (during the simulation, the proportion of informed nodes plus the node currently simulated in the block is compared to these bounds and the block data constraint for this block is activated or not). A block data set is given by:

- a number $nblockData$ of block data;
- for each block data $1 \leqslant i \leqslant nblockData$:
  - a number $nnode(i)$ of nodes;
  - a mean value $value(i)$;
  - a tolerance value $tolerance(i)$;
  - a minimal "activation proportion" value $propMin(i)$;
  - a maximal "activation proportion" value $propMax(i)$;
  - three vectors $ix(i)$, $iy(i)$, $iz(i)$ of size $nnode(i)$ containing the indices in the underlying grid of each node constituting the i-th block (set of nodes).

A block data set is given in a file in the following format.

Block data set file format

```
nblockData
nnode(1) value(1) tolerance(1) propMin(1) propMax(1)
ix(1)(1)        iy(1)(1)        iz(1)(1)
...
ix(1)(nnode(1)) iy(1)(nnode(1)) iz(1)(nnode(1))
...
nnode(nblockData) value(nblockData) tolerance(nblockData) propMin(nblockData) propMax(nblockData)
ix(nblockData)(1)                 iy(nblockData)(1)                 iz(nblockData)(1)
...
ix(nblockData)(nnode(nblockData)) iy(nblockData)(nnode(nblockData)) iz(nblockData)(nnode(nblockData))
```

Remark 5. *The entries $ix(i)(j)$, $iy(i)(j)$, $iz(i)(j)$ are node indices in a grid along x, y, z direction respectively. Node indices in a grid start from* 0 *(Sect. 2.1).*

2.5 Missing value

A reserved constant, `MPDS_MISSING_VALUE`, are used in DeeSse for encoding a missing (or undefined) value. `MPDS_MISSING_VALUE` is set to $-9\,999\,999$. This allows to define a variable on a part of an image or a point set.

3 Input parameters file for DeeSse

DeeSse allows the user to write comments in C-style format almost everywhere in the input parameters file ('//' introduces a comment for the rest of the current line, or a comment can be written (on several lines) between '/*' and '*/'). Hence, despite the numerous parameters to specify, the input file can be written in an comprehensible manner. Please refer to the Sect. A, where such a file is given (simple simulation case).

Note that the output file(s) are specified in the input parameters file, see below.

About the input parameters file

- The file ends with the character string `END`.
- The output flag for the simulation variable(s) indicates if the corresponding variable will be considered for output or not.
- The output settings for simulation are determined by one of the following key words:
  - `OUTPUT_SIM_NO_FILE`: no file will be produced in output.
  - `OUTPUT_SIM_ALL_IN_ONE_FILE`: one file in image file format will be produced in output, all variables for all realizations will be written in columns; for N+1 variables and M+1 realizations:

    ```
    var0_real0 ... varN_real0 ... var0_realM ... varN_realM
    ```

  - `OUTPUT_SIM_ONE_FILE_PER_VARIABLE`: one file in image file format per simulation variable will be produced in output, in each file all realizations will be written in columns; for variable I and M+1 realizations:

    ```
    varI_real0 ... varI_realM
    ```

  - `OUTPUT_SIM_ONE_FILE_PER_REALIZATION`: one file in image file format per realization will be produced in output, in each file all variables will be written in columns; for N+1 variables and realization J:

    ```
    var0_realJ ... varN_realJ
    ```

- Additional maps (image on the SG) can be retrieved in output for each realization:
  - the simulation path map: index of each node in the simulation path before possible post-processing (negative index for conditioning nodes or added conditioning nodes for connectivity),
  - the error map before possible post-processing: error for each simulated node, corresponding to the sum of relative error(s) compared to the threshold value(s) for the best retained candidate from the TI.

  Both images follow the format `OUTPUT_SIM_ONE_FILE_PER_REALIZATION` above.

- A report file can be written, it contains some informations about the simulation.
- A TI is given in a file in image file format. The number of variable(s) must be equal to the number of simulation variable(s), and the i-th variable corresponds to the i-th simulation variable, whatever its name. If more than one TI is used, one file for PDF is given in image file format; its grid must have the same parameters as those of the SG and it must contain as many variables as number of TIs, the i-th variable is the probability to choose the i-th TI.
- A TI can be incomplete, *i.e.* some variable(s) at some nodes can be uninformed (undefined), using the constant `MPDS_MISSING_VALUE` ($-9\,999\,999$). Moreover, a TI can be replaced by the SG itself, which means that the SG is scanned instead the TI during the simulation; for this, the TI file name is replaced by the string `_DATA_` in the input parameters file.
- Conditioning data (hard data) can be given in image(s) or in point set(s). The grid of an image of conditioning data must have the same parameters as those of the SG. The number of variable(s) given in image(s) or point set(s) of conditioning data can be different from the number of simulation variable(s); however the variable name(s) describing the conditioning data must correspond to the name(s) of the wanted simulation variable(s) or to the name(s) used for connectivity constraint(s). Note that the conditioning data given in image(s) are first integrated in the SG and then those given in point set(s) (in the given order), by possibly erasing the previous values. Moreover, if a mask is used, the conditioning data fallen in the unsimulated part of the SG are ignored (removed).
- The grid of the possible mask image file must have the same parameters as those of the SG. For multivariate simulation, the mask is applied to every variable.
- The grid of the possible image file(s) for homothety and/or rotation must have the same parameters as those of the SG.
- If local probability constraints are used for a simulated variable, one file for local PDF is given in image file format; its grid must have the same parameters as those of the SG and it must contain as many variables as number of specified classes of values, the i-th variable is the local probability for the i-th class. Moreover, some nodes can be uninformed; for such a node, the value for all the variables is set to `MPDS_MISSING_VALUE` ($-9\,999\,999$).
- Using pyramid is incompatible with the use of probability constraints (global or local) and with block data conditioning, and limit the use of connectivity constraints to the modes dealing with connectivity data before starting the simulation.

Remark 6. *Constant values for* DeeSse *are defined in the header file MPDSConst.h.*

4 Launching DeeSse

For launching the serial version of DeeSse as console application, you have to type in the terminal

```
DeeSse <params.in>
```

where `DeeSse` is the executable for the serial version of DeeSse and `<params.in>` the input parameters file (see previous section).

For using the parallel version of DeeSse (based on OpenMP), you have to type

```
DeeSseOMP <nthreads> <params.in>
```

where `DeeSseOMP` is the executable for the parallel version of DeeSse, `<ntreads>` the number of threads to be used and `<params.in>` the input parameters file. Note that specifying `nthreads` to $n \leqslant 0$ implies the use of the maximal number of threads available on the machine (returned by the OpenMP function *omp_get_max_threads()*) except n, but at least 1.

Remark 7. *Reproducibility of results is not guaranteed when using more than one thread and the following options: simulation grid as TI, global probability constraint, and block data conditioning.*

5 Recommendations to users

Here some useful recommendations about input parameters are given.

- Use a small simulation grid in a first step, to see if there is already some trouble.
- The maximal number(s) of neighbors, the distance threshod(s), and the maximal scanned fraction(s) of the TI(s) are key parameters; they can have a great impact on the CPU time required by the simulation. Hence, start with parameters not too demanding, and then adapt them. Note that the maximal scanned fraction of a TI should be chosen by taking into account the size of the TI.
- Search neighborhoods should be large enough (in number of nodes) for catching the structures within the TI(s).
- In presence of hard data, it can be useful to limit the radius of search neighborhoods to avoid to simulate the first nodes based on a similar wide pattern (made of hard data far from each other) for every realization.

Part II
Examples

6 About the examples

In the next sections, examples are presented to illustrate most features of DeeSse. In each example, the input parameters are given. If nothing is mentioned, the following parameters are used: SG spacing and origin: $Sx = Sy = Sz = 1.0$, $Ox = Oy = Oz = 0.0$; unconditional simulation; no homothety, no rotation; distance type 0 (mismatching nodes) for categorical variable and 1 (L_1) for continuous variable; relative distance mode off; search neighborhood: radii: '`SEARCHNEIGHBORHOOD_RADIUS_LARGE_DEFAULT`' (r_x, r_y, r_z automatically computed), anisotropy ratios: '`SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_ONE`' ($a_x = a_y = a_z = 1.0$, no anisotropy), angles: '`SEARCHNEIGHBORHOOD_ROTATION_OFF`' ($\alpha = \beta = \gamma = 0.0$, no rotation), power: $p = 0.0$; random simulation path; no probability constraint; no block data conditioning; no connectivity data conditioning; no use of pyramids; tolerance (on relative error): $tol = 0.0$; one post-processing path with the default parameters.

For each example, the dimensions of TI and SG, and the three keys parameters – N (maximal number of neighbors), t (distance threshold), and f (maximal scanned fraction of the TI) – are given. Furthermore, the real elapsed time is given, running DeeSse on a laptop CPU specification: *Intel(R) Core(TM) i7-4810MQ CPU @ 2.80GHz*.

Input files for every example are available on request.

7 Simple simulations

Example 1. Simple simulation – categorical with 3 categories. TI dimensions: 300×250, SG dimensions: 240×200, $N = 24$, $t = 0.05$, $f = 0.25$. See Fig. 1. *Real time using 4 CPUs:* ~ 0.9 s.

Example 2. Simple simulation – continuous. Same parameters as in example 1 except the distance type. See Fig. 2. *Real time using 4 CPUs:* ~ 8.6 s.

Example 3. Simple simulation 3D – categorical with 3 categories. TI dimensions: $100 \times 90 \times 80$, SG dimensions: $60 \times 60 \times 60$, $N = 36$, $t = 0.05$, $f = 0.08$. See Fig. 3. *Real time using 4 CPUs:* ~ 37.0 s.

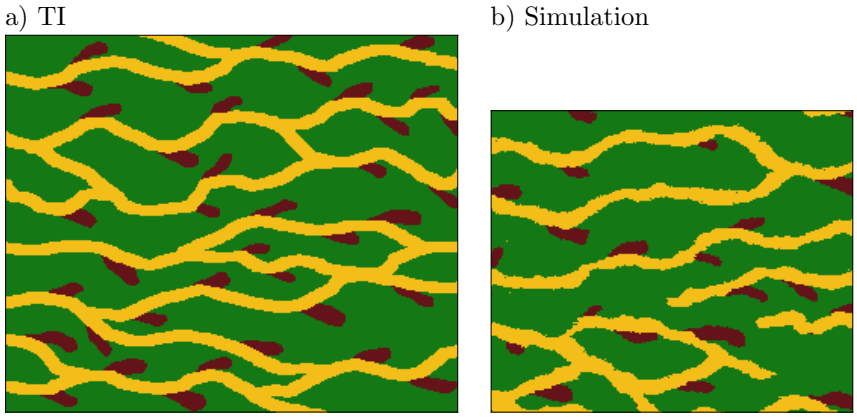


Fig. 1 Example 1: simple simulation – categorical. a) TI of size 300×250, b) one realization of size 240×200.

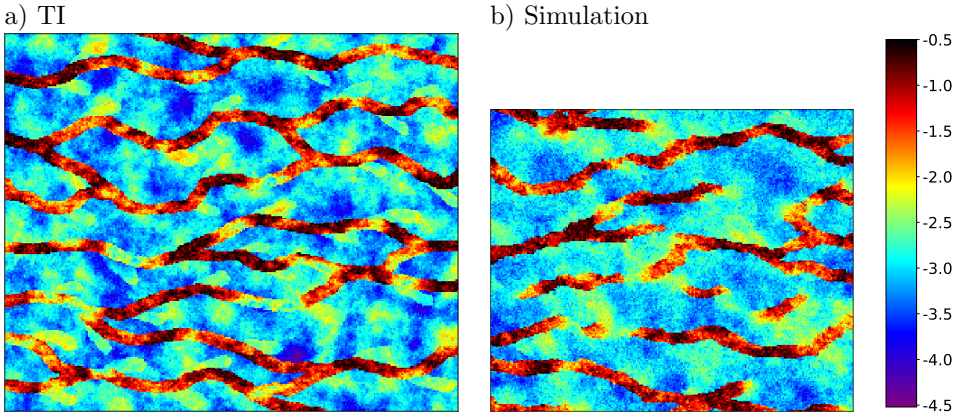


Fig. 2 Example 2: simple simulation – continuous. a) TI of size 300×250, b) one realization of size 240×200. (Same color bar for both images.)

a) TI

b) Simulation

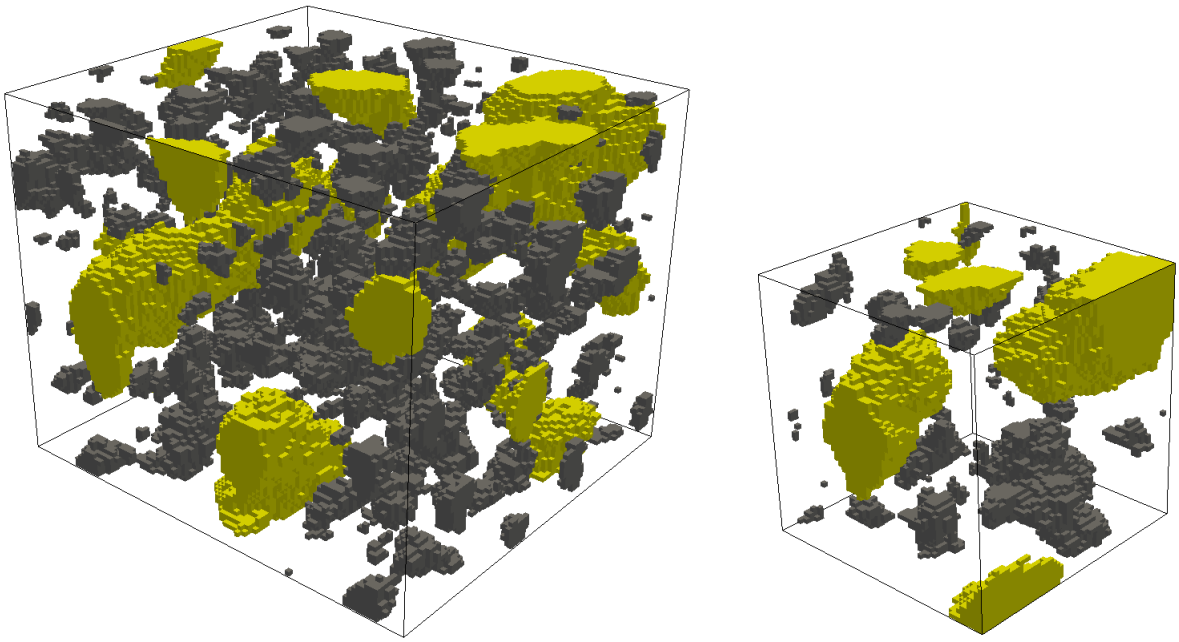


Fig. 3 Example 3: simple simulation 3D – categorical. a) TI of size $100 \times 90 \times 80$, b) one realization of size $60 \times 60 \times 60$.

8 Simulations with hard data

Simulations integrating hard data are illustrated for a continuous variable. The relative distance mode is not activated (standard) in Example 4.1 and activated in Example 4.2. Note that the relative mode allows to adapt the range of the simulated values to the conditioning data and to implicitly integrate a trend (Fig. 4).

Example 4. Continuous simulation with hard data. TI dimensions: 200×200, $N = 24$, $t = 0.02$, $f = 0.25$.

1. SG dimensions: 200×200, 10 data points, relative distance mode off.
2. SG dimensions: 420×200, 40 data points, relative distance mode on.

See Fig. 4. *Real time using 4 CPUs:* $\sim 6.4\,\mathrm{s}$ *(1) and* $\sim 28.7\,\mathrm{s}$ *(2).*

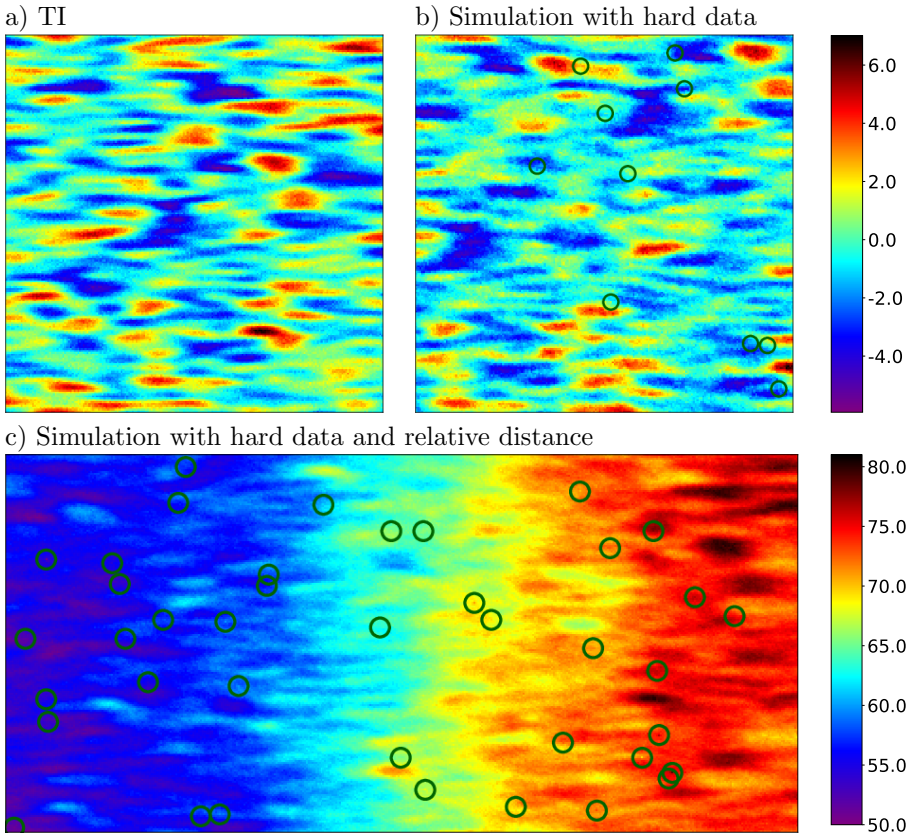


Fig. 4 Example 4: continuous simulation with hard data. a) TI of size 200×200, b) one realization of size 200×200, with 10 data points, relative mode off (Example 4.1), c) one realization of size 420×200, with 40 data points, relative mode on (Example 4.2). The data points are located at the center of the circles.

9 Simulations using homothety and rotation

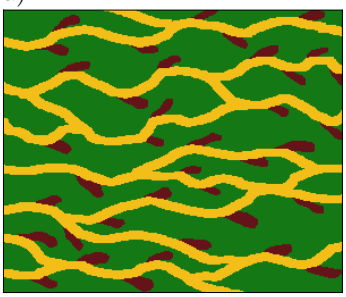
Geometrical transformations (homothethy and rotation) can be defined globally (for the entire SG) or locally, and with or without tolerance. These cases are illustrated below: simulation using global transformations in Example 5, and local transformations in Example 6.

Example 5. Categorical simulation using global homothety and rotation. TI dimensions: 300×250, SG dimensions: 400×300, $N = 24$, $t = 0.05$, $f = 0.25$. Homothety and rotation parameters:

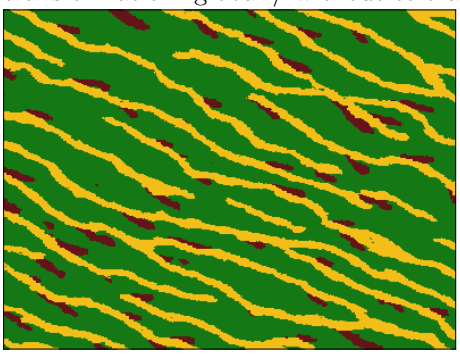
1. without tolerance, homothety ratios: $t_x = 1.33$, $t_y = 0.75$, $t_z = 1$, rotation angles: $\alpha = 20.0$, $\beta = \gamma = 0.0$;
2. with tolerance, homothety ratios: $t_x, t_y \in [0.5, 2.0]$, $t_z = 1$, rotation angles: $\alpha \in [0.0, 180.0]$, $\beta = \gamma = 0.0$.

See Fig. 5. *Real time using 4 CPUs:* $\sim 3.5\,\mathrm{s}$ *(1) and* $\sim 7.2\,\mathrm{s}$ *(2).*

a) TI



b) Simulation using geometrical transformation: global / without tolerance



c) Simulation using geometrical transformation: global / with tolerance

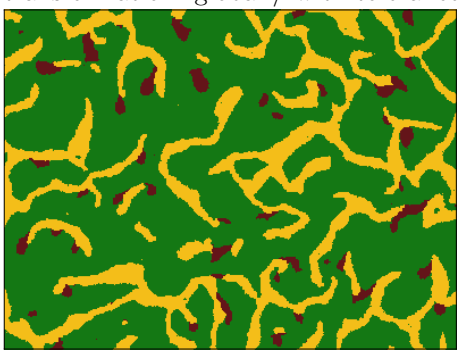


Fig. 5 Example 5: simulation using global geometrical transformation. a) TI of size 300×250, b-c) one realization of size 400×300, b) transformation without tolerance, homothety ratios: $t_x = 1.33$, $t_y = 0.75$, azimuth angle: $\alpha = 20.0$ (Example 5.1), c) transformation with tolerance, homothety ratios: $t_y, t_x \in [0.5, 2.0]$, azimuth angle: $\alpha \in [0.0, 180.0]$ (Example 5.2).

Example 6. Categorical simulation using local homothety and rotation. TI dimensions: 300×250, SG dimensions: 400×300, $N = 24$, $t = 0.05$, $f = 0.25$. Homothety and rotation:

1. without tolerance, see Fig. 6;
2. with tolerance, see Fig. 7.

Note that the minimal and maximal homothety ratios (resp. rotation angles) in Fig. 7c (resp. 7d) are obtained by adding and substracting 0.3 (resp. 45) to/from the homothety ratio (resp. rotation angle) of Fig. 6c (resp. 6d). *Real time using 4 CPUs:* $\sim 4.3\,\mathrm{s}$ *(1) and* $\sim 9.5\,\mathrm{s}$ *(2).*

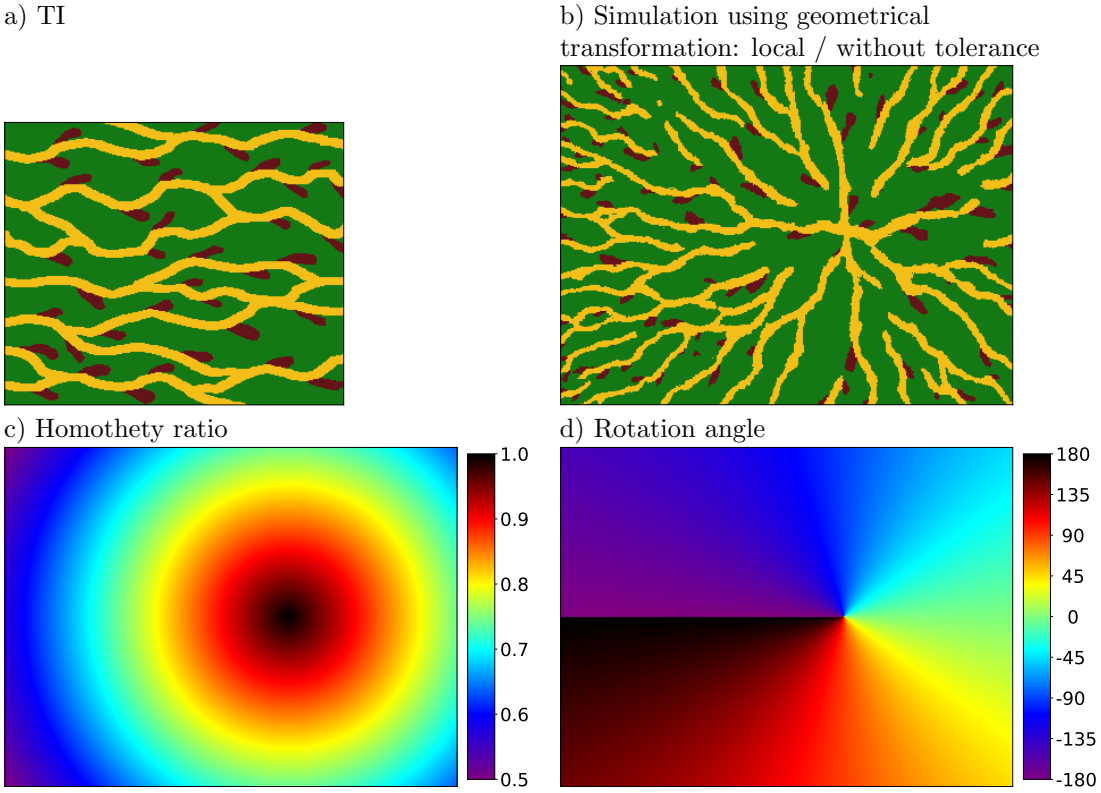


Fig. 6 Example 6.1: simulation using local geometrical transformation without tolerance. a) TI of size 300×250, b) one realization of size 400×300, c) homothety ratio t_x and t_y, d) azimuth α.

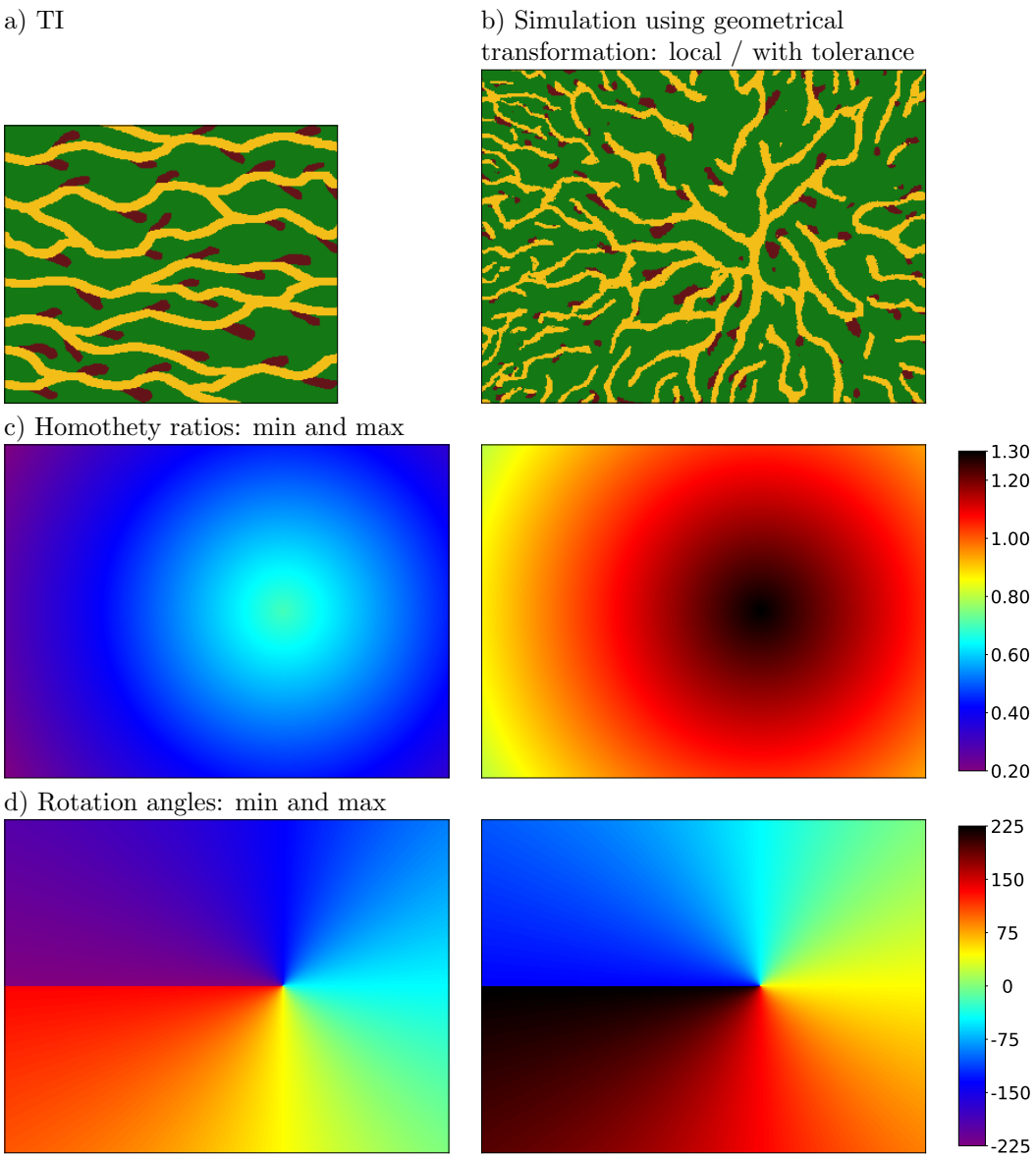


Fig. 7 Example 6.2: simulation using local geometrical transformation with tolerance. a) TI of size 300×250, b) one realization of size 400×300, c) lower (left) and upper (right) bounds for homothety ratio t_x and t_y, d) lower (left) and upper (right) bounds for azimuth α.

10 Bivariate simulations

Example 7. Bivariate simulation. TI dimensions: 300×250, SG dimensions: 350×150, first variable: categorical, second variable: continuous, $N = (18, 18)$, $t = (0.05, 0.02)$, $f = 0.25$, simulation mode: "full 4D" (`SIM_ONE_BY_ONE`). See Fig. 8. *Real time using 4 CPUs:* ~ 36.5 s.

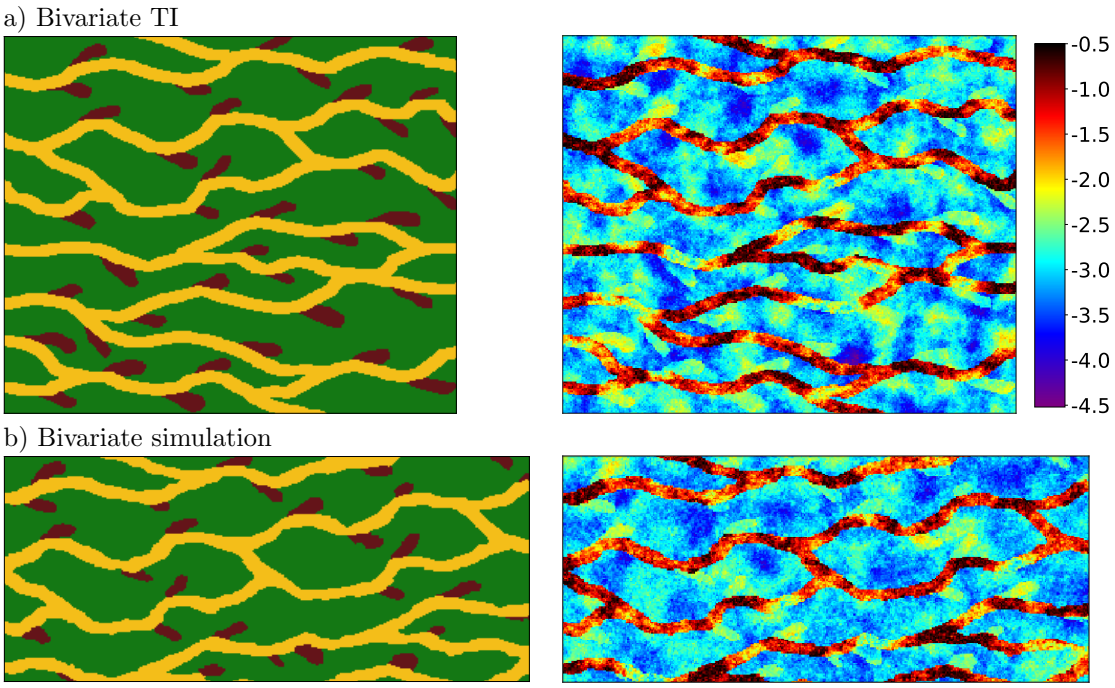


Fig. 8 Example 7: bivariate simulation. a) TI of size 300×250, b) one realization of size 350×150. The same color bar is used for the second variable of the TI and the simulation.

11 Simulation using a mask

The SG has a box shape. Using a mask allows to simulate only some part of the SG. The mask consists in a univariate image with the same grid as the SG, the variable must be defined in all nodes, with value 0 for nodes not to be simulated and 1 for nodes to be simulated. In the ouptut realizations, the unsimulated nodes has the value `MPDS_MISSING_VALUE` ($-9\,999\,999$). Note that if hard data are in the unsimulated part of the SG, they are removed. Note also that for multivariate simulation, the mask is applied to every variable.

Example 8. Categorical simulation using a mask. TI dimensions: 300×250, SG dimensions: 400×200, $N = 24$, $t = 0.05$, $f = 0.25$. See Fig. 9. *Real time using 4 CPUs:* ~ 1.3 s.

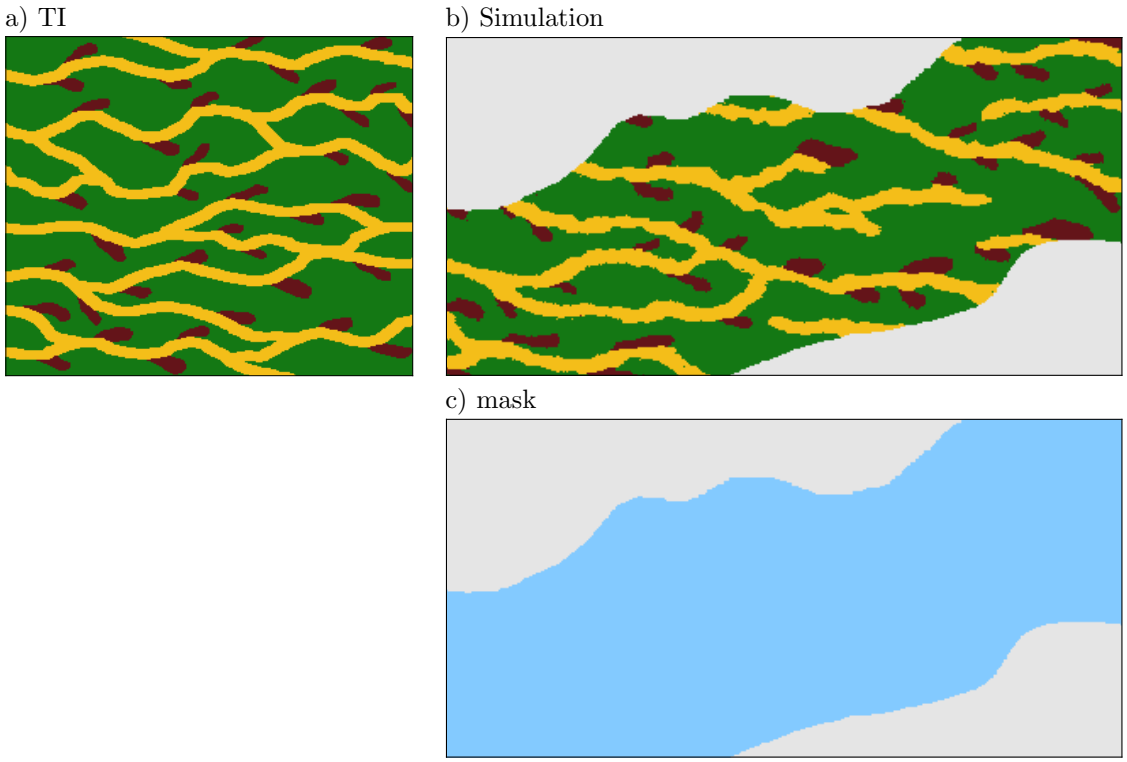


Fig. 9 Example 8: simulation using a mask. a) TI of size 300×250, b) one realization of size 400×200, c) mask, binary image: code 0 (gray) for nodes not to be simulated, code 1 (light blue) for nodes to be simulated.

12 Simulations using an incomplete TI / reconstruction

Incomplete TIs can be used in DeeSse. The uninformed values are set to `MPDS_MISSING_VALUE` ($-9\,999\,999$). However, the TI should contain a part densely informed, so that it contains information about spatial structures at fine scale.

Example 9. Categorical simulation based on an incomplete TI. TI dimensions: 300×250, SG dimensions: 300×250, $N = 24$, $t = 0.05$, $f = 0.25$.

1. Unconditional simulation.
2. Simulation with TI also set as hard data in the SG.
3. Simulation with TI set as hard data in the SG, and SG used as TI.

See Fig. 10. *Real time using 4 CPUs:* ~ 2.1 s *(1) and* ~ 0.6 s *(2), real time using 1 CPU:* ~ 2.1 s *(3).*

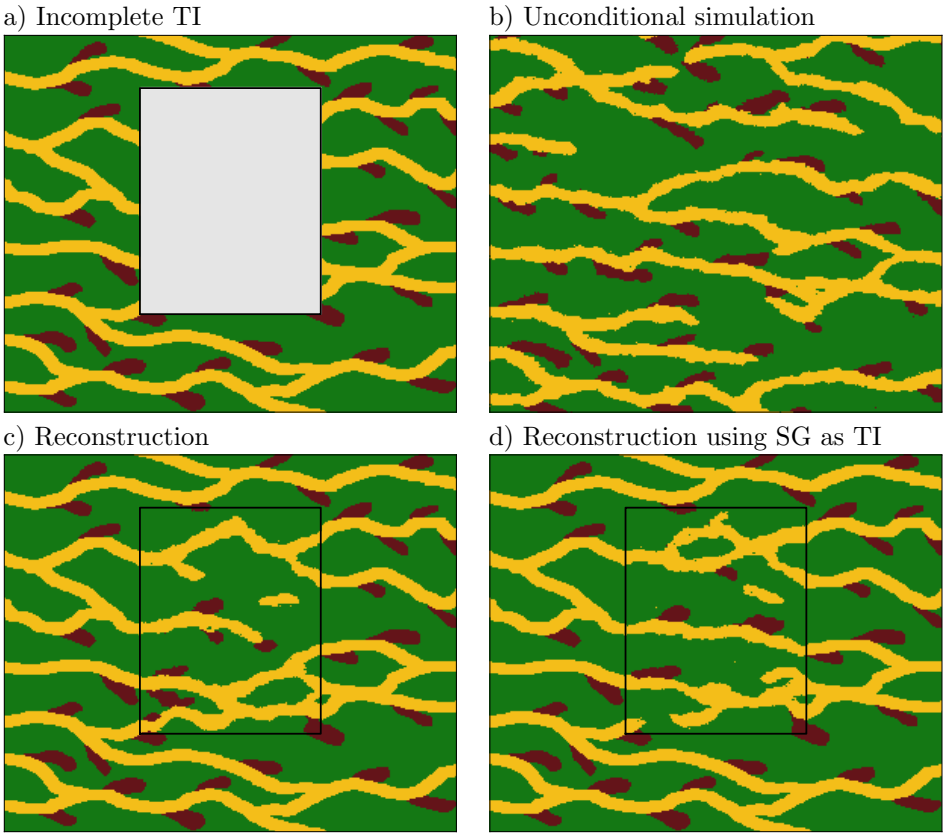


Fig. 10 Example 9: simulation using an incomplete TI. a) TI of size 300×250, b) one unconditional realization (Example 9.1), c) one conditional realization, TI set as hard data in the SG (Example 9.2), d) one conditional realization, TI set as hard data in the SG, and SG used as TI (Example 9.3). In (b-d), the SG has the same dimensions as the TI (300×250).

13 Simulations using multiple TIs

Example 10. Categorical simulation based on two TIs. TI A dimensions: 300×250, TI B dimensions: 200×200, SG dimensions: 480×180, $N = 24$, $t = 0.05$, $f = (0.25, 0.30)$. See Fig. 11. *Real time using 4 CPUs:* $\sim 2.1\,\mathrm{s}$.

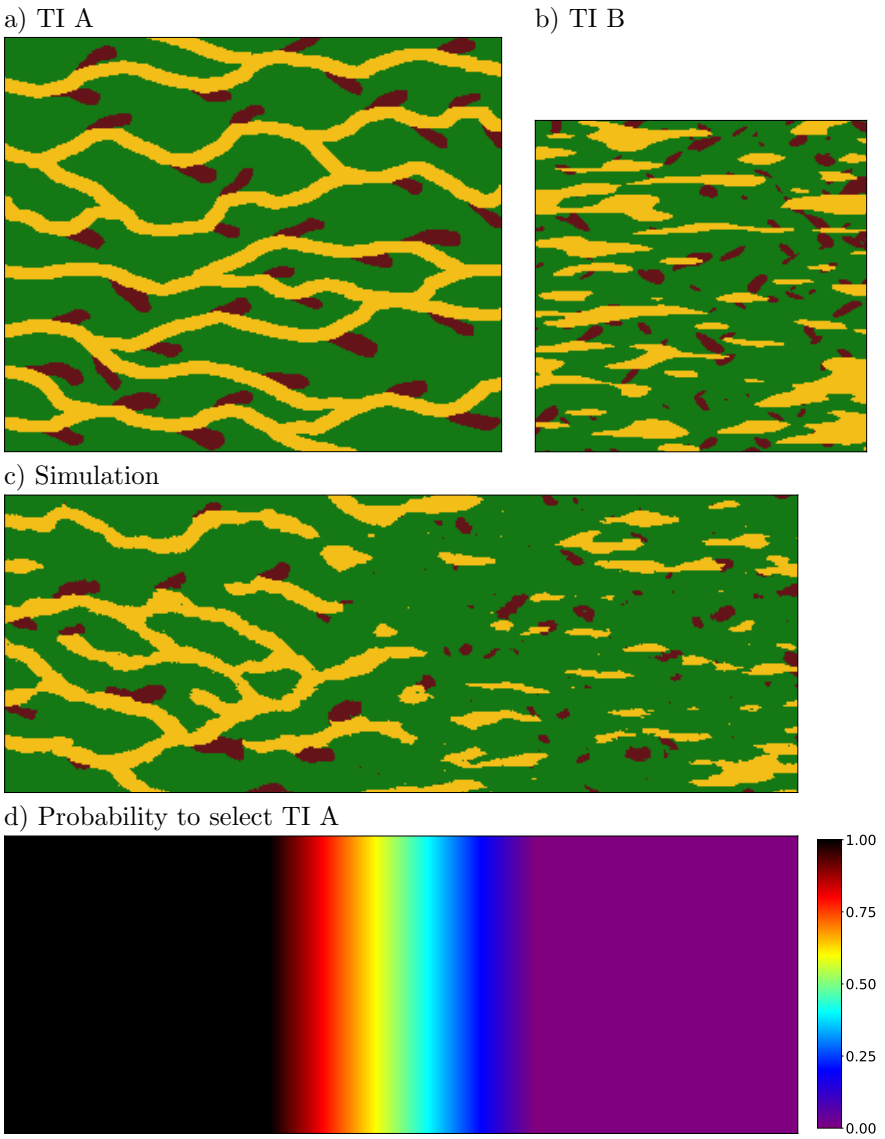


Fig. 11 Example 10: simulation using multiple TIs. a) TI A of size 300×250, b) TI B of size 200×200, c) one realization of size 480×180, d) probability to select TI A.

14 Simulations using probability constraints

Probability constraints consists in target global or local probabilities for specified classes of values. This allows to deal with continuous variable or to group together some categories for a categorical variable. The intersection of two distinct classes must be empty and every conditioning value for the considered variable and every value of the corresponding variable in the TI must be in one class. Below are given examples for categorical simulations (Example 11) and for continuous simulations (Example 12).

Example 11. Categorical simulation (3 categories) with probability constraints. TI from Allard et al. [2011], dimensions: 114×114, SG dimensions: 300×300, $N = 24$, $t = 0.05$, $f = 0.7$. Probability constraints: 3 classes of values corresponding to the category codes of the TI: ($\{0\}$, $\{1\}$, $\{2\}$).

1. Global constraints, target PDF: $(0.2, 0.5, 0.3)$, comparing PDFs using method "MLikRopt", deactivation distance: 4.0, constant threshold: 0.001. See Fig. 12.
2. Local constraints, target PDF: maps for code 1 and 2 in Figs 13c and 13d respectively, probability map for code 0 is the complementary to 1 which is constant (equal to 0.4), local current PDFs computed using mode "complete", comparing PDFs using method "MLikRopt", support radius (distance): 12.0, deactivation distance: 4.0, constant threshold: 0.001. See Fig. 13.

Real time using 1 CPU: $\sim 25.8\,\mathrm{s}$ *(1) and real time using 4 CPUs:* $\sim 7.2\,\mathrm{s}$ *(2).*

Example 12. Continuous simulation with probability constraints. TI from Zhang et al. [2006], dimensions: 200×200, SG dimensions: 200×200, $N = 24$, $t = 0.05$, $f = 0.5$.

1. Global probability constraints, 10 classes of values: regular subdivision of interval $[0, 256[$, target PDF: uniform, comparing PDFs using method "MLikRopt", deactivation distance: 2.0, constant threshold: 0.0. No post-processing. See Fig. 14.
2. Local probability constraints, 2 classes of values: $[0, 50[$ and $[50, 256[$, target PDF: given in in Figs 15c and 15d, local current PDFs computed using mode "complete", comparing PDFs using method "MLikRopt", support radius (distance): 12.0, deactivation distance: 4.0, constant threshold: 0.001. See Fig. 15.

Real time using 1 CPU: $\sim 42.5\,\mathrm{s}$ *(1) and real time using 4 CPUs:* $\sim 10.6\,\mathrm{s}$ *(2).*

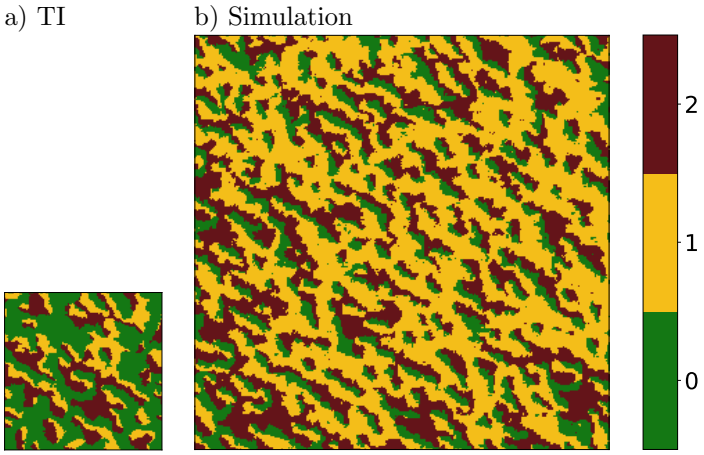


Fig. 12 Example 11.1: categorical simulation using global probability constraints. a) TI from Allard et al. [2011] of size 114×114, global categories proportions: $(0.515, 0.231, 0.254)$, b) one realization of size 300×300, global categories proportion: $(0.197, 0.508, 0.295)$ [target was $(0.2, 0.5, 0.3)$].

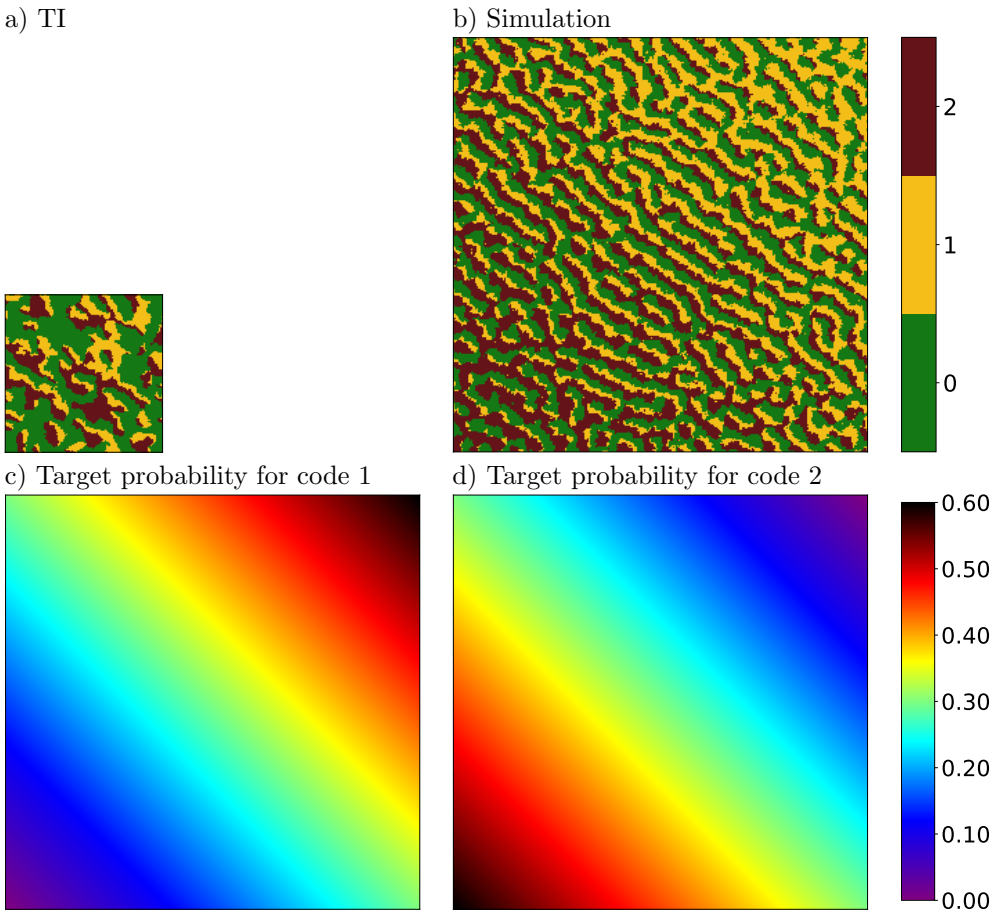


Fig. 13 Example 11.2: categorical simulation using local probability constraints. a) TI from Allard et al. [2011] of size 114×114, b) one realization of size 300×300, c-d) target probability for category 1 (c) and 2 (d) [target probability for category 0 is constant equal to 0.4].

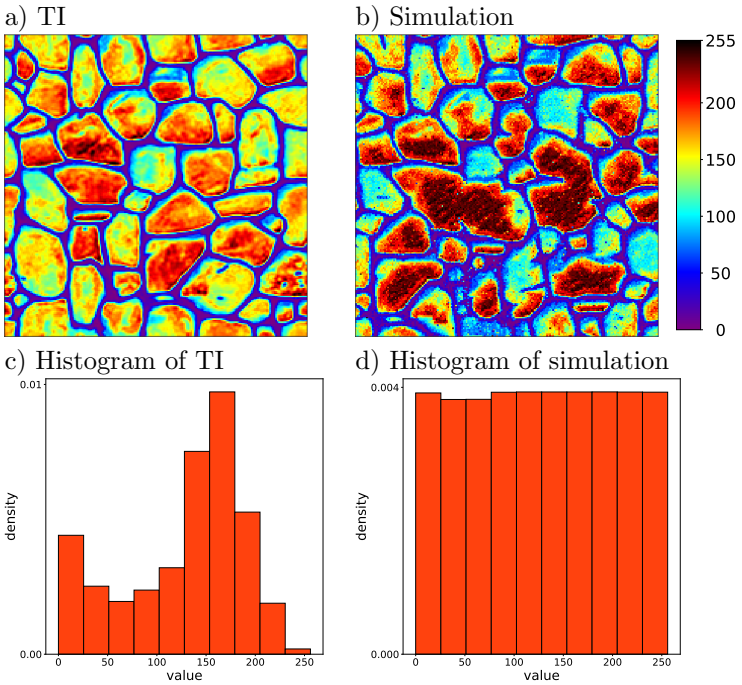


Fig. 14 Example 12.1: continuous simulation using global probability constraints. a) TI from Zhang et al. [2006] of size 200×200, b) one realization of size 200×200, c-d) histogram of TI values (c) and simulated values (d) over each target classes [target probability: uniform].

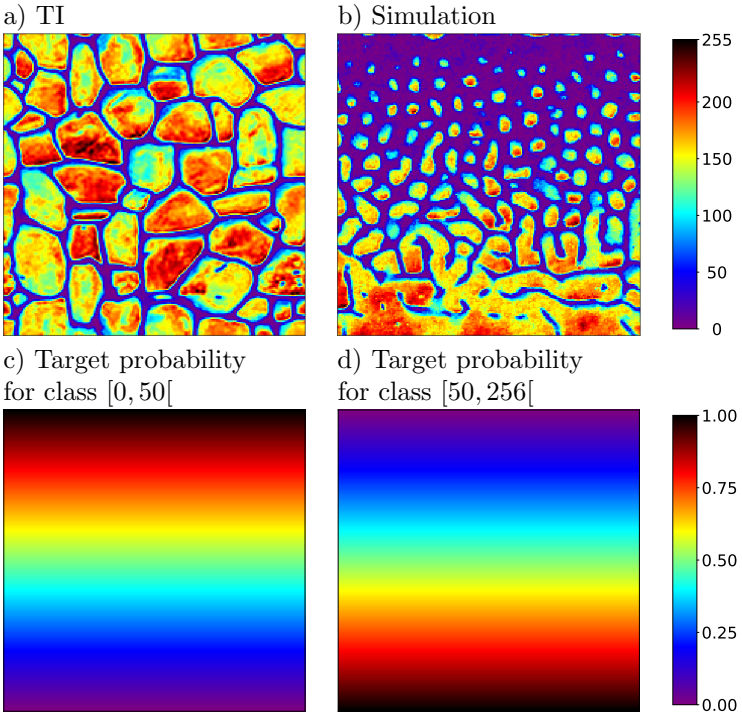


Fig. 15 Example 12.2: continuous simulation using local probability constraints. a) TI from Zhang et al. [2006] of size 200×200, b) one realization of size 200×200, c-d) target probability for class $[0, 50[$ (c) and $[50, 256[$ (d).

15 Simulations with block data

DeeSse allows to impose target (arithmethic) mean value on subsets of grid nodes (block). These subsets may overlap.

Example 13. Continuous simulation with block data. TI from Straubhaar et al. [2016], dimensions: 764×239, SG dimensions: 180×120, $N = 24$, $t = 0.02$, $f = 0.25$.

1. Block data: 90 rectangular non-overlapping blocks of size 20×12 covering the entire SG, for all blocks: tolerance is set to 0.5 and activation proportion min and max to 0.0 and 1.0. See Fig. 16.

2. Block data: 6 overlapping blocks having a disk shape of various size, for all blocks: tolerance is set to 0.02 and activation proportion min and max to 0.0 and 1.0. See Fig. 17.

Real time using 1 CPU: $\sim 2\,\text{min}6.2\,\text{s}$ *(1) and* $\sim 1\,\text{min}37.4\,\text{s}$ *(2).*

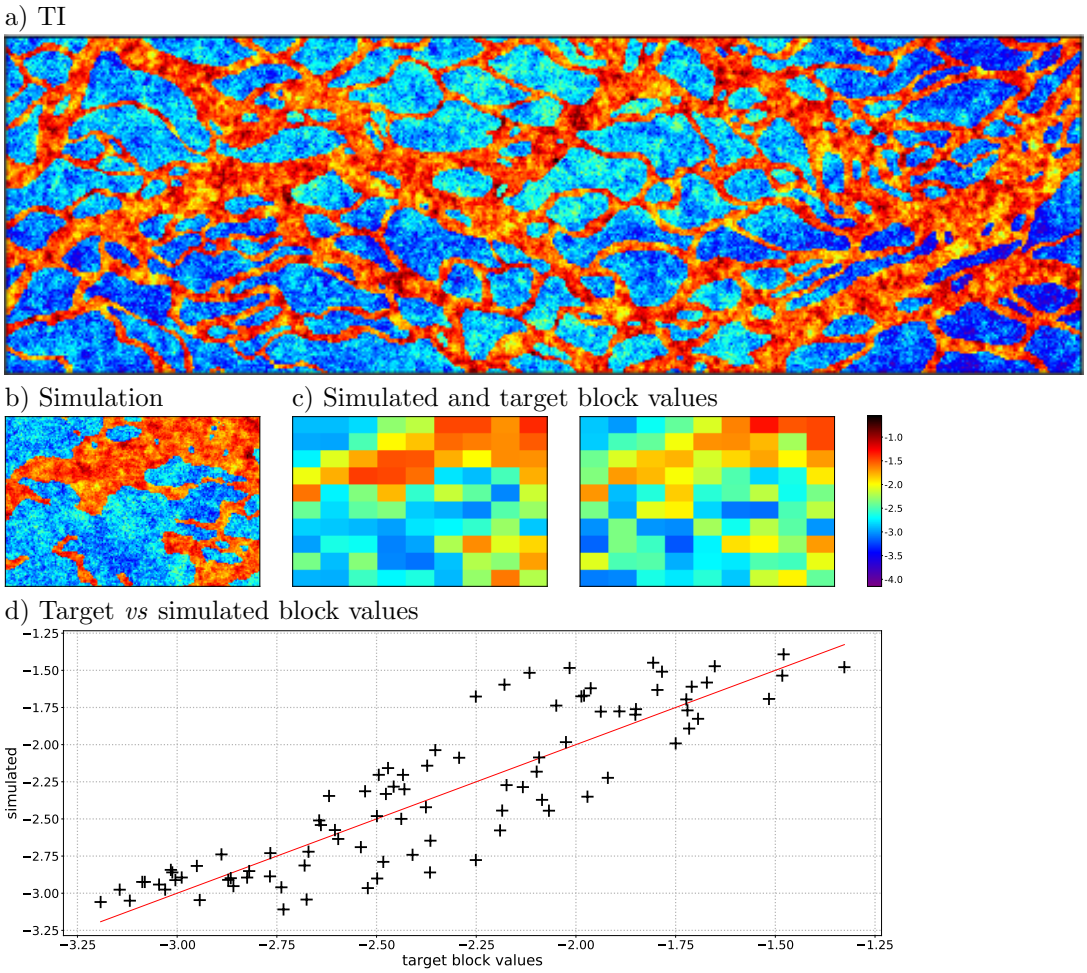


Fig. 16 Example 13.1: continuous simulation with block data. a) TI from Straubhaar et al. [2016] of size 764×239, b) one realization of size 180×120, c) mean values on blocks: computed from simulation (left), and target (right), e) target *vs* simulated block values (the line is the curve $y = x$). The same color bar is used for (a-c).

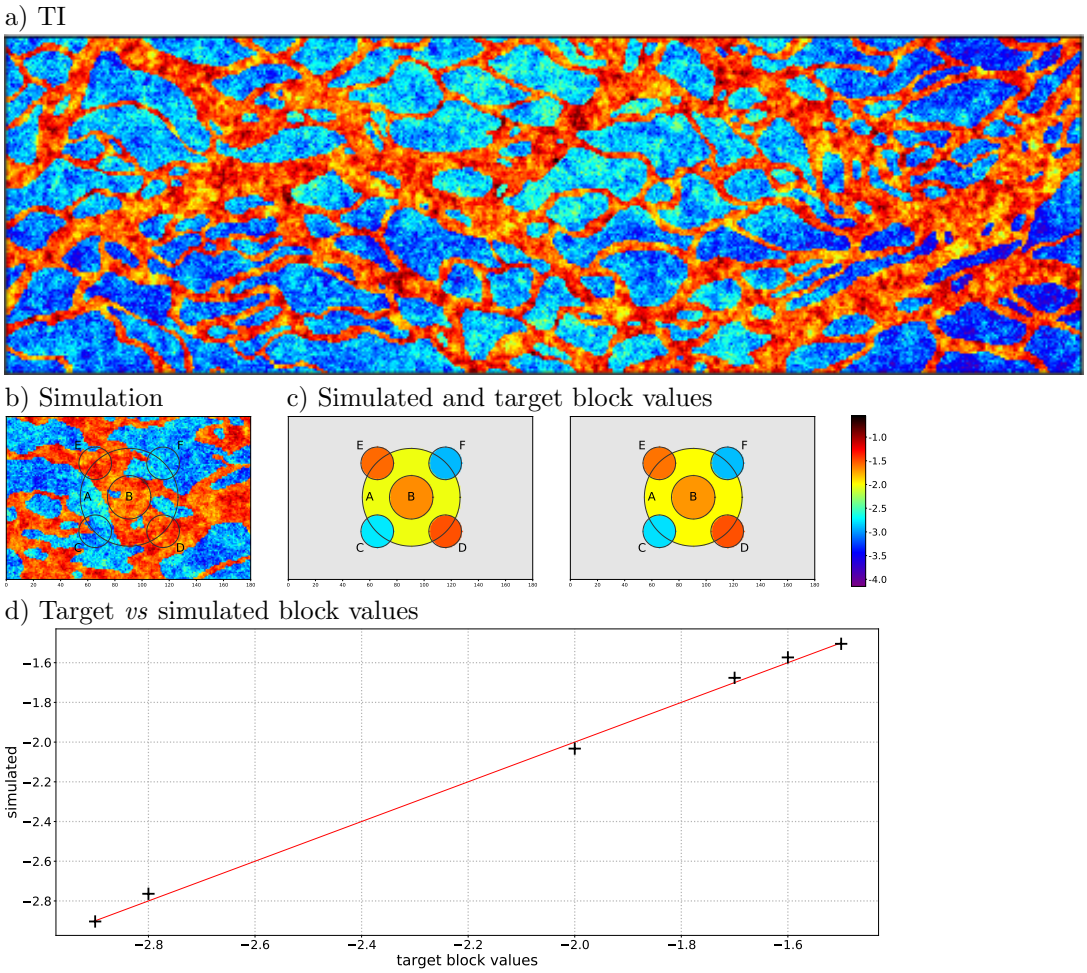


Fig. 17 Example 13.2: continuous simulation with block data. a) TI from Straubhaar et al. [2016] of size 764×239, b) one realization of size 180×120, c) mean values on blocks: computed from simulation (left), and target (right), e) target *vs* simulated block values (the line is the curve $y = x$). The same color bar is used for (a-c).

16 Simulations with connectivity data

Example 14. Categorical simulation (3 categories) with 4 hard data points, with two of them marked as connected together. Parameters for connectivity: data points connected by pasting binding path from the TI before starting the simulation; node connected by face, edge or corner (vertex); one connected class per category; TI dimensions: 300×250, SG dimensions: 240×200, $N = 24$, $t = 0.05$, $f = 0.25$. See Fig. 18. *Real time using 4 CPUs:* $\sim 1.0\,\mathrm{s}$.

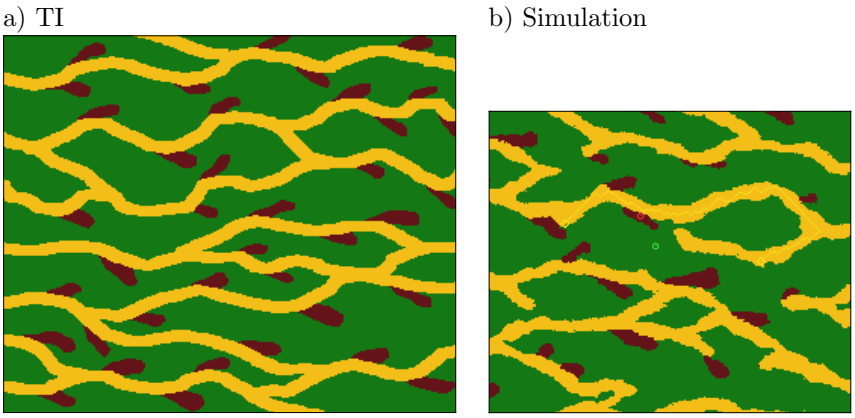


Fig. 18 Example 14: categorical simulation with connectivity data. a) TI of size 300×250, b) one realization of size 240×200. The data points are located at the center of the circles and diamonds; diamonds indicate data points with connectivity data; connected path pasted from the TI is highlighted in the simulation.

17 Simulations using pyramids (multiple scales)

DeeSse allows for simulations accounting for multiple scales based on Gaussian pyramids. The TI is considered at different scales by constructing its Gaussian pyramid: the level 0 is the initial TI, and for each next level, a mobile weighted average is computed (convolution) from the previous level and the dimensions, *i.e.* the numbers of nodes along each direction, are reduced. Depending on the selected pyramid simulation mode, the resulting images are also expanded onto the previous level. Then, the simulation is done through all levels, which may help to reproduce spatial features more properly.

Note that, mobile average are computed on the considered variable directly if it is continuous, and on the indicator variable of categories if it is categorical.

Example 15. Categorical simulation using pyramids. TI dimensions: 300×250, SG dimensions: 400×400, $N = 24$, $t = 0.05$, $f = 0.25$. Pyramids: number of additional levels $N = 2$, reduction factors $k_x = k_y = 2$ ($k_z = 0$) for levels 0 and 1, pyramid simulation mode: "hierarchical with expansion" (`PYRAMID_SIM_HIERARCHICAL_USING_EXPANSION`), default values for adpating factors (for maximal number of neighbors and distance threshold), pyramid type `PYRAMID_CATEGORICAL_AUTO`. See Figs 19 and 20. *Real time using 4 CPUs:* ~ 3.7 s.

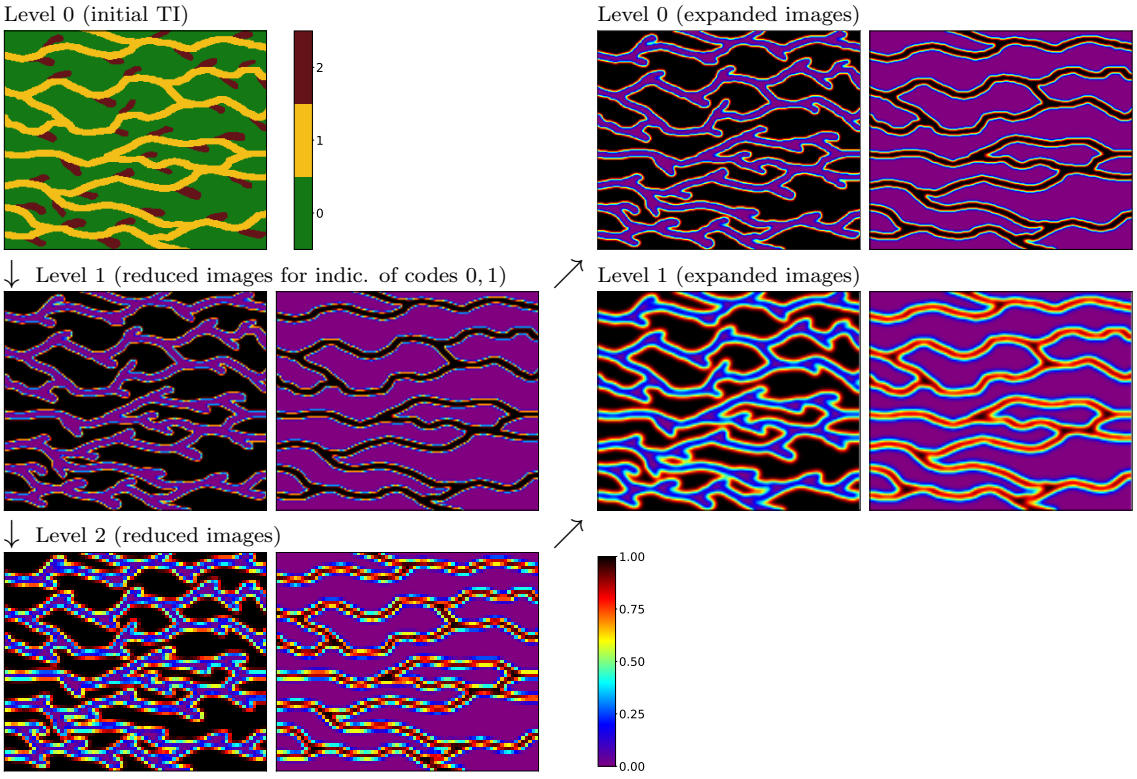


Fig. 19 Example 15: TI pyramid. Only the TI image at top left is given by the user, the other TI images are computed. Dimensions: 300×250 at level 0, 150×125 at level 1, 75×63 at level 2. The same color bar is used for all continuous images.

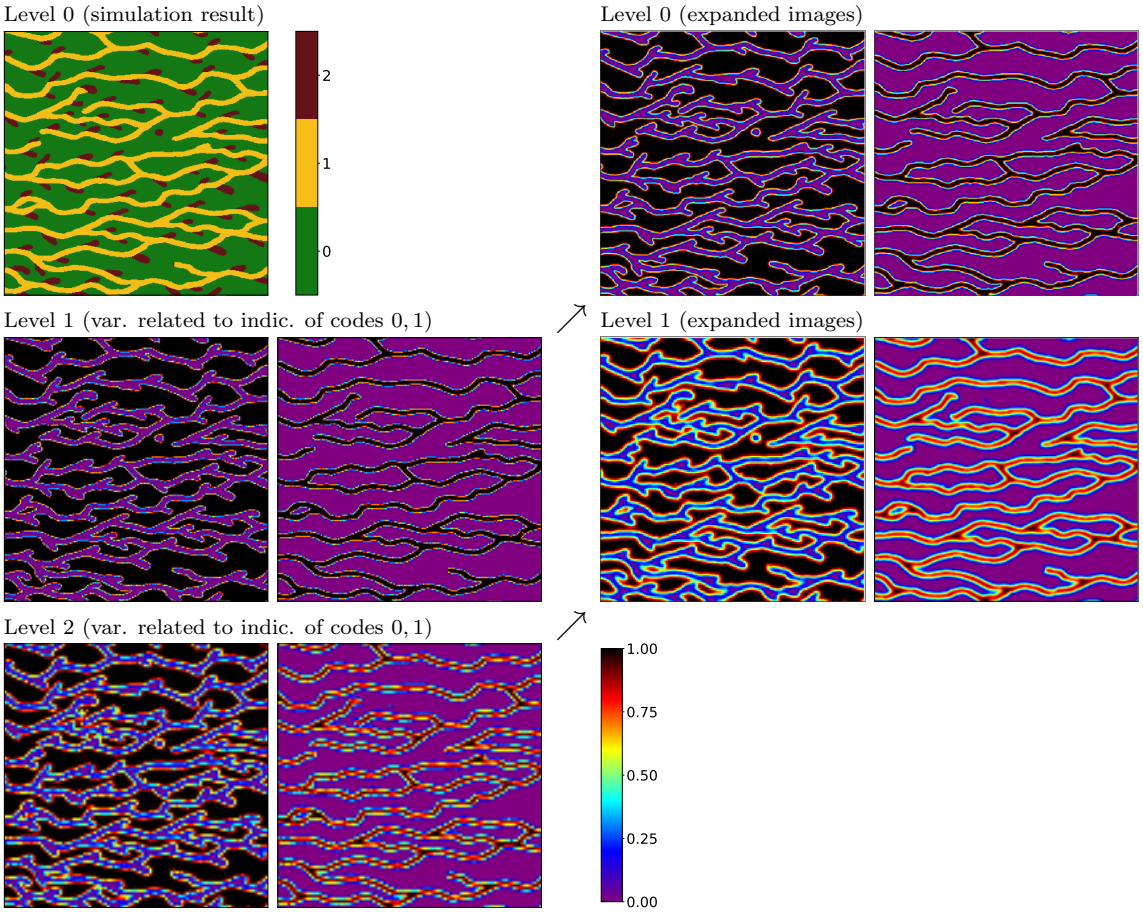


Fig. 20 Example 15: simulation result using pyramids. Simulation procedure: i) simulation at level 2, ii) expansion at level 1, iii) conditional simulation at level 1, iv) expansion at level 0, v) conditional simulation at level 0. Dimensions: 400×400 at level 0, 200×200 at level 1, 100×100 at level 2. The same color bar is used for all continuous images. *Note that only the result at top left is retrieved as output by* DeeSse.

Part III

Appendix

A Example of input parameters file for DeeSse

```
/* SIMULATION GRID (SG) */
240    200     1    // size    in each direction
  1.0    1.0   1.0  // spacing in each direction
  0.0    0.0   0.0  // origin

/* SIMULATION VARIABLES */
/* Number of simulation variable(s), and for each variable:
      variable name, output flag (0 / 1), and if output flag is 1: format string (as passed to fprintf).
   Write DEFAULT_FORMAT for format string to use default format.
   Example (with 3 variables):
      3
      varName1  1  %10.5E
      varName2  0
      varName3  1  DEFAULT_FORMAT
*/
1
code      1  DEFAULT_FORMAT

/* OUTPUT SETTINGS FOR SIMULATION */
/* Key word and required name(s) or prefix, for output of the realizations:
      - OUTPUT_SIM_NO_FILE:
           no file in output
      - OUTPUT_SIM_ALL_IN_ONE_FILE
           one file in output (every variable in output (flagged as 1 above),
           for all realizations will be written),
           - requires one file name
      - OUTPUT_SIM_ONE_FILE_PER_VARIABLE:
            one file per variable in output (flagged as 1 above),
            - requires as many file name(s) as variable(s) flagged as 1 above
      - OUTPUT_SIM_ONE_FILE_PER_REALIZATION:
           one file per realization,
           - requires one prefix (for file name, the string "_real#####.gslib"
             will be appended where "######" is the realization index)
*/
OUTPUT_SIM_ONE_FILE_PER_REALIZATION
test

/* OUTPUT: ADDITIONAL MAPS */
/* Additional maps (images) can be retrieved in output (one file per realization).
      - Flag ( 0 / 1) for path index map (index in the simulation path is attached
        to each simulation grid node), and if 1, one prefix (for file name)
      - Flag ( 0 / 1) for error map (error e, see TOLERANCE below) obtained
        during the simulation (excluding possible post-processing paths),
        and if 1, one prefix (for file name)
   Note: the string "_real#####.gslib" will be appended to these prefix, where
   "######" is the realization index. */
0 // path index map
0 // error map

/* OUTPUT REPORT */
/* Flag (0 / 1), and if 1, output report file. */
1
test_report.txt

/* TRAINING IMAGE */
/* Number of training image(s) (nTI >= 1), followed by nTI file(s)
   (a file can be replaced by the string "_DATA_" which means that the
```

```
   simulation grid itself is taken as training image), and
   if nTI > 1, one pdf image file (for training images, nTI variables). */
1
ti.gslib

/* DATA IMAGE FILE FOR SG */
/* Number of image file(s) (n >= 0), followed by n file(s). */
0

/* DATA POINT SET FILE FOR SG */
/* Number of point set file(s) (n >= 0), followed by n file(s). */
0

/* MASK IMAGE */
/* Flag (0: mask not used / 1: mask used) and if 1, mask image file
   (this image contains one variable on the simulation grid: flag (0 / 1)
   for each node of the simulation grid that indicates if the variable(s)
   will be simulated at the corresponding node (flag 1) or not (flag 0). */
0

/* HOMOTHETY */
/* 1. Homothety usage, integer (homothetyUsage):
        - 0: no homothety
        - 1: homothety without tolerance
        - 2: homothety with tolerance
   2a. If homothetyUsage == 1,
          then for homothety ratio in each direction,
          first for x, then for y, and then for z-axis direction:
             - Flag (0 / 1) indicating if given in an image file,
               followed by
                  - one value (real) if flag is 0
                  - name of the image file (one variable) if flag is 1
   2b. If homothetyUsage == 2,
          then for homothety ratio in each direction,
          first for x, then for y, and then for z-axis direction:
             - Flag (0 / 1) indicating if given in an image file,
               followed by
                  - two values (lower and upper bounds) (real) if flag is 0
                  - name of the image file (two variables) if flag is 1
*/
0

/* ROTATION */
/* 1. Rotation usage, integer (rotationUsage):
        - 0: no rotation
        - 1: rotation without tolerance
        - 2: rotation with tolerance
   2a. If rotationUsage == 1,
          then for each angle,
          first for azimuth, then for dip, and then for plunge:
             - Flag (0 / 1) indicating if given in an image file,
               followed by
                  - one value (real) if flag is 0
                  - name of the image file (one variable) if flag is 1
   2b. If rotationUsage == 2,
          then for each angle,
          first for azimuth, then for dip, and then for plunge:
             - Flag (0 / 1) indicating if given in an image file,
               followed by
                  - two values (lower and upper bounds) (real) if flag is 0
                  - name of the image file (two variables) if flag is 1
*/
0

/* CONSISTENCY OF CONDITIONING DATA (TOLERANCE RELATIVELY TO THE RANGE OF TRAINING VALUES) */
/* Maximal expansion (expMax): real number (negative to not check consistency).
   The following is applied for each variable separetely:
```

```
      - For variable with distance type set to 0 (see below):
          * expMax >= 0:
               if a conditioning data value is not in the set of training image values,
               an error occurs
          * expMax < 0:
               if a conditioning data value is not in the set of training image values,
               a warning is displayed (no error occurs)
      - For variable with distance type not set to 0 (see below): if relative distance
        flag is set to 1 (see below), nothing is done, else:
          * expMax >= 0: maximal accepted expansion of the range of the training image values
               for covering the conditioning data values:
               - if conditioning data values are within the range of the training image values:
                    nothing is done
               - if a conditioning data value is out of the range of the training image values:
                    let
                       new_min_ti = min ( min_cd, min_ti )
                       new_max_ti = max ( max_cd, max_ti )
                    with
                       min_cd, max_cd, the min and max of the conditioning values,
                       min_ti, max_ti, the min and max of the training imges values.
                    If new_max_ti - new_min_ti <= (1 + expMax) * (ti_max - ti_min), then
                    the training image values are linearly rescaled from [ti_min, ti_max] to
                    [new_ti_min, new_ti_max], and a warning is displayed (no error occurs).
                    Otherwise, an error occurs.
          * expMax < 0: if a conditioning data value is out of the range of the training image
               values, a warning is displayed (no error occurs), the training image values are
               not modified.
*/
0.05

/* NORMALIZATION TYPE (FOR VARIABLES FOR WHICH DISTANCE TYPE IS NOT 0 AND DISTANCE IS ABSOLUTE) */
/* Available types:
      - NORMALIZING_LINEAR
      - NORMALIZING_UNIFORM
      - NORMALIZING_NORMAL */
NORMALIZING_LINEAR

/* SEARCH NEIGHBORHOOD PARAMETERS */
/* A search neighborhood is a 3D ellipsoid, defined by:
      - (rx, ry, rz): search radius (in number of nodes) for each direction
      - (ax, ay, az): anisotropy ratio or inverse distance unit for each direction;
        the distance to the central node is computed as the Euclidean distance by
        considering the nodes with the unit 1/ax x 1/ay x 1/az;
      - (angle1, angle2, angle3): angles (azimuth, dip and plunge) defining the
        rotation that sends the coordinates system xyz onto the coordinates
        system x'y'z' in which the search radii and the anisotropy ratios are given.
      - power: at which the distance is elevated for computing the weight of each
        node in the search neighborhood.
   Note that
     - the search neighborhood is delimited by the search radii and the angles
     - the anisotropy ratios are used only for computing the distance to the central
       node, from each node in the search neighborhood
     - the nodes inside the search neighborhood are sorted according to their
       distance to the central node, from the closest one to the furthest one

   FOR EACH VARIABLE:
      1. search radii, available specifications (format: key word [parameters]):
            SEARCHNEIGHBORHOOD_RADIUS_LARGE_DEFAULT
            SEARCHNEIGHBORHOOD_RADIUS_TI_RANGE_DEFAULT
            SEARCHNEIGHBORHOOD_RADIUS_TI_RANGE
            SEARCHNEIGHBORHOOD_RADIUS_TI_RANGE_XY
            SEARCHNEIGHBORHOOD_RADIUS_TI_RANGE_XZ
            SEARCHNEIGHBORHOOD_RADIUS_TI_RANGE_YZ
            SEARCHNEIGHBORHOOD_RADIUS_TI_RANGE_XYZ
            SEARCHNEIGHBORHOOD_RADIUS_MANUAL rx ry rz
         with the following meaning, where tx, ty, tz denote the ranges of the
         variogram of the TI(s) in x-, y-, z-directions:
```

```
    - SEARCHNEIGHBORHOOD_RADIUS_LARGE_DEFAULT:
        large radii set according to the size of the SG and the TI(s),
        and the use of homothethy and/or rotation for the simulation
        (automatically computed)
    - SEARCHNEIGHBORHOOD_RADIUS_TI_RANGE_DEFAULT:
        search radii set according to the TI(s) variogram ranges,
        one of the 5 next modes SEARCHNEIGHBORHOOD_RADIUS_TI_RANGE* will be
        used according to the use of homothethy and/or rotation for the
        simulation
        (automatically computed)
    - SEARCHNEIGHBORHOOD_RADIUS_TI_RANGE:
        rx prop. to tx, ry prop. to ty, rz prop. to tz,
        independently in each direction
        (automatically computed)
    - SEARCHNEIGHBORHOOD_RADIUS_TI_RANGE_XY:
        rx = ry  prop. to max(tx, ty), independently from rz prop. to tz
        (automatically computed)
    - SEARCHNEIGHBORHOOD_RADIUS_TI_RANGE_XZ:
        rx = rz  prop. to max(tx, tz), independently from ry prop. to ty
        (automatically computed)
    - SEARCHNEIGHBORHOOD_RADIUS_TI_RANGE_YZ:
        ry = rz  prop. to max(ty, tz), independently from rx prop. to tx
        (automatically computed)
    - SEARCHNEIGHBORHOOD_RADIUS_TI_RANGE_XYZ:
        rx = ry = rz prop. to max(tx, ty, tz)
        (automatically computed)
    - SEARCHNEIGHBORHOOD_RADIUS_MANUAL:
        search radii are explicitly given

2. anisotropy ratios, available specifications (format: key word [parameters]):
    SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_ONE
    SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_RADIUS
    SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_RADIUS_XY
    SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_RADIUS_XZ
    SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_RADIUS_YZ
    SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_RADIUS_XYZ
    SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_MANUAL ax ay az
  with the following meaning:
    - SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_ONE:
        ax = ay = az = 1
        (isotropic distance:
        maximal distance for search neighborhood nodes will be equal to the
        maximum of the search radii)
    - SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_RADIUS:
        ax = rx, ay = ry, az = rz
        (anisotropic distance:
        nodes at distance one on the border of the search neighborhood,
        maximal distance for search neighborhood nodes will be 1)
    - SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_RADIUS_XY:
        ax = ay = max(rx, ry), az = rz
        (anisotropic distance:
        maximal distance for search neighborhood nodes will be 1)
    - SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_RADIUS_XZ:
        ax = az = max(rx, rz), ay = ry
        (anisotropic distance:
        maximal distance for search neighborhood nodes will be 1)
    - SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_RADIUS_YZ:
        ay = az = max(ry, rz), ax = rx
        (anisotropic distance:
        maximal distance for search neighborhood nodes will be 1)
    - SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_RADIUS_XYZ:
        ax = ay = az = max(rx, ry, rz)
        (isotropic distance:
        maximal distance for search neighborhood nodes will be 1)
    - SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_MANUAL:
        anisotropy ratios are explicitly given
```

```
      3. rotation, available specifications (format: key word [parameters]):
            SEARCHNEIGHBORHOOD_ROTATION_OFF
            SEARCHNEIGHBORHOOD_ROTATION_ON angle1 angle2 angle3
         with the following meaning:
            - SEARCHNEIGHBORHOOD_ROTATION_OFF:
                 search neighborhood is not rotated
            - SEARCHNEIGHBORHOOD_ROTATION_ON:
                 search neighborhood is rotated, the rotation is described by
                 angle1 (azimuth), angle2 (dip) and angle3 (plunge)

      4. power (real number)
*/
/* SEARCH NEIGHBORHOOD PARAMETERS FOR VARIABLE #0 */
SEARCHNEIGHBORHOOD_RADIUS_LARGE_DEFAULT          // search radii
SEARCHNEIGHBORHOOD_ANISOTROPY_RATIO_ONE          // anisotropy ratios
SEARCHNEIGHBORHOOD_ROTATION_OFF                  // rotation
0.0                                              // power for computing weight according to distance

/* MAXIMAL NUMBER OF NEIGHBORING NODES FOR EACH VARIABLE (as many number(s) as number of variable(s)) */
24

/* MAXIMAL DENSITY OF NEIGHBORING NODES IN SEARCH NEIGHBORHOOD FOR EACH VARIABLE (as many number(s)
   as number of variable(s)) */
1.0

/* RESCALING MODE FOR EACH VARIABLE (as many specification as number of variable(s))*/
/* Available specifications (format: key word [parameters]):
      RESCALING_NONE
      RESCALING_MIN_MAX min_value max_value
      RESCALING_MEAN_LENGTH mean_value length_value
   with the following meaning:
      - RESCALING_NONE:
           target interval for simulated values = interval of training image(s) value(s)
           (standard mode)
      - RESCALING_MIN_MAX:
           target interval for simulated values given by min_value and max_value
      - RESCALING_MEAN_LENGTH:
           target interval for simulated values given by mean_value and length_value
   For the two last modes, a linear correspondence is set between the target interval
   for simulated values and the interval of the training image(s):
      - for mode RESCALING_MIN_MAX, min_value and max_value are set in correspondence
        with the min and max training image(s) values respectively
      - for mode RESCALING_MEAN_LENGTH, mean_value is set in correspondence
        with the mean training image(s) value, and length_value is the length
        of the target interval.
   Note that:
      - conditioning data values (if any) must be in the target interval.
      - the two last modes are not compatible with distance type 0
        (non-matching nodes), or with relative distance mode (see below)
      - class of values (for other options such probability constraints or
        connectivity constraints) are given according to the target interval.
*/
RESCALING_NONE

/* RELATIVE DISTANCE FLAG FOR EACH VARIABLE (as many flag(s) (0 / 1) as number of variable(s)) */
0

/* DISTANCE TYPE FOR EACH VARIABLE (as many number(s) as number of variable(s)) */
/* Available distance (between data events):
      - 0: non-matching nodes (typically for categorical variable)
      - 1: L-1 distance
      - 2: L-2 distance
      - 3: L-p distance, requires the real positive parameter p
      - 4: L-infinity distance */
0

/* WEIGHT FACTOR FOR CONDITIONING DATA, FOR EACH VARIABLE (as many number(s) as number of variable(s)) */
```

```
/* For the computation of distance between data events, if a value is a conditioning
   data, its corresponding contribution is multiplied by the factor given here. */
1.0

/* SIMULATION TYPE */
/* Key word:
      - SIM_ONE_BY_ONE:
           successive simulation of one variable at one node in the simulation grid (4D path)
      - SIM_VARIABLE_VECTOR:
           successive simulation of all variable(s) at one node in the simulation grid (3D path) */
SIM_ONE_BY_ONE

/* SIMULATION PATH */
/* Available paths (format: key word [parameters]):
      - PATH_RANDOM
      - PATH_RANDOM_HD_DISTANCE_PDF s
      - PATH_RANDOM_HD_DISTANCE_SORT s
      - PATH_RANDOM_HD_DISTANCE_SUM_PDF p s
      - PATH_RANDOM_HD_DISTANCE_SUM_SORT p s
      - PATH_UNILATERAL u (vector)
   with the following meaning:
      - PATH_RANDOM:
           Random path, for simulation type:
              - SIM_ONE_BY_ONE     : path visiting all nodes and variables in a random order
              - SIM_VARIABLE_VECTOR: path visiting all nodes in a random order
      - PATH_RANDOM_HD_DISTANCE_PDF:
           This path preferentially visits first the SG nodes close to the conditioning nodes.
           For any uninformed node, the distance d to the set of conditioning nodes is computed,
           and a weight proportional to a^d is attached, with 0 < a = 1 + s*(a0-1) <= 1, a0
           being the minimal value for a.
           The path is then built by successively drawing nodes according to their weight.
           The required parameter s in [0,1] controls the strength of the distance
           (s = 0: random, s = 1: most guided by the distance).
           Notes: 1) in absence of conditioning node, one random node in the path is considered
           to compute the distances, 2) if simulation type is SIM_VARIABLE_VECTOR, variables
           exhaustively informed are not considered as conditioning data for computing distance.
     - PATH_RANDOM_HD_DISTANCE_SORT:
           This path preferentially visits first the SG nodes close to the conditioning nodes.
           For any uninformed node, the distance d to the set of conditioning nodes is computed,
           and the normalized distance d' in [0,1] is combined with a random number r in [0,1]
           to define a weight w = s*d' + (1-s)*r.
           The path is then built by sorting the nodes such that the weight are in increasing order.
           The required parameter s in [0,1] controls the strength of the distance
           (s = 0: random, s = 1: most guided by the distance - quasi deterministic).
           Notes: 1) in absence of conditioning node, one random node in the path is considered
           to compute the distances, 2) if simulation type is SIM_VARIABLE_VECTOR, variables
           exhaustively informed are not considered as conditioning data for computing distance.
      - PATH_RANDOM_HD_DISTANCE_SUM_PDF:
           The path is built as for PATH_RANDOM_HD_DISTANCE_PDF, but the distance to the set of
           conditioning nodes is replaced by the sum of the distances to every conditioning node,
           each distance being raised to a power p.
           Two parameters are required: p > 0 controls the computation of the distances,
           and s in [0,1] controls the strength of the distance
           (s = 0: random, s = 1: most guided by the distance).
           Note: 1) in absence of conditioning node, random path is considered.
      - PATH_RANDOM_HD_DISTANCE_SUM_SORT:
           The path is built as for PATH_RANDOM_HD_DISTANCE_SORT, but the distance to the set of
           conditioning nodes is replaced by the sum of the distances to every conditioning node,
           each distance being raised to a power p.
           Two parameters are required: p > 0 controls the computation of the distances,
           and s in [0,1] controls the strength of the distance
           (s = 0: random, s = 1: most guided by the distance - quasi deterministic).
           Note: 1) in absence of conditioning node, random path is considered.
      - PATH_UNILATERAL:
           It requires a vector u of parameters of size n=4 (resp. n=3) for simulation type set to
           SIM_ONE_BY_ONE (resp. SIM_VARIABLE_VECTOR), given as ux uy uz uv (resp. ux uy uz).
           Some components of u can be equal to 0, and the other ones must be the integer 1,...,m,
```

```
            with sign +/-. For
                u[j(1)] = ... = u[j(n-m)] = 0 and (u[i(1)], ..., u[i(m)]) = (+/-1,...,+/-m),
            the path visits all nodes in sections of coordinates j(1),...,j(n-m) randomly,
            and makes varying the i(1)-th coordinate first (most rapidly), ..., the i(m)-th coordinate
            last, each one in increasing or decreasing order according to the sign (+ or - resp.).
            Examples:
              - for simulation type SIM_ONE_BY_ONE, and u = (0, -2, 1, 0), then the path will visit
                all nodes: randomly in xv-sections, with increasing z-coordinate first (most rapidly),
                and decreasing y-coordinate.
              - for simulation type SIM_VARIABLE_VECTOR, and u = (0, 2, 1), then the path will visit
                all nodes: randomly in x-direction, with increasing z-coordinate first (most rapidly),
                and increasing y-coordinate.
*/
PATH_RANDOM

/* DISTANCE THRESHOLD FOR EACH VARIABLE (as many number(s) as number of variable(s)) */
0.05

/* PROBABILITY CONSTRAINTS */
/* FOR EACH VARIABLE:
   1. Probability constraint usage, integer (probabilityConstraintUsage):
        - 0: no probability constraint
        - 1: global probability constraint
        - 2: local probability constraint

   2. If probabilityConstraintUsage > 0, then the classes of values (for which the
      probability constraints will be given) have to be defined; a class of values
      is given by a union of interval(s): [inf_1,sup_1[ U ... U [inf_n,sup_n[;
      Here are given:
         - nclass: number of classes of values
         - for i in 1,..., nclass: definition of the i-th class of values:
              - ninterval: number of interval(s)
              - inf_1 sup_1 ... inf_ninterval sup_ninterval: inf and sup for each interval
                   these values should satisfy inf_i < sup_i

   3a. If probabilityConstraintUsage == 1, then
        - global probability for each class (defined in 2. above), i.e.
          nclass numbers in [0,1] of sum 1
   3b. If probabilityConstraintUsage == 2, then
        - one pdf image file (for every class, nclass variables)
          (image of same dimensions as the simulation grid)
        - support radius for probability maps (i.e. distance according to
          the unit defined in the search neighborhood parameters for the
          considered variable)
        - method for computing the current pdf (in the simulation grid),
           integer (localCurrentPdfComputation):
             - 0: "COMPLETE" mode: all the informed node in the search neighborhood
                  for the considered variable, and within the support are taken into account
             - 1: "APPROXIMATE" mode: only the neighboring nodes (used for the
                  search in the TI) within the support are taken into account

   4. If probabilityConstraintUsage > 0, then
      method for comparing pdf's, integer (comparingPdfMethod):
         - 0: MAE (Mean Absolute Error)
         - 1: RMSE (Root Mean Squared Error)
         - 2: KLD (Kullback Leibler Divergence)
         - 3: JSD (JSD (Jensen-Shannon Divergence)
         - 4: MLikRsym (Mean Likelihood Ratio (over each class indicator, symmetric target interval))
         - 5: MLikRopt (Mean Likelihood Ratio (over each class indicator, optimal target interval))

   5. If probabilityConstraintUsage > 0, then
      - deactivation distance, i.e. one positive number
        (the probability constraint is deactivated if the distance between
        the current simulated node and the last node in its neighbors (used
        for the search in the TI) (distance computed according to the corresponding
        search neighborhood parameters) is below the given deactivation distance)
```

```
   6. If probabilityConstraintUsage > 0, then
         - threshold type for pdf's comparison, integer (probabilityConstraintThresholdType)
              - 0: constant threshold
              - 1: dynamic threshold
           note: if comparingPdfMethod is set to 4 or 5, probabilityConstraintThresholdType must be set to 0
         6.1a If probabilityConstraintThresholdType == 0, then
                 - threshold value
         6.1b If probabilityConstraintThresholdType == 1, then the 7 parameters:
                 - M1 M2 M3
                 - T1 T2 T3
                 - W
              These parameters should satisfy:
                 0 <= M1 <= M2 < M3,
                 T1 >= T2 >= T3,
                 w != 0.
              The threshold value t is defined as a function of the number M
              of nodes used for computing the current pdf (in the simulation grid)
              including the candidate (i.e. current simulated) node by:
                 t(M) = T1, if M < M1
                 t(M) = T2, if M1 <= M < M2
                 t(M) = (T3 - T2) / (M3^W - M2^W) * (M^W - M2^W) + T2, if M2 <= M < M3
                 t(M) = T3, if M3 <= M
*/
/* PROBABILITY CONSTRAINTS FOR VARIABLE #0 */
0

/* CONNECTIVITY CONSTRAINTS */
/* FOR EACH VARIABLE:
   1. Connectivity constraint usage, integer (connectivityConstraintUsage):
         - 0: no connectivity constraint
         - 1: set connecting paths before simulation by successively
              binding the nodes to be connected in a random order
         - 2: set connecting paths before simulation by successively
              binding the nodes to be connected beginning with
              the pair of nodes with the smallest distance and then
              the remaining nodes in increasing order according to
              their distance to the set of nodes already connected.
         - 3: check connectivity pattern during simulation
   2. If connectivityConstraintUsage > 0, then key word for type of connectivity:
         - CONNECT_FACE             : 6-neighbors connected (by face)
         - CONNECT_FACE_EDGE        : 18-neighbors connected (by face or edge)
         - CONNECT_FACE_EDGE_CORNER : 26-neigbhors connected (by face, edge or corner)
   3. If connectivityConstraintUsage > 0, then the classes of values (that can
         be considered in the same connected components) have to be defined; a
         class of values is given by a union of interval(s):
            [inf_1,sup_1[ U ... U [inf_n,sup_n[;
         Here are given:
            - nclass: number of classes of values
            - for i in 1,..., nclass: definition of the i-th class of values:
                 - ninterval: number of interval(s)
                 - inf_1 sup_1 ... inf_ninterval sup_ninterval: inf and sup for each interval
                      these values should satisfy inf_i < sup_i
   4. If connectivityConstraintUsage > 0, then:
         - variable name for connected component label
           (included in data image / data point set above)
           (Note: label negative or zero means no connectivity constraint)
   5a. If connectivityConstraintUsage == 1 or connectivityConstraintUsage == 2, then:
         - name of the image file and name of the variable for the search of
           connected paths (set string "_TI_" instead for searching in the (first)
           training image and the corresponding variable index)
   5b. If connectivityConstraintUsage == 3, then:
         - deactivation distance, i.e. one positive number
           (the connectivity constraint is deactivated if the distance between
           the current simulated node and the last node in its neighbors (used
           for the search in the TI) (distance computed according to the corresponding
           search neighborhood parameters) is below the given deactivation distance)
         - threshold: threshold value for acceptation of connectivity pattern
```

```
*/
/* CONNECTIVITY CONSTRAINTS FOR VARIABLE #0 */
0

/* BLOCK DATA */
/* FOR EACH VARIABLE:
   1. Block data usage, integer (blockDataUsage):
         - 0: no block data
         - 1: use of block data (mean value)

   2. If blockDataUsage == 1, then
         - block data file name
*/
/* BLOCK DATA FOR VARIABLE #0 */
0

/* MAXIMAL SCAN FRACTION FOR EACH TI (as many number(s) as number of training image(s)) */
0.25

/* PYRAMIDS */
/* I. PYRAMID GENERAL PARAMETERS:
      I.1. Number of pyramid level(s) (in addition to original simulation grid, i.e. number of
           reduction operations), integer (npyramidLevel):
              - = 0: no use of pyramids
              - > 0: use pyramids, npyramidLevel should be equal to the max of "nlevel" entries
                     in pyramid parameters for every variable (point II.1 below);
                     pyramid levels are indexed from fine to coarse:
                        * index 0            : original simulation grid
                        * index npyramidLevel: coarsest level
      If npyramidLevel > 0:
         I.2. for each level, i.e. for i = 1,..., npyramidLevel:
                 - kx, ky, kz (3 integer): reduction step along x,y,z-direction for the i-th reduction:
                      k[x|y|z] = 0: nothing is done, same dimension in output
                      k[x|y|z] = 1: same dimension in output (with weighted average over 3 nodes)
                      k[x|y|z] = 2: classical gaussian pyramid
                      k[x|y|z] > 2: generalized gaussian pyramid
         I.3. pyramid simulation mode, key work (pyramidSimulationMode):
                 - PYRAMID_SIM_HIERARCHICAL:
                      (a) spreading conditioning data through the pyramid by simulation at each
                          level, from fine to coarse, conditioned to the level one rank finer
                      (b) simulation at the coarsest level, then simulation of each level, from coarse
                          to fine, conditioned to the level one rank coarser
                 - PYRAMID_SIM_HIERARCHICAL_USING_EXPANSION:
                      (a) spreading conditioning data through the pyramid by simulation at each
                          level, from fine to coarse, conditioned to the level one rank finer
                      (b) simulation at the coarsest level, then simulation of each level, from coarse
                          to fine, conditioned to the gaussian expansion of the level one rank coarser
                 - PYRAMID_SIM_ALL_LEVEL_ONE_BY_ONE:
                      co-simulation of all levels, simulation done at one level at a time
         I.4. Factors to adapt the maximal number of neighboring nodes:
              I.4.1. Setting mode, key word (factorNneighboringNodeSettingMode):
                        - PYRAMID_ADAPTING_FACTOR_DEFAULT: set by default
                        - PYRAMID_ADAPTING_FACTOR_MANUAL : read in input
              If factorNneighboringNodeSettingMode == PYRAMID_ADAPTING_FACTOR_MANUAL:
                 I.4.2. The factors, depending on pyramid simulation mode:
                    - if pyramidSimulationMode == PYRAMID_SIM_HIERARCHICAL
                      or PYRAMID_SIM_HIERARCHICAL_USING_EXPANSION:
                         - faCond[0], faSim[0], fbCond[0], fbSim[0],
                           ...,
                           faCond[n-1], faSim[n-1], fbCond[n-1], fbSim[n-1],
                           fbSim[n]:
                              I.e. (4*n+1) positive numbers where n = npyramidLevel, with the following
                              meaning. The maximal number of neighboring nodes (according to each variable)
                              is multiplied by
                                (a) faCond[j] and faSim[j] for the conditioning level (level j)
                                    and the simulated level (level j+1) resp. during step (a) above
                                (b) fbCond[j] and fbSim[j] for the conditioning level (level j+1) (expanded
```

```
                                    if pyramidSimulationMode == PYRAMID_SIM_HIERARCHICAL_USING_EXPANSION)
                                    and the simulated level (level j) resp. during step (b) above
                    - if pyramidSimulationMode == PYRAMID_SIM_ALL_LEVEL_ONE_BY_ONE:
                         - f[0],...,f[npyramidLevel-1],f[npyramidLevel]:
                              I.e. (npyramidLevel + 1) positive numbers, with the following meaning. The
                              maximal number of neighboring nodes (according to each variable) is
                              multiplied by f[j] for the j-th pyramid level.
          I.5. Factors to adapt the distance threshold (similar to I.4):
               I.5.1. Setting mode, key word (factorDistanceThresholdSettingMode):
                         - PYRAMID_ADAPTING_FACTOR_DEFAULT: set by default
                         - PYRAMID_ADAPTING_FACTOR_MANUAL : read in input
               If factorDistanceThresholdSettingMode == PYRAMID_ADAPTING_FACTOR_MANUAL:
                  I.5.2. The factors, depending on pyramid simulation mode (similar to I.4.2).

          II. PYRAMID PARAMETERS FOR EACH VARIABLE:

          II.1. nlevel: number of pyramid level(s) (number of reduction operations)
                          - = 0: no use of pyramid for the considered variable
                          - > 0: use pyramids for the considered variable, with nlevel level

          If nlevel > 0:
             II.2. Pyramid type, key word (pyramidType):
                       - PYRAMID_CONTINUOUS        : pyramid applied to continuous variable (direct)
                       - PYRAMID_CATEGORICAL_AUTO  : pyramid for categorical variable
                                                     - pyramid for indicator variable of each category
                                                       except one (one pyramid per indicator variable)
                       - PYRAMID_CATEGORICAL_CUSTOM: pyramid for categorical variable
                                                     - pyramid for indicator variable of each class
                                                       of values given explicitly (one pyramid per
                                                       indicator variable)

             If pyramidType == PYRAMID_CATEGORICAL_CUSTOM:
                II.3.  The classes of values (for which the indicator variables are
                   considered for pyramids) have to be defined; a class of values is given by a union
                   of interval(s): [inf_1,sup_1[ U ... U [inf_n,sup_n[.
                   Here are given:
                      - nclass: number of classes of values
                      - for i in 1,..., nclass: definition of the i-th class of values:
                           - ninterval: number of interval(s)
                           - inf_1 sup_1 ... inf_ninterval sup_ninterval: inf and sup for each interval
                                these values should satisfy inf_i < sup_i
                   Then, for each class, the number of pyramid levels (number of reduction operations) is
                      - nlevel_i (for i in 1,..., nclass)
*/
/* PYRAMID GENERAL PARAMETERS */
0

/* TOLERANCE */
/* Tolerance t on the threshold value for flagging nodes (for post-processing):
   let d(i) be the distance between the data event in the simulation grid and in the training
   image for the i-th variable and s(i) be the distance threshold for the i-th variable, and let
   e(i) = max(0, (d(i)-s(i))/s(i)) be the relative error for the i-th variable, i.e. the relative
   part of the distance d(i) beyond the threshold s(i); during the scan of the training image, a node
   that minimizes e = sum (e(i)) is retained (the scan is stopped if e = 0); if e is greater than the
   tolerance t (given here), then the current simulated node and all non-conditioning nodes of the
   data events (one per variable) in the simulation grid are flagged for resimulation (post-processing).
   Note that if probability / connectivity / block data constraints used a similar error as e(i) that
   contributes in the sum defining the error e.
*/
0.0

/* POST-PROCESSING */
/* 1. Maximal number of path(s) (npostProcessingPathMax)
   2. If npostProcessingPathMax > 0:
      key word for post-processing parameters (i. e. number of neighboring nodes, distance threshold,
      maximal scan fraction, and tolerance):
         - POST_PROCESSING_PARAMETERS_DEFAULT: for default parameters
```


```
         - POST_PROCESSING_PARAMETERS_SAME   : for same parameters as given above
         - POST_PROCESSING_PARAMETERS_MANUAL : for manual settings
   3. If npostProcessingPathMax > 0 and POST_PROCESSING_PARAMETERS_MANUAL:
         MAXIMAL NUMBER OF NEIGHBORING NODES FOR EACH VARIABLE (as many number(s) as number of variable(s))
         MAXIMAL DENSITY OF NEIGHBORING NODES IN SEARCH NEIGHBORHOOD FOR EACH VARIABLE (as many number(s)
            as number of variable(s))
         DISTANCE THRESHOLD FOR EACH VARIABLE (as many number(s) as number of variable(s))
         MAXIMAL SCAN FRACTION FOR EACH TI (as many number(s) as number of training image(s))
         TOLERANCE
*/
1
POST_PROCESSING_PARAMETERS_DEFAULT

/* SEED NUMBER AND SEED INCREMENT */
444
1

/* NUMBER OF REALIZATION(S) */
1

END
```

References

D. Allard, D. D'Or, and R. Froidevaux. An efficient maximum entropy approach for categorical variable prediction. *EUROPEAN JOURNAL OF SOIL SCIENCE*, 62(3):381–393, JUN 2011. DOI: 10.1111/j.1365-2389.2011.01362.x.

G. Mariethoz, P. Renard, and J. Straubhaar. *A deterministic version of the multiple point geostatistics simulation / reconstruction method with the simulated / reconstructed values are directly taken from the training images without prior estimation of the conditional*, Publication number: US8682624 B2, Publication date: Mar 25, 2014, Application number: US 13/108,393, Applicant: University of Neuchâtel (CH), 2014.

G. Mariethoz, P. Renard, and J. Straubhaar. The Direct Sampling method to perform multiple-point geostatistical simulations. *WATER RESOURCES RESEARCH*, 46, NOV 20 2010. DOI: 10.1029/2008WR007621.

G. Mariethoz, J. Straubhaar, P. Renard, T. Chugunova, and P. Biver. Constraining distance-based multipoint simulations to proportions and trends. *ENVIRONMENTAL MODELLING & SOFTWARE*, 72:184–197, OCT 2015. DOI: 10.1016/j.envsoft.2015.07.007.

E. Meerschman, G. Pirot, G. Mariethoz, J. Straubhaar, M. Van Meirvenne, and P. Renard. A practical guide to performing multiple-point statistical simulations with the Direct Sampling algorithm. *COMPUTERS & GEOSCIENCES*, 52:307–324, MAR 2013. DOI: 10.1016/j.cageo.2012.09.019.

J. Straubhaar, P. Renard, and G. Mariethoz. Conditioning multiple-point statistics simulations to block data. *SPATIAL STATISTICS*, 16:53–71, MAY 2016. DOI: 10.1016/j.spasta.2016.02.005.

T. Zhang, P. Switzer, and A. Journel. Filter-based classification of training image patterns for spatial simulation. *MATHEMATICAL GEOLOGY*, 38(1):63–80, JAN 2006. DOI: 10.1007/s11004-005-9004-x.

To table of contents