USER GUIDE

For:

Bunch Pasting Direct Sampling Multiple-Point Geostatistical Simulation Algorithm Matlab Code®

Written by:

Hassan Rezaee, email: h.rezaee@ut.ac.ir

Thank you for your interest in Bunch-DS algorithm. I hope you enjoy it!

This is a manual for the Matlab (2012 version) program written with the purpose of simulation using the bunch-pasting multiple-point geostatistical simulation. Like DS original version, Bunch DS method works for both categorical and continuous training images. To use this program you may use the Matlab script named "*running_the_main_function.m"* which is to run the main algorithm. The main function is saved under the name of "*Bunch_DS.m"*®. There are some sub-functions embedded in *Bunch_DS* which will be called when needed throughout the code. What you need to do is to open the *running_the_main_function* program. In this script, you will see a set of parameters which originally are for the main function *Bunch_DS* but here to ease running the program. All the input parameters in this code are allocated with default values and codes which may be changed by users in due way. The parameters are explained once in the main function *Bunch_DS* and are recalled with more explanations in here. The main function of the *Bunch_DS* is provided below:

[Final_Realizations post_maps]= Bunch_DS(ti_name , Condition_mode , conditioning_weight , cond_data_name , sim_dim , Search_Radii , fract_of_ti_to_scan , dist_threshold , max_no_nodes_in_data_event , window_type , bunch_size , sim_path_mode , ti_path_mode , No_Realizations , post_proc_mode , simulation_show_mode);

INPUTS

ti_name: The *ti* of interest should be saved in a text file located in the directory of the main Matlab code. Write its name in single quotes to be readable by this program, for example *'TI_A'*. You may find three sample TIs (also used in the paper) in the code package in text files as well.

The *ti* file should be provided in the form *[x_coord y_coord z_coord variable]*. Note that the first growing coordinate is x, after which y comes along, and finally z. Look at sample file below:

1	1	1	15
2	1	1	45
3	1	1	36
1	2	1	45
2	2	1	70
3	2	1	93
1	3	1	129
2	3	1	151
3	3	1	168
1	1	2	187
2	1	2	204
3	1	2	207
1	2	2	205
2	2	2	203
3	2	2	195
1	3	2	180
2	3	2	152
3	3	2	137
1	1	3	118
2	1	3	90
3	1	3	62
1	2	3	34
2	2	3	18
3	2	3	14
1	3	3	58
2	3	3	93
3	3	3	109

Condition_mode: Consider the following codes to do:

0. non-conditional simulation

1. conditional simulation.

conditioning_weight: In conditional simulation mode, to help the algorithm honor the conditioning data better, these data can be given weights more than the previously simulated nodes which act as conditioning data at the same time. The default value is 10. Using very high values will honor the conditioning data at best but at the cost of losing structures.

cond_data_name: If the conditional simulation is selected then you should enter the name of conditioning data file. This should be a text file located in the directory of *Bunch_DS*. Conditioning data should be in the form *[x_coord y_coord z_coord variable]*. Example:

42	60	1	5
23	50	1	126
37	13	1	144
27	54	1	2
23	3	1	96
53	31	1	15
32	55	1	24
20	17	1	182
7	37	1	25
23	33	1	199
50	58	1	3

If non-conditional mode is selected then type anything else which carries no conditioning data while doing the simulation. Example = "test"

sim_dim: Number of simulation grid nodes along x, y and z directions, for example [10 15 1] is 2D grid of 10, 15 and 1 nodes along x, y and z directions.

Search_Radii: This should be in the form *[search_radius_x search_radius_y search_radius_x]*. Using different window types applies these search radii in different states. In the case of rectangular window type, these are the length, width and height (if 3D) of the search window. If disc shape window is selected, then:

outer_search_radius = max(***Search_Radii***) & *inner_search_radius* = min(***Search_Radii***).

and finally if circular window is selected then:

radius = min(***Search_Radii***).

Example: [8 8 8]

fract_of_ti_to_scan: The fraction of *ti* to be scanned to find the matching pattern; a number is to be chosen between [0-1]. Example = 0.3

dist_threshold: Distance threshold between N_x and N_y which is one of the stopping criteria of the simulation. Example = 0.05

max_no_nodes_in_data_event: Maximum number of nodes in data event which is formed by both primary conditioning and previously-simulated data. Obviously, applying larger numbers decrease the computational efficiency of the algorithm. Example = 40

window_type: Consider the following codes to specify the search window type:

1. circular

2. squared

3. disc Shape

Do not use disc shape since it produces noise and delivers garbage sometimes!

bunch_size: This value should be entered as an integer value: [0, 1, 2,…]. If ***bunch_size*** > 0 the simulation mode is through "*bunch*" pasting manner otherwise (***bunch_size*** = 0), pixel-based or traditional DS (Mariethoz, 2010) is selected. The maximum number of nodes in the bunch equals with $(2*\boldsymbol{bunch\_size}+1)^2$ in 2D and $(2*\boldsymbol{bunch\_size}+1)^3$ in 3D which is when the space around the node under simulation is completely empty. Example = 2

sim_path_mode: Use the following codes to specify the simulation path:

1. random

2. unilateral

ti_path_mode: Use the following codes to determine the way through which *ti* is scanned:

1. random

2. unilateral with a random starting point

No_Realizations: Number of realizations to be produced

post_proc_mode: Use the following codes to carry out post-processing calculations:

0. no post-processing step is taken after the simulation

1. E-type, Inter Quartile Range (IQR) and Quantile (probability 0.5) maps are calculated after the simulation.

simulation_show_mode: To illustrate the realization while simulation consider 1 otherwise any other code.

OUTPUTS

Final_Realizations: This is the final realizations file. It is in the form:

[x_coord y_coord z_coord realz#1 realz#1,… realz#n], where, *n*= ***No_Realizations***

post_maps: If post processing steps are specified to be taken after the simulation, the produced maps are saved too. This is in the form:

[x_coord y_coord z_coord E-type IQR Quantile]

Parfile: All the major input parameters are saved in a text file in the directory of the *Bunch_DS* under the name of "*Parfile.txt*". This is done for every time the function is run. You may change the name of the output file in the last line of *Bunch_DS* function.

Box below shows a sample output parfile of *Bucch_DS*:

TI name = TI_A

Conditioning mode = Conditional Simulation

conditioning weight = 10

Conditioning data name = TI_A_Cond

Simulation Grid Dimension (x,y) = 64 64 1

Search Radii (X , Y , Z) = 15 15 1

Fraction of ti which is scanned = 0.1

Distance type = Continuous variable

Distance threshold = 0.05

Maximum number of nodes in dataevent = 50

Window type = Circular

Bunch size = 25

Simulation path mode = Nodes are visited "Randomly"

The TI scanned through = Unilateral path with starting point

Number of realizations that are produced = 2

Post-processing maps = E-type, IQR and Quantile maps are produced"

Simulation time for one realization is (sec): 27.6785

Simulation time for all realizations is (sec): 56.8564

All parameters are explained enough to run the algorithm with ease. This algorithm can be easily changed into the original DS algorithm by considering ***bunch_size*** = 0 which pastes only one node at a time to the simulation grid.

SUB-FUNCTIONS

This program uses the following subroutines:

dlmcell.m: Writes cell array to text file (Roland Pfister, Version: 01.06.2010)

cond_dev_extract.m: Based on the window type this function extracts the data event from the neighborhood of the node under simulation within different shapes.

sim_post_proc.m: Calculates the post-processing maps of E-Type, Interquartile and Quantile maps.

All parts of the code are tried to be self-explained enough which is done by the adopting the meaningful names for the variables employed; however, extra explanations are associated in different steps of the code.

Authors would be glad to help you if any question raises regarding both the algorithm and the present Matlab code.

All the bests,

Hassan