s2Dcd

Release 0.1.0

Alessandro Comunian

Jan 21, 2020

CONTENTS:


s2Dcd

CHAPTER

ONE

CODE PURPOSE

`s2Dcd`:

`s` sequential

`2D` bi-dimensional (multiple-point statistics simulation with)

`cd` conditioning data

The code *s2Dcd* allows to apply the sequential 2D (multiple-point simulation) with conditioning data approach described in the paper by [Comunian2012]. For a list of publications where the *s2Dcd* was used, please see the section publications.

Note: At the moment the `s2Dcd` is it only a **wrapper** library: it requires an external MPS simulation engine to work.

Warning: Since 2017 only the direct sampling (DS) version of the `s2Dcd` is supported. Therefore, in the following documentation all references to the `Impala` MPS simulation engine should be considered outdated.

1.1 Not only `s2Dcd`...

The package can be used to do other things in addition to the application of the *s2Dcd* approach. For example, it contains some modules which are used as interface to the multiple-point simulation (MPS) codes *DeeSse* and *Impala*. Therefore, if for example you need to run a number of simulation tests with different parameters, you can use *Python* and the interface to the two codes to create customized benchmarks with the flexibility provided by *Python*. See the documentation provided in the appendices and the examples for more details.

Also, some simple functions that allows to read and write from *numpy* to *GSLIB* and *VTK* are provided.

If you have any questions, you want to contribute, suggest new features, you've found a bug... you can write me an e-mail: *alessandro DOT comunian AT gmail DOT com*

1.2 MPS simulation engines

As already mentioned, *s2Dcd* is only a wrapper library and requires an MPS simulation engine to work. This version contains and interface to the DeeSse MPS simulation engine (see this link for mode info.) Another simulation engine (*Impala*) was supported until 2017, but the current development of the interface to this engine is discontinued.

Nevertheless, users are encouraged to develop interfaces to their favorite MPS simulation engine (like for example SGeMS or MPSLib).

1.3 What's new in this version

- Removed the dependencies from the *VTK* libraries.
- Removed the dependencies from some additional modules derived by *Fortran90* subroutines.
- Solved some bugs.
- Added some new examples.

1.4 If you use `s2Dcd`

Please:

- Let us know!
- Cite the paper [Comunian2012]

1.5 Acknowledgments

Many thanks to Philippe Renard and Julien Straubhaar for their support and suggestions, to Andrea Borghi for the fruitful discussions and for finding some bugs, and to my brother Thomas for the design of the s2Dcd logo.

CHAPTER

TWO

INSTALLATION

All the modules included in the "package" are written in Python (version 3.X), which is platform independent. Therefore, the "package" should work on Linux, Mac OS X and Windows. Some tests were performed on all the three platforms, but at the moment the MPS engines implemented in the *s2Dcd* are available only for Linux and Windows.

2.1 Requirements

- Python (tested version 3.6.9 on Linux). Older versions should also work.
- If you are on Linux, you will also need the python development packages.
- The Python numerical library numpy and pandas and the Python package installer `pip`.
- The Multiple-point statistics simulation engines *DeeSse* and/or *Impala*.

Suggested software (optional)

Some software that you might find useful:

- ParaView, useful to visualize the *VTK* output of *Impala*.
- SGeMS, useful to visualize the *GSLIB* output of *DeeSse*.

2.1.1 Python, numpy and pandas installation

The "easy" way

If you do not want to install all the Python packages separately, there are some bundled distributions that allow to install all the required libraries in "one click". One of this distribution is Anaconda, which is available for MS Windows, Linux and Mac OS X (note however that this will require 200/300 Mbytes on your hard drive). Alternatively, you can use another "bundled distribution" or install each python package separately as explained in the following sections.

Warning: When you download Anaconda, select the Python *3.X* version and **NOT** the one for Python *2.X*.

Note: Personally, when working on Linux, I prefer to install Python packages using the OS package manager.

MS Windows

If you are on MS Windows, I strongly suggest you to use something like the aforementioned bundled distribution *Anaconda*.

Linux

Use you package manager, or `sudo apt-get install` to install `python3`, `python3-dev`, `python3-numpy` and `python3-pandas`. If *pandas* is not available for your distribution, please check on the last installation instructions here.

Mac OS X

Python and numpy are available for Mac OS X too. However, at the moment the binaries of the MPS simulation engines with an interface to the *s2Dcd* are not available for Mac OS X. You can still use some utility scripts to work with *GSLIB* files.

2.1.2 The multiple-point simulation engine *DeeSse*

Actually, the python module *s2Dcd* is only a wrapper for a MPS simulation engine. For the moment, the implementation allows to use the MPS simulation engine *DeeSse*.

The executable of your MPS simulation engine (in the case of the *DeeSse* they are called `deesse` and `deesseOMP`; add `.exe` if you are working on MS Windows), should be located in your `bin` directory, in the working directory or in a directory listed in the `PATH` environment variable.

Adapt the names of the MPS binaries

If the provided binaries have a different name...

The name of the binary of the simulation code can be changed using the variable *s2Dcd.mps_exes*, which is a Python dictionary.

To add a directory (i.e. `C:\Users\alex\my_dir` for MS Windows, or `/home/alex/my_dir`) to your path variable do:

MS Windows

1. *Start* > *Control Panel* > *System and Security* > *Advances System Settings*
2. Click on the button *Environment variables* at the bottom right.
3. Under *System Variables*, select the line containing the variable *Path*.
4. Edit the variable: add to the end of the variable value `;C:\Users\alex\my_dir\` (note the comma **;**)

Linux

Add the following line to your *.bash_aliases* file (eventually, create it and make sure that your *.bashrc* loads it):

```
export PATH=${PATH}:/home/alex/my_dir
```

Note: Before testing the *s2Dcd* module, check if the MPS simulation engines work correctly.

2.2 Installation of the module *s2Dcd* and other python tools

Download the sources from Github. From the command line (use the Anaconda prompt if you installed it), move into the directory `s2Dcd` (if needed, unpack it) and type the command:

```
pip install -e .
```

To check if the installation was successful, you can run the python shell and try import the main module:

```
>>> import s2Dcd.s2Dcd
```

2.3 Some details about the main modules:

s2Dcd.py the main module containing for the functions for the *s2Dcd* simulations.

deesse.py a simple interface to the parameters required by *DeeSse*.

gslibnumpy.py to convert from numpy and GSLIB and vice versa.

utili.py some simple utilities...

More details about these in section *Appendices*.

CHAPTER

THREE

EXAMPLES

Some usage examples of the `s2Dcd` module, including a *commented* example.

Warning: Some of the examples contained in the provided directory are very work in progress. If the directory related to an example contains a file `README.txt` where it is clearly stated that the example is **work in progress**, do not trust too much the corresponding example.

3.1 How to run the examples

3.1.1 MS Windows

Note: Here we suppose that you are using the default GUI interface ('IDLE') provided with Python. However, if you installed Python using *Anaconda*, there is a nicer user friendly GUI named *spyderlib* that you can use instead.

If you have installed *Python*, then you should have also the Python *IDLE* (a GUI interface). From the interface you can open one of the files in the `examples` directory, and from the menu of the *IDLE* select *run*. Another window should open and show you the advancement of the simulation.

Alternatively, you can use the *Command Prompt*, like this:

```
C:\Users\toto\examples\01_Strebelle> python.exe  s2Dcd_run-ex01.py
```

Note: If you are using *spyderlib* to run your scripts (and you don't have any Python installation other than the one that comes with the *Anaconda*), it can be useful, once you set the `PYTHONPATH` environment variable from the *spyderlib* menu (inside the *Python Path Manager* menu), to push the button *synchronize...* (and accept with *Yes*). This should allow you to use *s2Dcd* also with an external python shell. Then, to run the *s2Dcd*, go to *Run* > *Configuration per file...* and select *Execute in an external system terminal*, and tick *Interact with the python console after execution*. Then you should able to run the *s2Dcd* with these option without seeing plenty of command-prompt windows popping up... Note that you will probably need to restart the Python kernel.

3.1.2 Linux

In the directory of the each example, you can simply:

```
username@machine$ ./s2Dcd_run-ex01.py
```

You might need to provide the execution rights to your user (for example `chmod +x s2Dcd_run-ex01.py`).

If you want, you can redirect the output to some file and submit the process in background:

```
username@machine$ ./s2Dcd_run-ex01.py > somefile.out &
```

3.2 Commented example

This is a commented version of the file `s2Dcd_run-ex02.py` that you can find in the `..\examples\02_Strebelle-conditional` directory. It runs a simulation using two training images and some conditioning data, both in the form of data points and in the form of a conditioning slice. The example presents the use of the *DeeSse* MPS simulation engine, but the principles to run a simulation with a different MPS simulation engine (i.e. *Impala*) are similar.

Note: In python, all the text enclosed by `'''`, `"""` and the lines starting with # are comments.

In the first row we define the python interpreter... this is a standard command that can be included in each python script:

```
#!/usr/bin/env python3
```

Import some *standard* python modules:

```
import os
import time
import sys
import numpy as np
import random
import copy
```

Import the modules which are part of the *s2Dcd* software, that is:

```
import s2Dcd.s2Dcd as s2Dcd
```

a module that contains all the interface for the *DeeSse* MPS simulation engine:

```
import s2Dcd.deesse as mpds_interface
```

Note: If you want to use a different MPS engine *Impala*, you need to create your own module for this.

a simple module containing some utilities:

```
import s2Dcd.utili as utili
```

a module to read and write output in the GSLIB format:

```
import s2Dcd.gslibnumpy as gslibnumpy
```

If you can use a multi-thread version of *DeeSse*, you can define here the number of threads:

```
s2Dcd.nb_threads = 4
```

if you don't specify a value for this variable, the default value of 1 is used.

Then, the following line is used to print out some information about the run and record the start time:

```
time_start = utili.print_start()
```

A random seed can be set with the following command, in order to allow to get repeatable results:

```
seed = numpy.random.RandomState(456833)
```

Then you have to load the default definition for the parameters for the *DeeSse* simulation. In our case, we set all the parameters in a template file with in mind the resulting 3D simulation. Therefore, the grid size and the definition of the search template will be provided as we would do a 3D simulation. Here all the parameters are contained in a standard `.in` file for *DeeSse*. We therefore create a the template of parameters (`par_template`) from the file `template.in`:

```
template_in = "template.in"
par_template = mpds_interface.Param(file_name=template_in)
```

Note: You could also call `Param` without the argument `file_name`, and you will get the default parameters defined in the `mpds_interface`.

Now we define the parameters which are used for the simulation along the x, y and z axes. We always read the default parameters from the file defined by the variable `template_in`, but we could potentially select a different set of parameters for each direction (and each step of the sequence). Here we keep the parameters defined in the default `template.in` file. However, we customize them defining a different training image for each direction. The value `None` must be specified if we don't have a TI for the plane normal to that simulation direction:

```
par_Xnorm = mpds_interface.Param(file_name=template_in)
par_Xnorm.tis[0].file_name = None
par_Ynorm = mpds_interface.Param(file_name=template_in)
par_Ynorm.tis[0].file_name = "ti_250x1x250.gslib"
par_Znorm = mpds_interface.Param(file_name=template_in)
par_Znorm.tis[0].file_name = "ti_250x250x1.gslib"
```

Then you have to specify the maximum number of simulation step. You can usually put this value to a number bigger that the expected number of slices required to complete your simulation domain. The simulation will stop when the simulation domain will be filled.

However, in some cases when you would like to run only the fist step of the simulation, to check is the simulation and the conditioning are OK, you can set this value to a smaller integer, for example `6`. In this case we select a big value:

```
step_max = 3000
```

Now we define the simulation grid, which is extracted from the size of the grid contained in the file `template.in` and therefore in the variable `par_template`:

```
simODS = par_template.grid
```

the grid is printed to the standard output (for double check), together with other information about the simulation:

```
s2Dcd.print_sim_info(simODS, par_Xnorm, par_Ynorm, par_Znorm)
```

Then, it is required to define a numpy array which will contain the results of each step of the simulation. The default values of the array are initialized using the variable `s2Dcd.no_data`. The simulation will go on until all the *no_data* values will be simulated (or until the maximum simulation step `step_max`):

```
hard_data = s2Dcd.no_data * numpy.ones(
    (simODS.nx, simODS.ny, simODS.nz), 'float')
```

The default value for the variable `s2Dcd.no_data` is -1.

Note: You can redefine the value of the variable `s2Dcd.no_data` to suit different conversions for the no data values. In this example we consider the default value -1, therefore a definition is not required. If you want to change the no data value you can do something like:

```
s2Dcd.no_data = -999999
```

The conditioning data, if available, can be loaded in the GSLIB point data format using the following command:

```
s2Dcd.add_gslib_pointdata(["data_points.gslib"], hard_data, simODS)
```

Note: If you have many files containing conditioning data in the GSLIB point data format, then you can add them at the same time like this:

```
s2Dcd.add_gslib_pointdata(["file1.gslib",
    "file2.gslib","file3.gslib"], hard_data, simODS)
```

With this command the data points contained in the file `data_points.gslib` will be added to the numpy array that contains all the results. They will also be extracted at each simulation step if they are located in the simulated section. You can also add conditioning data in a structured grid format, for example if you have access to an outcrop or some data saved as a GSLIB "image". With the following command, we load the file `ti_250x1x250.gslib` and we associate its values to the section with *index* 4 along the *y* coordinate (the exact location depends in the definition of your grid...):

```
hd_section = gslibnumpy.gslib2numpy("ti_250x1x250.gslib")
hard_data[:,4,:] = hd_section[0:simODS.nx,0,0:simODS.nz]
```

Now we can create the simulation sequence. Basically, we alternatively simulate along the directions defined by the parameters defined by `par_Xnorm`, `par_Ynorm`, and `par_Znorm` only when a training image file is present. In this case we didn't provided a value for the TI file for the sections perpendicular to the plane *yz* (that is the parameters contained in `par_Xnorm`). Moreover, for a given direction we always use the same parameters along the duration of the simulation sequence. Therefore, to initialize the simulation sequence we only need:

```
seq = s2Dcd.create_seq(simODS, par_Xnorm, par_Ynorm, par_Znorm)
```

Then comes the true simulation step, which uses the information collected in the previous ones:

```
s2Dcd.sim_run(seq, step_max, hard_data, simODS, par_template, seed)
```

And finally, compute the running time (approximated!) and print *STOP*:

```
utili.print_stop(time_start)
```

3.3 Examples with *DeeSse*

3.3.1 Example 1: Strebelle's TI along two directions

file name *s2Dcd_run-ex01.py*

directory */examples/deesse/01_Strebelle*.

In this example, the *s2Dcd* method is applied to the celebrated Strebelle's 2D training image to obtain a 3D simulation. Here we use the same training images along the directions normal to the axis z and y.

No conditioning data are considered.

Note: Another possibility could be, for example, to use the two TI normal to axis x and y. In this case the hypothesis about the 3D geometry are

---

3.3.2 Example 2: Conditioning data

file name *s2Dcd_run-ex02.py*

directory */examples/02_Strebelle-conditional*.

Same as *Example 1*, but with conditioning points.

3.3.3 Example 3: Multiple realizations

file name *s2Dcd_run-ex03.py*

directory */examples/03_Strebelle-many-realizations*.

In this example, the same simulation framework of *Example 1* is applied many times in order to obtain different simulations with different random seeds.

3.3.4 Example 4: Changing the simulation parameters

file name *s2Dcd_run-ex01.py*

directory */examples/04_Strebelle-many-param*.

The modules related to the *s2Dcd* can be used also for other purposes, like for example to play with the parameters to be used for a MPS simulation in an automated way. You can for example use the modules to run different simulations changing automatically the size of the data template. Here we demonstrate this using the Strebelle's training image.

No conditioning data are considered.

CHAPTER

FOUR

NOTES

4.1 Auxiliary variables

The information provided in this section are useful to set up the template file and the *Python* script useful to run the `s2Dcd` using auxiliary variables.

4.1.1 Implementation in DeeSse

Note: This option might work properly, but it is in a early stage development.

The use of auxiliary variables with the *DeeSse* MPS engine could be implemented in a number of ways. Here the following strategy is selected:

1) The 2D slices (primary and auxiliary variable) are provided in the *same GSLIB* file using the variable name provided in the `template.in` file. In other words, only one TI is defined in the section `TRAINING IMAGE`.
2) The intermediate output files are provided as a unique file too. Therefore, in the section `OUTPUT SETTINGS FOR SIMULATION` in the `template.in` file the keyword `OUTPUT_SIM_ALL_IN_ONE_FILE` should be used.

4.1.2 Implementation in Impala

Warning: This MPS engine is not supported any more since 2017. This section is left just as a reference for further developments, potentially oriented towards other MPS simulation engines.

There are two main modes that can be implemented to use auxiliary variables:

Full 3D mode In this mode a full 3D auxliary variable map must be provided. Then the *s2Dcd* automatically slices it along the current simulated section to extract a 2D auxiliary variable map. This set up is useful when it is relatively easy to find a full 3D map for the auxiliary variable.

2D mode This should be the default mode, when you use the 2D maps of auxiliary variables attached to the 2D TIs provided for the simulation.

The *Full 3D mode* can be enabled by setting the `geostats.s2Dcd.AUX_VAR_FULL3D` parameter to `True`. Otherwise, the default value for the *2D mode* is used.

For the `Impala` MPS simulation engine the implementation of usage of the auxiliary variable is quite univoque and follows a quite standard workflow. Simply, the user have to provide (in addition to the 2D TIs of the main variable):

- a 2D auxiliary variable map (*VTK* files) with the same size of the provided 2D TIs, one for each TI.
- a 3D auxiliary variable map with the same size of the 3D simulation grid.

Note:

1) The format of the variable in the *VTK* files contaning the auxiliary variables should be `float`.
2) During the simulation process many *VTK* files containing sections of the 3D auxiliary variable map will be created. This is somehow redundant and in the future should be removed. For the moment you have to manually remove manually these files.
3) Very often the information contained in the 3D auxiliary variable sections is not very different from the 2D auxiliary variable corresponding to the 2D TI. Some computation time could be saved in these cases, but for the moment a complete 3D auxliary variable map is required to provide more flexibility.

4.2 Licence Issues

The recent versions of the DS codes are running with a licence manager. Therefore, it is possible that running the code will give an error code. At the moment the quick and dirty solution is to wait for some time and retry to run again the simulation of the same section. You can tune this in the module `deesse.py`, changing the values of the variables `NB_LIC_WAIT` (number of attempts to contact the lincese server) and `LIC_WAIT_TIME` (pause from one attempt to the other).

Warning: Note that the computing time can be heavily affected by this problem!

CHAPTER

FIVE

PUBLICATIONS

Some pubblications where the *s2Dcd* was used:

- A.Pickel, J.D.Frechette, **A.Comunian** and G.S.Weissmann (2015) "Building a Better Training Image with Digital Outcrop Models" - Journal of Hydrology 531(part 1) DOI: 10.1016/j.jhydrol.2015.08.049
- P.Bayer, **A.Comunian**, D.Höyng, & G.Mariethoz (2015) "High resolution multi-facies realizations of sedimentary reservoir and aquifer analogs" - Scientific Data, 2 DOI: 10.1038/sdata.2015.33 -
- T.C.Kessler, **A.Comunian**, F.Oriani, P.Renard, B.Nilsson, K.E.Klint and P.L.Bjerg (2013) "Modeling Fine-Scale Geological Heterogeneity - Examples of Sand Lenses in Tills." Groundwater 51(5) DOI: 10.1111/j.1745-6584.2012.01015.x
- **A.Comunian**, P.Renard and J.Straubhaar (2012) "3D multiple-point statistics simulation using 2D training images." Computers & Geosciences 40 DOI: 10.1016/j.cageo.2011.07.009 -
- N.Gueting, J.Caers, **A.Comunian**, J.Vanderborght, A.Englert, "Reconstruction of Three-Dimensional Aquifer Heterogeneity from Two-Dimensional Geophysical Data" - Mathematical Geosciences - DOI: 10.1007/s11004-017-9694-x -
- Q.Chen, G.Mariethoz, G.Liu, **A.Comunian**, and X.Ma "Locality-based 3-D multiple-point statistics reconstruction using 2-D geological cross-sections" - Hydrology and Earth System Sciences - DOI: 10.5194/hess-2018-256 -

CHAPTER

SIX

APPENDICES

Here some additional documentation about the *Python* modules used by the *s2Dcd* tool.

6.1 The `s2Dcd` module

license This file is part of s2Dcd.

s2Dcd is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

s2Dcd is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with s2Dcd. If not, see <https://www.gnu.org/licenses/>.

Purpose A module to apply the *s2Dcd* multiple-point simulation approach. At the moment it is implemented with the *DeeSse* MPS simulation engine, but could be adapted easily to other MPS engines.

File name `s2Dcd.py`

Version

0.9.9 , 2018-01-23 :

- Solved one bug that prevented simulation perpendicular to axis *x*.

0.9.8 , 2017-12-15 :

- Solved one but related to casting some index.

0.9.6 , 2014-08-12 :

- Try to include the auxiliary variables treatment.
- Changed name of the module related to the DS.

0.9.5 , 2013-11-15 :

- Converted to Python3 with *2to3*.
- Corrected a bug in the function *matrioska_interval*.
- Adapted to the new version of gslibnumpy.

0.9.4 , 2012-09-05 :

- Last version before the movement to Python3

0.9.3 , 2012-09-04 :

- Tested on a simple case study with one thread and no auxiliary variables.

0.9.2, 2012-05-02 :

- Defined some variables to describe the range of integer values that the seed should take.
- Corrected a bug related to bad parenthesis (thank you Andrea!)

0.9.1, 2012-05-02 :

- Added an option to better define when a problem is for categorical variables or for continuous ones.
- Solved a bug (forgotten to print the *in* file in the case of multiple threads).

0.8, 2012-04-02 :

- Modified some structures in order to include them in a module containing some tools to deal with structured grids... in order to prepare the interface to the MPDS code.

0.7, 2012-02-24 :

- Using the variable `no_data` the default value for the not yet simulated nodes can be redefined with more flexibility.

0.6, 2012-02-23 :

- Added a functionality allowing to define a random seed.
- Added a function to create a simple *standard* simulation sequence.

0.5, 2012-02-16 :

- added the class *SeqStep* to simplify notation.

0.4, 2012-02-16 :

- cleaned up some procedures and documentation.

0.3, :

- implemented the treatment of auxiliary variables.

0.2, :

- some improvements...

0.1, :

- first version.

Authors Alessandro Comunian.

6.1.1 Usage

See the examples in the corresponding directory and the documentation of the main functions.

6.1.2 Limitations

auxiliary variables: For the moment they are implemented only for the `implala` MPS engine.

number of variables: *MPDS* can be used with only one variable.

6.1.3 TODO

- Include the usage of auxiliary variables when using all the MPS engines.
- Implement all the features that can be described in the parameter files.
- Deal with the connection problem that can raise when the licence cannot be verified or the connection is slow.
- Improve the output and input format, adding for example the SGeMS binary input/output format (look for the format in mGstat, for example)
- For some simulation engines (like the one based on the snesim/impala engines), that can store information coming from threes/lists in and external file, it can be useful to allow the re-use of these.
- Add some functionality to delete all the output files created, something like *DEBUG* or *VERBOSE* mode.

`s2Dcd.s2Dcd.`**`GetWhereToAddData`**(*vtkReaderOutput*)

Get where the data file should be added into the hard data archive.

Parameters:

vtkReaderOutput: The result of a "GetOutput()" from a "vtkStructuredPointsReader" object.

Returns:

- The axis normal to the plane where the simulation was performed in the format character, that is *x*, *y* or *z*.
- the "coordinate" (in the reference system of the simulation, that is related to the "pixel" which have to be simulated in the matrix of hard data) of the slice where the new simulated data should be added.

> **Warning:** This function is obsolete.

class `s2Dcd.s2Dcd.`**`SeqStep`**(*direct*, *level*, *param*, *pseudo3D=0*)

A class that contains all the information required to create a parameters input file for the used MPS engines, that is a *step* of the simulation sequence of the approach *s2Dcd*.

`create_list`(*simODS*, *par_template*)

Run Impala to create the lists required along a given direction.

Parameters:

simODS: object of type `Grid` Contains all the info concerning the simulation domain.

par_template: class `impala_interface.Param`. The parameters for creating the *.in* file for the simulations.

Returns:

- Create for each level of multigrid a list in binary format.

`simul`(*step*, *step_max*, *hard_data*, *simODS*, *in_par*, *seed*, *rcp_lists*)

Run a MPS simulation for a simulation step of a sequence.

Parameters:

step: int The current simulation step.

step_max: int The maximum simulation step. Note that the simulation domain can be often filled before this step. In this case, the simulation procedure is stopped. Defining this parameter smaller than the total number of simulations expected to fill the domain can be useful for doing some preliminar test.

hard_data: numpy array A numpy array containig all the simulation. The value of the global variable `no_data` is used in locations not yet simulated.

simODS: object of type `Grid` Contains all the info concerning the simulation domain.

in_par: string or member of the class `Param`. The parameters for creating the *.in* file for the simulations. In case it is a string: The name of the file containing the template of the input parameters file. In case it is a *Param* object: All the parameters readed from a template *.in* file required for a MPDS simulation. seed: instance of the class `RandomState` To keep track of the random seed and create simulations that can be reproduced using the same seed.

rcp_lists: flag Useful if one wants to re-compute the list at each simulation step. This parameter has sense only for the *Impala* family of MPS simulation engines.

Returns:

- Create some files required for the simulation.
- Update the content of the numpy array *hard_data* including new simulated nodes.

If the selected section is full of hard data, simply returns 1 without creating files and without running the MPS simulation. If there is an error, returns -1.

`s2Dcd.s2Dcd.`**`adaptAuxVarFile`**(*seq*, *simODS*)

Change the dimension in the VTK file containing the auxiliary variable map in order to adapt it for the current 2D simulation.

Parameters:

seq: object of type *`SeqStep`* Contains the information related to the current simulation step.

simODS: object of type `Grid` Contains the dimension required to define a simulation domain.

Returns: A file containing a auxiliary variable whith dimensions suitable for the current simulation step.

`s2Dcd.s2Dcd.`**`addVtk2HdArchive`**(*hard_data*, *facies*)

A function to add all the VTK files contained into the current directory into an hard data archive file (that is a 3D matrix which once filled in will be the "final" simulation).

Parameters:

hard_data: string Name of the file which contains all the conditioning data.

facies: string A string containing the facies which are considered.

> **Warning:** This function is obsolete. It can be time consuming if a lot of VTK files are present in the working directory.

`s2Dcd.s2Dcd.`**`add_gslib_pointdata`**(*data_files*, *hard_data*, *simODS*)

Add the hard data contained in a number of GSLIB point data files.

Parameters:

files: list of strings A number of files containing the data. Note that this must be in a list format even if only one file is considered. For example, you should always use a syntax like *["file1.gslib"]* even if you provide only one file.

hard_data: 3D numpy array Where all the simulated points are stored.

simODS: object of type `Grid` Contains all the info concerning the simulation domain.

Returns: Update the content of the array *hard_data*.

`s2Dcd.s2Dcd.`**`add_hd`**(*seq_step*, *new_hd*, *hard_data*, *simODS*)

Add a numpy array containing the data simulated at a given sequence step to the array that stores the simulated nodes.

Parameters:

seq_step: instance of the class *`SeqStep`* Information about the current simulation step.

new_hd: numpy array (2D) The new simulated section.

hard_data: numpy array (3D) The contained of all the simulated data.

simODS: object of type `Grid` Information about the simulation grid.

Returns: Fit the content provided with *new_hd* into the right position into the array *hard_data*.

Note:

- A the moment the fact that a gslib file can contain multiple variables is handled with a quick and dirty trick.

`s2Dcd.s2Dcd.`**`all_segms0`**(*segms*)

Check if a list of segments contains at least one segment with length > 0.

Parameters:

segms: A list of tuples representing 1D segments.

Returns: True if all the segments contained into the list have length = 0, False if at least one segment has a length > 0.

Example

```
>>> all_segms0([(1,4), (2,7), (20,20)])
False
>>> all_segms0([(1,1), (7,7), (20,20)])
True
```

`s2Dcd.s2Dcd.`**`check_ti_file`**(*par*)

Check if a parameter file for *Impala* or *MPDS* has a name for the TI file or not.

Parameters: par: object of type *Param* or *Param*.

Returns: True if a TI file name is defines, false if None.

`s2Dcd.s2Dcd.`**`create_in_file4Impala`**(*in_par*, *seq_si*, *file_cond*, *file_name_sim*, *seed=None*)

Create a parameters input file for *Impala*. A big part of the information is read directly from the template input file for *Impala*.

Parameters:

in_par: object of type *impala_interface.Param*. All the information contained in the template file.

seq_si: object of the class *`SeqStep`*. All the informations required to create a simulation for the current simulation step. See the class `SeqStep` for details.

file_cond: string or None Name of the conditioning file, if None the simulation is considered as non conditional.

file_name_sim: string Name of the input paramter file for *Impala*.

seed: instance of the class `RandomState`, optional To keep track of the random seed and create simulations that can be reproduced using the same seed. A definition of the seed is not required when the MPS core is called only for the generation of the lists. Therefore, in this case the value of seed can be None, and the value of the module variable *seed_default* is used.

Returns: A **.in* file containing the parameters for running *Impala*.

Note: In order to increase the variability of the simulations, a new random seed is generated for each simulation file that is created if a `RandomState` instance is provided with the paramter *seed*. Otherwise, the value of the global variable *seed_default* is used.

`s2Dcd.s2Dcd.`**`create_in_file4MPDS`**(*in_par*, *seq_si*, *file_cond*, *file_name_sim*, *seed=None*)

Create a parameters input file for *MPDS* adapted to the current simulation step and the current conditioning data file.

Parameters:

in_par: object of type *ds_interface.Param*. All the info about the in template file.

seq_si: object of the class *`SeqStep`*. (sequence stepA info) All the informations required to create a simulation for the current simulation step. See the class *`SeqStep`* for details.

file_cond: string or None Name of the conditioning file, if None the simulation is considered as non conditional.

file_name_sim: string Name of the input paramter file for *MPDS*.

seed: instance of the class `RandomState`, optional To keep track of the random seed and create simulations that can be reproduced using the same seed. A definition of the seed is not required when the MPS core is called only for the generation of the lists. Therefore, in this case the value of seed can be None, and the value of the module variable *seed_default* is used.

Returns: A **.in* file for the current simulation.

Note: In order to increase the variability of the simulations, a new random seed is generated for each simulation file that is created if a `RandomState` instance is provided with the paramter *seed*. Otherwise, the value of the global variable *seed_default* is used.

`s2Dcd.s2Dcd.`**`create_lists`**(*simODS*, *par_template*, *par_Xnorm*, *par_Ynorm*, *par_Znorm*)

Create the lists which will be used by *Impala* for the simulation.

This preliminary step is required and allows use *Impala* to compute the MPS list only once, before starting the simulation, and therefore save computing resources.

Parameters:

simODS: object of type `Grid` Contains all the info concerning the simulation domain.

par_template: impala_interface.Param All the information contained in the *template.in* file.

par_Xnorm: object of type `impala_interface.Param` Information about the *Impala* parameters normal to the direction *x*.

par_Ynorm: object of type `impala_interface.Param` Information about the *Impala* parameters normal to the direction *y*.

par_Znorm: object of type `impala_interface.Param` Information about the *Impala* parameters normal to the direction *z*.

Returns: A list (in binary format) for each multi-grid level and for each simulation direction is created in the current directory running *Impala*.

s2Dcd.s2Dcd.**create_seq**(*simODS*, *par_Xnorm*, *par_Ynorm*, *par_Znorm*, *pseudo3D=0*)
Creates a simple simulation sequence.

Parameters:

simODS: grid definition Information about the simulation grid.

par_Xnorm, par_Ynorm, par_Znorm: `Param` Information related to the simulation directions. See the class documentation for details.

pseudo3D: integer (default=0) Set or not the "pseudo3D" simulation option when this value is >0.

Returns:

- A sequence of *SeqStep* objects, with all the information needed to run each simulation step.

Note: This function provide the **basic** definition for a simulation sequence when the simulation domain is quite simple, like for example when it has a "box" shape (the sizes of the simulation grid along the directions *x*, *y* and *z* are comparable). If the dimensions along the different axes of your simulation domain are not comparable, you can for example explicitly add some customized simulation steps to improve the quality of the results. There is one example of this in the examples directory.

---

s2Dcd.s2Dcd.**file_name_sec**(*file_name*, *axe*)
Return the name of a file which will be used to store a section of the input parameter *file_name*.

Parameters:

file_name: string The name of the input file which should be sectioned

axe: string in ('x','y','z') The axis perpendicular to the section.

Note: It is supposed that the sectioned file lies in the current directory. Therefore, the path in the name of the input file name will be dropped.

---

s2Dcd.s2Dcd.**matrioska_interval**(*points_nb*)
Computes a "spreaded" list of integers.

This function is useful to provide a simulation sequence along one axis that allows, in priciple, to obtain as much as intersections as possible along the other directions.

Parameters:

points_nb: integer The number of points contained in the list.

Returns: A list of integers containing a "matrioska sequence". If there is an error, returns -1.

Example: A typical output of the function is:

```
>>> matrioska_interval(8)
[0, 7, 3, 1, 5, 2, 4, 6]
```

Note:

- The output sequence always start from 0.
- The algorithm is probably not efficient, but for small number, that is for sizes of simulations (<1000), it can be OK. Moreover, it should be called few times (max 3).

s2Dcd.s2Dcd.**numpy2hd4Impala**(*simODS*, *hd*, *hd_file*, *seq=None*)

Given a numpy array, prints into an output file all the values in the hard data file format of *Impala*.

Parameters:

simODS: object of type `Grid` This object contains all the dimension required to define a simulation domain.

hd: numpy array A numpy array containing all the data which to be converted into hard data. The array contains `no_data` where there are not conditioning data.

hd_file: string Name of the file where the hard data will be printed.

seq: object type *`SeqStep`*, optional Some information about the current simulation step. If None, then all the *informed* content of the input numpy file is saved.

Returns: A file containing all the conditioning data required for a given simulation step in the *Impala* hard data format.

s2Dcd.s2Dcd.**numpy2hd4MPDS**(*simODS*, *hd*, *seq*)

Given a numpy array, prints into to a *xyz_data* file to be used by the function *gslibnumpy.numpy2gslib_points* to create a file with the conditioning points for the simulation for the MPS code *MPDS*.

Parameters:

simODS: object of type `Grid` This object contains all the dimension required to define a simulation domain.

hd: numpy array A numpy array containing all the data which to be converted into hard data. The array contains `no_data` where there are not conditioning data. seq: object type *`SeqStep`* Some information about the current simulation step.

Returns: A "xyz_data" file (a tuple of numpy arrays) containing the information about the coordinates and the data.

s2Dcd.s2Dcd.**print_result**(*hd*, *simODS*, *file_name*)

Print into an ouput file the simulation output.

Parameters:

hd: numpy array The numpy (3D) array containing all the simulated nodes.

simOSD: object of type `Grid` Contains the dimensions of the simulation domain.

file_name: string Name of the output file. The extension can be "vtk" or "gslib".

Returns: Create the output file. If some error occurs, returns -1.

s2Dcd.s2Dcd.**print_sim_info**(*simODS*, *par_Xnorm*, *par_Ynorm*, *par_Znorm*)

Print some information about the parameters of the simulation.

Parameters:

simODS: object of type `Grid` Information about the simulation grid.

par_Xnorm: object of type `Param` or `Param`. Information about the *Impala* parameters normal to the direction *x*. (idem for *y* and *z*) and for *MPDS*.

s2Dcd.s2Dcd.**sim_run**(*seq_steps*, *step_max*, *hard_data*, *simODS*, *in_par*, *seed*, *res_file_root='result'*, *rcp_lists=False*)

Run a *s2Dcd* simulation.

Parameters:

seq_steps: list object of type *`SeqStep`* A list containing all the information of each simulation step.

step_max: int The maximum simulation step. Note that the simulation domain can be often filled before this step.

hard_data: numpy array A numpy array containig all the simulation. The value `no_data` is used in location not simulated.

simODS: object of type `Grid` Contains all the info concerning the simulation domain.

in_par: string or a object of type *ds_interface.Param* The name of the file containing the template of the input parameters file for *Impala* or the object containing all the parameters required for a *MPDS* simulation.

seed: instance of the class `RandomState` To keep track of the random seed and create simulations that can be reproduced using the same seed.

res_file_root: string The root name of the file containing the output results. The right extension will be added once defined the MPS simulation core.

rcp_lists: flag (default False) This flag is useful if one wants to re-compute the list at each s2Dcd simulation step. When True it is a waste of time if the TI along the same direction is always the same. It is only useful when different training images are used for the simuations along the same direction.

Returns: A VTK or a GSLIB file containing the results of the simulation. If the max number of interation is reached, a result file containing "no data" values is printed out.

`s2Dcd.s2Dcd.`**`split_segms`**(*segms*, *mid_list=[]*)

A recursive function to print out a "matrioska-like" sequence of integers.

Parameters

segms [list of tuples of type (a,b)] A list containing a number of tuples representing a segment.

mid_list: list of integers, optional The list of integers that represents the required output. By default this should be called without this argument.

Returns A list containing all the integers in the mixed "matrioska" order.

Example

```
>>> matrioska_interval([(11,26)])
[11, 26, 18, 14, 22, 12, 16, 20, 24, 13, 15, 17, 19, 21, 23, 25]
```

Note: It is supposed that the coordinates of the segments are all integers.

6.2 The `s2Dcd.deesse` module

license This file is part of s2Dcd.

s2Dcd is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

s2Dcd is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with s2Dcd. If not, see <https://www.gnu.org/licenses/>.

Purpose A collection of classes and functions to interact with the Multiple Point Direct Sampling code (DeeSse).

File name `deesse.py`

Version

0.7 , 2020-01-20 :

- Some clean up before the upload to Github.

0.6 , 2017-12-15 :

- Adapted to the 2017 vesion of the code (includes "Pyramids" option)
- Partially takes the comments for the .in file from an external .json file.

0.5 , 2015-10-15 :

- Extending to deal with multiple variables simulation.

0.4 , 2013-06-17 :

- Some small modifications to make the setting of the default values more coherent.

0.3 , 2013-03-13 :

- Moved to Python3 using *2to3*.

0.2 , 2012-07-30 :

- Moved the function *skip_ccomments* to the module *utili.py*.
- Last version before the movement to Python3.

0.1 , 2012-03-30 : First version.

Authors Alessandro Comunian

Note:

- A *deesse_file_in_text.json* is required and contains all comment text that put in the deesse *.in* file. To create easily a *.json* file that respects the intendation and the new lines of the original *.in* file, you can use something like:

```
gawk '$1=$0' ORS='\n' file.in
```

to replace all the new lines by n, at the same time keeping the original indentation of the file.

Warning:

- The coverage and the support of the simulation options provided by DS simulation engine is only partial.

class s2Dcd.deesse.**Param**(*file_name=None, gridin=<s2Dcd.grid.Grid object>, var_nb=1, vars=[<s2Dcd.deesse.VarInfo object>], out_set='OUTPUT_SIM_ALL_IN_ONE_FILE', out_file='test_simul.gslib', out_report=True, out_report_file='test_report.txt', ti_nb=1, data_img_nb=0, data_img_files=[], data_pointset_nb=0, data_pointset_files=[], mask=False, homot_usage=0, rot_usage=0, cond_data_tol=0.05, norm_type='NORMALIZING_LINEAR', sim_type='SIM_ONE_BY_ONE', sim_path='PATH_RANDOM', tol=0.0, post_proc_path_max=1, post_proc_par='POST_PROCESSING_PARAMETERS_DEFAULT', pyramids=0, seed=444, seed_inc=1, real_nb=1*)

A class containing all the parameters contained in the parameters file.

print_blockdata()
Print info related to the BLOCK_DATA option

print_blockdata_intro()
Print some introductory text to the parameters related to the block data option

> **Warning:** Probability constraints are still not implemented in this version of the DeeSse interface.

print_consi()
Print info about the consistency

print_difile()
Print the content of the data image file

print_distthr()
Print info about the distance threshold

print_disttype()
Print info related to the distance type

print_dpset()
Print info related to the data point set

print_file(*file_name*)
Print the content of the parameters object into a file.

Parameters:

file_name: string Name of the file where to print.

print_homo()
Print info about the homothety

print_maskimage()
Print infor related to the mask image

print_max_dens()
Print info about the max density of nodes

print_max_nod()
Print the max number of neighborihg nodes

print_maxscan()
Print info about the percentage of image to scan

print_neigh_intro()
Print some introductory text to the parameters related to the search neighborhood

print_neighs()
Print info related to the search neighborhood

print_norm()
Print info about the normalization

print_out_report()
Print the output report

print_out_set()
Print the output settings

print_post()
Print info related to the post-processing procedure

print_proconst()
Print info related to the probability constraints

print_proconst_intro()
Print some introductory text to the parameters related to the probability constraints

> **Warning:** Probability constraints are still not implemented in this version of the DeeSse interface.

print_pyramids()
Print info related to the pyramids procedure

print_realnb()
Print the number of realizations

print_reldist()
Print info relative to the relative distance

print_rot()
Print info about the rotation

print_seed()
Print info about the seed

print_sim_grid()
Print the simulation grid info

print_sim_var()
Print the simulation variables info

print_simpath()
Print info about the simulation and the path

print_ti()
Print the TI info

print_tol()
Print info about the tolerance

print_weight()
Print info about the weighting factors

class s2Dcd.deesse.**TiInfo**(*file_name='ti.gslib'*, *max_scan=0.3*)
A class to support the class ParamMPDS. Contains all the information related to one training image.

Main attributes:

file_name: string Name of the file containing the training image

max_scan: float Fraction of the TI to scan

class s2Dcd.deesse.**VarInfo**(*name='facies'*, *out_flag=True*, *fmt='%10.5E'*, *search_r_x=120.0*, *search_r_y=120.0*, *search_r_z=0.0*, *search_anis_x=1.0*, *search_anis_y=1.0*, *search_anis_z=1.0*, *search_ang_az=0.0*, *search_ang_dp=0.0*, *search_ang_pl=0.0*, *search_pow=0.0*, *max_nb_neigh=20*, *max_dens_neigh=1.0*, *rel_dist_flag=0*, *dist_type=0*, *weight=1.0*, *dist_thre=0.01*, *prob_constr=0*, *block_data=0*)

A class to support the class ParamMPDS. It tries to regroup all the information about a variable which have to be simulated.

6.3 The s2Dcd.utili module

license This file is part of s2Dcd.

s2Dcd is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

s2Dcd is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with s2Dcd. If not, see <https://www.gnu.org/licenses/>.

this file *utili.py*

Purpose A collection of small utility functions

Version

- **0.6 , 2017-12-15 :**
  - Improved the way computing time is computed by using `time.perf_counter()`
- **0.5 , 2015-12-03 :**
  - Some minor cleanup
- **0.4 , 2013-03-06 :**
  - Converted to Python3.X
- **0.3 , 2013-03-06 :**
  - Last version before the conversion to python3.X
  - Added a function to skip the C-like comments.
- **0.2 , 2012-05-02 :**
  - Added a function usefult for the output print...
- **0.1 , 2012-03-09 :**
  - First version.

Authors Alessandro Comunian

s2Dcd.utili.**add_file_id**(*file_name*, *file_id*)

Add a file ID to a file name, given an int ID.

Parameters:

file_name: string The name of the file (with extension...)

file_id: int The ID which is associated to the file name

s2Dcd.utili.**dtype_fmt**(*x*)
Returns a printing format according to the type of the input numpy array.

Parameters: x: numpy array

Returns: A string that defines the output format that should be used for the formatted output, that is '%d' is the case of an integer, and '%.4e' in case of a float. In case of an error, returs -1.

s2Dcd.utili.**print_start**(*title=None*)
Print the "begin" header and start counting time...

Parameters:

title: string, optional A message to be printed in a "formatted stype". If a string is not provided, then use the name of the calling script.

Returns: The output of the function *time.clock()*.

s2Dcd.utili.**print_stop**(*time_start=None*)
Print the "end" message and the computing time.

Parameters:

time_start: ouput of the function *print_start()*, optional The function should be called in conjunction with the function *print_start*. If a value for time_start is not provided, then simply print out the current time and the stop message.

Returns: Print out the computing time

s2Dcd.utili.**progress_bar**(*i*, *nb_steps*)
A simple progress bar

Parameters:

i: integer The current index

nb_steps: The total number of steps

s2Dcd.utili.**progress_bar_end**()
Print the end of the progress bar

s2Dcd.utili.**savepdf**(*filename*, *title=''*)
Save a matplotlib figure as PDF and set up some useful information as metadata in the PDF file.

Parameters:

filename [string] The name of the PDF file.

title [string, optional] The title of the PDF document.

Returns: A PDF file containing the figure with the selected metadata.

s2Dcd.utili.**skip_ccomments**(*par_file*)
Read the connection to a file and extract a list containing all the fields, excluding line commenten in a *C-like* style, that is starting with "//" or enclosed by "/.../".

Parameters

file: file object The file to be read.

Returns A list containing all the strings (the parameters), separated by a space. Of course, the structure of the list depends on the structure of the paramters file.

Note: For the moment only the space characters are considered as field separators. All the fields in the list are returned as strings, therefore you will have to "cast" them according to your needs.

Warning: The comments at the end of a line containing parameters are not deleted. Therefore, used should be aware about what they are reading on each line...

6.4 The `s2Dcd.grid` module

license This file is part of s2Dcd.

s2Dcd is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

s2Dcd is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with s2Dcd. If not, see <https://www.gnu.org/licenses/>.

Purpose A module containing some classes and function useful to work with structured grids.

This file *grid.py*

Version

0.3 , 2013-08-11 :

- Adapted to the case of grids made of cells or grid made of points.

0.2 , 2013-03-06 :

- Converted to Python3.X with *2to3*

0.1 , 2012-04-02 :

- First version.

Authors Alessandro Comunian

Note: The grids for the moment are always considered as 3D.

---

class s2Dcd.grid.**Grid**(*ox=0.0*, *oy=0.0*, *oz=0.0*, *dx=1.0*, *dy=1.0*, *dz=1.0*, *nx=200*, *ny=200*, *nz=1*, *gtype='points'*)

A simple class that contains the *Origin*, *Delta* and *Size* of a simulation. It can also be used as container for some information contained in a VTK structured grid file.

Note:

- By default, the size of the grid is considered in term of points.

See also:

`vtknumpy`

property cells
Number of cells

compute_max()
Compute the max values for *x*, *y* and *z* of the grid.

get_cells()
Update the number of cells for a cells grid

get_center()
Returns the center of the grid

get_lx()
Provide as output a tuple containing the size of a grid. Useful for the implementation of `property`.

get_ly()
Provide as output a tuple containing the size of a grid. Useful for the implementation of `property`.

get_lz()
Provide as output a tuple containing the size of a grid. Useful for the implementation of `property`.

get_points()
Update the number of points for a points grid

get_size()
Compute the total number of points/cells in the grid.

property lx
'size' along *x* of the grid.

property ly
'size' along *y* of the grid.

property lz
'size' along *z* of the grid.

origin()
To print out the origin of the grid as a tuple

Parameters: self : an instance of the Grid class

Returns: A tuple containing the origin defined in the grid.

property points
Number of points

print4gslib()
A function to print out the grid information in the header of a GSLIB file.

print_intervals(*axis='xyz'*)
Print the intervals that constitute the simulation domain in a format like:

```
[ ox, ox+nx*dx] [ oy, oy+ny*dy] [ oz, oz+nz*dz]
```

where *ox* is the origin, *nx* is the number of points and *dx* is the delta between points (*idem* for *y* and *z*).

Parameters:

axis: string containing ['x','y','z'], optional If the default value "xyz" is used, then all the intevals are printed.

shape()

To print out the shape of the grid as a tuple

Parameters:

self:

an instance of the Grid class

gtype: string in ('points', 'cells')

String to decide to print the shape in terms of points or in terms of cells.

Returns: A tuple containing the origin defined in the grid.

spacing()

To print out the spacing of the grid as a tuple

Parameters: self : an instance of the Grid class

Returns: A tuple containing the spacing defined in the grid.

s2Dcd.grid.**sg_info**(*data*)

Collect some information about a structured grid contained in a numpy array

6.5 The s2Dcd.gslibnumpy module

license This file is part of s2Dcd.

s2Dcd is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

s2Dcd is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with s2Dcd. If not, see <https://www.gnu.org/licenses/>.

Purpose A module with some utilities to convert from numpy to gslib and vice-versa, and to support some GSLIB based software.

For more details about the GSLIB software, see [Deutsch1988] .

This file `gslibnumpy.py`

Version

0.9 , 2020-01-20 :

- Adapted to the inclusion in the s2Dcd package on Github.

0.8 , 2015-12-02 :

- Added *gslib_slice*.
- General clean-up
- Improved numpy2gslib to allow printing many variables

- Improved the pandas usage in numpy2gslib

0.7 , 2014-11-13 :

- Introduced dependencies from "pandas" to load txt faster

0.6 , 2013-08-26 :

- Improved how errors are handled in *gslib2numpy* and the extension available.

0.5 , 2013-07-07 :

- Improved the function *gslib_points2numpy*.

0.4 , 2013-03-06 :

- Converted to Python3 using *2to3*.
- Improved the flexibility of *gslib2numpy*.
- Remover the dependence from the module *gslibnumpy_f*.

0.3 , 2012-11-19 :

- Last version before the conversion to Python3.
- Module *gslib2numpy* improved to be able to read also files containing multiple variables.
- merged the function *dat2numpy* into *gslib2numpy*.

0.2 , 2012-11-18 :

- Some improvement using the suggestions of Spyder.
- Deleted the option for the "dtype" in function "gslib_points2numpy".
- included a new function to load GSLIB files coming from Isatis (dat2numpy, for files `.dat`)

0.1 , 2012-04-27 :

- Implemented in FORTRAN90 the subroutines *numpy2gslib*.

0.0 , 2012-03-07:

- First version.

Authors Alessandro Comunian

6.5.1 References

6.5.2 Functions and classes

class s2Dcd.gslibnumpy.**VarioStruct**(*stype=1, c_par=1.0, angles=[0.0, 0.0, 0.0], ranges=[100.0, 100.0, 100.0]*)

A class to contain info about one structure of a variogram

print4csv()

Print a string with the information contained in the class useful in a CSV file.

print4par()

Print a string with the information contained in the class useful in a parameter file.

class s2Dcd.gslibnumpy.**Variogram**(*str_nb=1*, *nugget=0.0*, *cat_thr=0*)

A class to contain all the parameters related to a GSLIB variogram model.

plot (*file_name*, *h_max=200.0*)
Plot the variogram model using matplotlib

Parameters:

h_max: float Maximum value of the lag

plot_3dir (*file_name*, *h_max=None*, *suptitle=None*)
Plot the variogram model using matplotlib along all the three direction for all the available ranges.

Parameters:

h_max: float Maximum value of the lag

suptitle: string A string for the super-title.

Note: Use by default the range of the 1st structure to define the upper limit of the x axis.

---

print4csv()
Print the info about a variogram for a CSV file

print4par()
Print the info about a variogram for a parameter file

s2Dcd.gslibnumpy.**align_parfile** (*par_file*)
Read a string as a parfile, detect where the character "" is and align everything.

Parameters:

par_file: string A string containing the parameter file to be indented.

Returns: A string containing the parameter file indented.

s2Dcd.gslibnumpy.**exponential** (*h*, *c=1.0*, *a=100.0*)
Definition of the model of a Exponential variogram.

Parameters:

h: float The values of lag h where to plot the variogram

c: float The sill value c

a: float The actual range a

Returns: The values of the variogram $\gamma(h)$

s2Dcd.gslibnumpy.**gaussian** (*h*, *c=1.0*, *a=100.0*)
Definition of the model of a Gaussian variogram.

Parameters:

h: float The values of lag h where to plot the variogram

c: float The sill value c

a: float The actual range a

Returns: The values of the variogram $\gamma(h)$

s2Dcd.gslibnumpy.**gslib2numpy** (*file_name*, *verbose=True*, *formats=None*, *names=None*, *source=None*, *grd=None*)
Convert a GSLIB ASCII grid into a numpy array.

Can convert only a subclass of GSLIB ASCII files. See note for more details.

Parameters:

file_name: string Name of the GSLIB input file.

verbose: bool, optional A flag to print reading information or not.

formats: list of characters, optional (default=None) The definition of the input format, `numpy.genfromtxt()` like. If nothing is provided, float values are considered for each variable.

names: tuple, optional (default=None) The definition of the names of the variables. If not provided, the names of the variables contained in the GSLIB file are used.

source: string in ['impala','isatis', 'fluvsim'], optional (default=None) The header changes a little if the file is to be used with "impala", or it is an output from "isatis". The functions tries to extrapolate from the file extension if the file comes from "impala" (.gslib) or from "Isatis" (.dat). If this argument is provided, then the format of the header is forced according to this (see the note for more details). The output files from FLUVSIM (.out) have a header that contains only the dimensions of the grid, and therefore they fall as a simple case of "implala" files.

Returns:

out_dict: A numpy dictionary of arrays with the data contained in the GSLIB file.

in_grd: A "grid.Grid" object, containing the grid definition (if provided).

Note: Limited to a subsclass of GSLIB ASCII files. The files can have an "impala" (of FLUVSIM) format, like:

```
nx ny nz [ dx dy dz [ ox oy oz] ]
nb_var
var_name[0]
var_name[1]
...
var_name[nb_var-1]
var0_value[0] var1_value[0] ...
var0_value[1] var1_value[1] ...
...           ...           ...
```

Or, in case the software is "Isatis", then the header will be like:

```
Description line
nb_var nx ny nz ox oy oz dx dy dz
...
```

See also:

`numpy.genfromtxt()`

`s2Dcd.gslibnumpy.`**`gslib2numpy_onevar`**(*file_name*, *verbose=True*, *formats=None*, *names=None*, *source=None*)

Convert a GSLIB ASCII grid into a numpy array.

This is a simplified version of `gslib2numpy` that can be used when you only have one variable and you don't care about the grid information contained in the file.

Can convert only a subclass of GSLIB ASCII files. See note for more details.

Parameters:

file_name: string Name of the GSLIB input file.

verbose: bool, optional A flag to print reading information or not.

formats: list of characters, optional (default=None) The definition of the input format, `numpy.genfromtxt()` like. If nothing is provided, float values are considered for each variable.

names: tuple, optional (default=None) The definition of the names of the variables. If not provided, the names of the variables contained in the GSLIB file are used.

source: string in ['impala','isatis'], optional (default=None) The header changes a little if the file is to be used with "impala", or it is an output from "isatis". The functions tries to extrapolate from the file extension if the file comes from "impala" (*.gslib) or from "Isatis" (*.dat). If this argument is provided, then the format of the header is forced according to this (see the note for more details).

Returns:

out_dict: A numpy dictionary of arrays with the data contained in the GSLIB file. NON NON .... ONLY A VARIABLE! UPDATE THIS DOCUMENTATION IF IT WORKS.

in_grd: A "grid.Grid" object, containing the grid definition (if provided).

Note: Limited to a subsclass of GSLIB ASCII files. The files can have an "impala" format, like:

```
nx ny nz [ dx dy dz [ ox oy oz] ]
nb_var
var_name[0]
var_name[1]
...
var_name[nb_var-1]
var0_value[0] var1_value[0] ...
var0_value[1] var1_value[1] ...
...           ...           ...
```

Or, in case the software is "Isatis", then the header will be like:

```
Description line
nb_var nx ny nz ox oy oz dx dy dz
...
```

See also:

`numpy.genfromtxt()`

`s2Dcd.gslibnumpy.`**`gslib_points2numpy`**(*file_name*, *verbose=True*, *formats=None*)

Convert to a GSLIB ASCII point data file into a numpy array.

Parameters:

file_name [string] Name of the GSLIB input file.

verbose [bool, optional] A flag to print out information or not.

formats: list of characters, optional (default=None) The definition of the input format, `numpy.genfromtxt()` like. If nothing is provided, then *genfromtxt* tries to understand the format automatically.

Returns: An *ndarray* containing a key for each variable contained in the GSLIB file and a string containing the header of the file.

Example: Read the content of the file *test.gslib*:

```
>>> data = gslib_points2numpy("test.gslib")
>>> x = data['x']
```

s2Dcd.gslibnumpy.**gslib_slice**(*file_in*, *axis*, *level=None*)

Cut a 2D slice within a 3D *GSLIB* file perpendicular to a given *axis* at a given *level*. The *GSLIB* file can contain more than one variable.

Parameters:

file_in [string,] The name of the input file.

axis [char, in ['x', 'y', 'z']] The axis coordinate which is used to cut the slice.

level [int, optional (default=None)] The index (**not** the true coordinate) along the coordinate where *axis* is defined where to cut the slice. If *None* then an index in the middle of the input domain is selected.

Returns: A file with the same name as *file_in* and a suffix -<*axis*><*level*>.*gslib* is created, and 0 if successful. If the return value is < 0 then there was some problem...

Warning: If in the target directory there is a file with the same name as the file created as return value this is overwritten.

s2Dcd.gslibnumpy.**numpy2dat**(*file_name*, *data*, *grd=None*, *varname=None*)

Convert a numpy array into a GSLIB ASCII file in the *Isatis* format.

Parameters:

file_name [string] The name of the GSLIB file where to save the data.

data [numpy array] The numpy array to be saved as GSLIB file.

grd [grid.Grid object, optional] The definition of the grid for the dataset.

varname [string, optional] The name of the variable to be stored in the GSLIB file.

Note:

- Works only with 1D, 2D or 3D numpy arrays.
- Only one variable per file.
- In some cases the input array must be flatten.

s2Dcd.gslibnumpy.**numpy2gslib**(*data*, *file_name*, *var_names=('data',)*, *grd=None*, *float_format=None*, *verbose=False*)

Convert a tuple of numpy arrays into a GSLIB ASCII file.

Parameters:

data [tuple of numpy arrays] The numpy arrays to be saved as GSLIB file.

file_name [string (optional)] The name of the GSLIB file where to save the data.

varname [tuple of string, optional] The names of the variable to be stored in the GSLIB file. (one for each numpy array in data).

grd [grid object, optional] If available, get the dimension of the grid from a grid object.

float_format: string (optional, default None) This is useful to pass to pandas *to_csv* the float format required.

verbose [boolean flag, optional] Print or not some info useful for debugging...

Note:

- Works only with 1D, 2D or 3D numpy arrays.
- When more than one variable is provided, the order provided in "var_names" is respected when printing the columns.
- *data* should be **always** provided as a tuple, even when it contains one array only, otherwise it may result in a corrupted file.

---

s2Dcd.gslibnumpy.**numpy2gslib_points**(*xyz_data*, *file_name*, *varname='data'*)
Convert a numpy array into a GSLIB points ASCII file.

Parameters:

xyz_data [tuple of numpy arrays] The tuple is made of four numpy arrays of the same lenght: the coordinates x, y, z and the point data *data*. Depending on the size of the tuple, the points are considered 1D, 2D or 3D.

file_name [string] The name of the GSLIB file where to save the data.

varname [string, optional] The name of the variable.

Returns: A gslib point file containing the coordinates of the points and the data values. If the size of the input array is <1, then the value -1 is returned and a warning message is printed out.

Note:

- The default names for the variables are *x*, *y*, *z* and *data*.
- The data type is automatically detected from the data type of the data.

---

s2Dcd.gslibnumpy.**power**(*h*, *c=1.0*, *omega=1.5*)
Definition of the model of a power variogram.

Parameters:

h: float The values of lag h where to plot the variogram

c: float The sill value c

omega: float The exponent of the model ω

Returns: The values of the variogram $\gamma(h)$

s2Dcd.gslibnumpy.**spherical**(*h*, *c=1.0*, *a=100.0*)
Definition of the model of a Spherical variogram.

Parameters:

h: float The values of lag h where to plot the variogram

c: float The sill value c

a: float The actual range a

Returns: The values of the variogram $\gamma(h)$

6.6 The `s2Dcd.ext` module

license This file is part of s2Dcd.

> s2Dcd is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> s2Dcd is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with s2Dcd. If not, see <https://www.gnu.org/licenses/>.

this_file *ext.py*

Purpose A library containing the definition for the extensions for a number of formats...

Version

- **0.0.2, 2013-02-26 :** Added some formats and improved the readibility.
- **0.0.1, 2012-05-15 :** The first version.

CHAPTER

SEVEN

LICENCE

s2Dcd, a Python library to interact with some Multiple Point Simulation codes and apply the sequential 2D simulation with conditioning data approach. Copyright (C) 2012-2018 Alessandro Comunian

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

For question, contributions, etc you can send me and e-mail: alessandro DOT comunian AT gmail DOT com

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Comunian2012] Comunian, A.; Renard, P. and Straubhaar, J. *3D multiple-point statistics simulation using 2D training images* Computers & Geosciences, 2012, **40**, 49-65. doi:10.1016/j.cageo.2011.07.009

[Deutsch1988] Deutsch, C. V. & Journel, A.G. (1988) GSLIB: Geostatistical Software Library and User's Guide 2nd edition. Oxford University Press

PYTHON MODULE INDEX

S

INDEX

S

T

V