iPerc

1.0

Generated by Doxygen 1.7.5.1

Thu Nov 13 2014 00:11:50

Contents

Chapter 1

iPerc Manual

Author

Yder Masson

Date

October 22, 2014

1.1 Introduction

iPerc is a software suite for modeling invasion percolation as introduced by Wilkinson and Willemsen in 1983 *(WILKINSON, David et WILLEMSEN, Jorge F. Invasion percolation: a new form of percolation theory. Journal of Physics A: Mathematical and General, 1983, vol. 16, no 14, p. 3365.)*. The code is written in Fortran 2003 and implement fast algorithms for simulating invasion percolation on arbitrary lattices. Both gravity and trapping can be modeled. This software explicitely model site percolation but it can also be used to model bond percolation. Some additional tools for generating random media and for visualization are also part of this package.

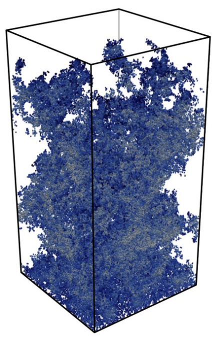

Figure 1.1: this is a caption for the image

1.2 Licence

1.3 References

If you use this code for your own research, please cite the following articles written by the developers of the package:

[1] MASSON, Yder et PRIDE, Steven R. A fast algorithm for invasion percolation. Transport in porous media, 2014, vol. 102, no 2, p. 301-312.

[2] MASSON, Yder et PRIDE, Steven R. A fast algorithm for invasion percolation Part II: Efficient a posteriory treatment of trapping. (To be published).

1.4 Installation

1.4.1 Prerequisites

∗ You should have received a copy of the source code: **iPerc.1.0.tar.gz**

∗ You must have the GNU Fortran compiler **gfortran** installed.

∗ For a better visual experience, you may want to install the ParaView visualisation software or any 3D plotter using the Visual toolkit (e.g. Check Mayavi if you are a Python addict). **(This is optional)**

Note

You can of course compile **iPerc** using another Fortran compiler. In this case, you have to edit the Makefile located in the **iPerc**/ directory. Replace **gfortran** with your compiler and make sure to change the compiling options accordingly.

See also

http://www.paraview.org/
http://mayavi.sourceforge.net/
http://en.wikipedia.org/wiki/Gfortran

1.4.2 started

Unzip the archive:

```
tar -zxvf iPerc.1.0.tar.gz
```

Move to the main directory:

```
{sh} cd iPerc/
```

Compile the source code:

```
make
```

This will compile the iPerc library as well as the examples in the **iPerc/examples/src/** directory and the projects in the **iPerc/my_projects/src/**. Then, you can try to run the examples in the **iPerc/examples/bin/** directory and the projects in the **iPerc/my_-projects/bin/**, for example type: ./examples/bin/the_example_name.exe

1.4.3 and compiling new projects

```
{.f90}
integer, dimension(:,:), allocatable i
```

```
{.py}
 class Python:
    pass
```

lkjdl jk lkj

Chapter 2

Data Type Index

2.1 Class List

Here are the data types with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Type Documentation

4.1 module_binary_tree Module Reference

Defines the binary tree data structure and the associated **add_branch** and **update_-root** procedures.

Public Member Functions

- subroutine add_branch (values, ind)

  Add a new site to the tree.
- subroutine update_tree_root (values)

  Update the root of the binary tree.
- subroutine swap (a, b)

  *exchange **a** and **b** values.*
- subroutine allocate_binary_tree (new_size)

  *Allocate memory for the **tree** array.*
- subroutine deallocate_binary_tree ()

  *Deallocate array **tree**.*

Public Attributes

- integer treedim

  Tree dimension (i.e. the number of sites or nodes in the tree, not the memory size)
- integer, dimension(:), allocatable tree

  *Tree array, each site with index **i** points toward his parent site with index **tree(i)***

4.1.1 Detailed Description

Defines the binary tree data structure and the associated **add_branch** and **update_-root** procedures.

Author

Yder MASSON

Date

October 7, 2014

See also

A fast algorithm for invasion percolation Y Masson, SR Pride - Transport in porous media, 2014 - Springer
http://link.springer.com/article/10.1007/s11242-014-0277-8
https://sites.google.com/site/ydermasson/

4.1.2 Member Function/Subroutine Documentation

4.1.2.1 subroutine module_binary_tree::add_branch (real, dimension(:) *values,* integer *ind*)

Add a new site to the tree.

Author

Yder Masson

Date

October 6, 2014

See also

Y Masson, SR Pride - Transport in porous media, 2014 - Springer
http://link.springer.com/article/10.1007/s11242-014-0277-8
https://sites.google.com/site/ydermasson/

Parameters

ind	Index of the site to be added to the tree
values	Array containing sites's values

4.1.2.2 subroutine module_binary_tree::allocate_binary_tree (integer *new_size*)

Allocate memory for the **tree** array.

Author

Yder Masson

Date

October 6, 2014

If the array is too small, use the intrinsic function **move_alloc** to increase the size of the **tree** array. To prevent reallocation, allocate enough memory in the first call.

Parameters

new_size	Desired dimension for the **tree** array.

4.1.2.3 subroutine module_binary_tree::deallocate_binary_tree ( )

Deallocate array **tree**.

Author

Yder Masson

Date

October 6, 2014

4.1.2.4 subroutine module_binary_tree::swap (integer *a*, integer *b*)

exchange **a** and **b** values.

Author

Yder Masson

Date

October 6, 2014

Parameters

a	input value 1
b	input value 2

4.1.2.5 subroutine module_binary_tree::update_tree_root (real, dimension(:) *values*)

Update the root of the binary tree.

Author

Yder Masson

Date

October 6, 2014

See also

Y Masson, SR Pride - Transport in porous media, 2014 - Springer
http://link.springer.com/article/10.1007/s11242-014-0277-8
https://sites.google.com/site/ydermasson/

Parameters

values	array containing the values atached to sites (tree nodes)

4.1.3 Member Data Documentation

4.1.3.1 integer, dimension(:), allocatable module_binary_tree::tree

Tree array, each site with index **i** points toward his parent site with index **tree(i)**

4.1.3.2 integer module_binary_tree::treedim

Tree dimension (i.e. the number of sites or nodes in the tree, not the memory size)

The documentation for this module was generated from the following file:

- modules/src/module_binary_tree.f90

4.2 module_cubic_indices Module Reference

This module contains functions to transform 3D indices to 1D index and vice versa.

Public Member Functions

- integer function ijk2ind (nx, ny, i, j, k)

  Transform 3D indices (i,j,k) to 1D index (ijk2ind)
- subroutine ind2ijk (nx, ny, ind, i, j, k)

  Transform 1D index (ijk2ind) to 3D indices (i,j,k)

4.2.1 Detailed Description

This module contains functions to transform 3D indices to 1D index and vice versa.

Author

Yder MASSON

Date

October 23, 2014

4.2.2 Member Function/Subroutine Documentation

4.2.2.1 integer function module_cubic_indices::ijk2ind (integer *nx*, integer *ny*, integer *i*, integer *j*, integer *k*)

Transform 3D indices (i,j,k) to 1D index (ijk2ind)

Parameters

nx	Grid dimension in the x direction (**input**)
ny	Grid dimension in the y direction (**input**)
i	i index, $1 < i <$ nx (**input**)
j	j index, $1 < j <$ ny (**input**)
k	k index, $1 < k <$ nz (**input**)

4.2.2.2 subroutine module_cubic_indices::ind2ijk (integer *nx*, integer *ny*, integer *ind*, integer *i*, integer *j*, integer *k*)

Transform 1D index (ijk2ind) to 3D indices (i,j,k)

Parameters

ind	1D index, $1 <$ ind $<$ nx $\times$ ny $\times$ nz
nx	Grid dimension in the x direction (**input**)
ny	Grid dimension in the y direction (**input**)
i	i index, $1 <$ i $<$ nx (**input**)
j	j index, $1 <$ j $<$ ny (**input**)
k	k index, $1 <$ k $<$ nz (**input**)

The documentation for this module was generated from the following file:

- modules/src/module_cubic_indices.f90

4.3 module_disjoint_set Module Reference

Define disjoint set data structure and the associated **Union**, **Find** and **Create_set** procedures.

Public Member Functions

- integer function create_set (largest_label)

  *Create a new label or class with label **largest_label**.*
- integer integer function find (label)

  *This function returns the canonical label (tree's root) or class associated with **label**.*
- integer function union (label1, label2)

  *Set **label1** and **label2** to equivalence classes (i.e. after the call **label1** and **label2** will point to the same canonical root label).*
- subroutine allocate_disjoint_set (new_size)

  *Allocate memory for the arrays **labels** and **ranks**.*
- subroutine deallocate_disjoint_set ()

Public Attributes

- integer, dimension(:), allocatable labels

  *Array containing the label trees, each label **i** points toward its parent **labels(i)** the root or canonical labels satisfies **labels(i)=i**.*
- integer, dimension(:), allocatable ranks

  Array containing the label trees's ranks. It is used for union by rank (also called weighted union)
- integer largest_label

  Number of labels currently used.

4.3.1 Detailed Description

Define disjoint set data structure and the associated **Union**, **Find** and **Create_set** procedures.

Author

Yder Masson

Date

October 6, 2014

Path compression and union by rank are implemented for efficiency. See subroutines's descriptions for more details.

See also

http://en.wikipedia.org/wiki/Disjoint-set_data_structure

4.3.2 Member Function/Subroutine Documentation

4.3.2.1 subroutine module_disjoint_set::allocate_disjoint_set (integer *new_size*)

Allocate memory for the arrays **labels** and **ranks**.

Author

Yder Masson

Date

October 6, 2014

When needed, use the intrinsic function **move_alloc** to increase the size of the arrays on the fly. To avoid reallocation, allocate enough memory at first call.

See also

https://gcc.gnu.org/onlinedocs/gfortran/MOVE_005fALLOC.html

Parameters

new_size	Desired dimension for : **ranks** and **labels** arrays.

4.3.2.2 integer function module_disjoint_set::create_set (integer *largest_label*)

Create a new label or class with label **largest_label**.

Author

Yder Masson

Date

October 6, 2014

This function allocates more memory to arrays **labels** and **ranks** if needed

See also

http://en.wikipedia.org/wiki/Disjoint-set_data_structure

Parameters

largest_label	Number of labels currently used

4.3.2.3 subroutine module_disjoint_set::deallocate_disjoint_set ( )

Author

Yder Masson

Date

October 6, 2014 Deallocate the **labels** and **ranks** arrays.

4.3.2.4 integer integer function module_disjoint_set::find (integer *label*)

This function returns the canonical label (tree's root) or class associated with **label**.

Author

Yder Masson

Date

October 6, 2014

Path compression is implemented for more efficiency.

See also

http://en.wikipedia.org/wiki/Disjoint-set_data_structure

Parameters

label	label for wich we search root or canonical label or class

4.3.2.5 integer function module_disjoint_set::union (integer *label1*, integer *label2*)

Set **label1** and **label2** to equivalence classes (i.e. after the call **label1** and **label2** will point to the same canonical root label).

Author

Yder Masson

Date

October 6, 2014

This function returns canonical label or class of the union. The union by rank alogorithm (also called weighted union) is implemented.

See also

http://en.wikipedia.org/wiki/Disjoint-set_data_structure

Parameters

label1	input label 1
label2	input label 2

4.3.3 Member Data Documentation

4.3.3.1 integer, dimension(:), allocatable module_disjoint_set::labels

Array containing the label trees, each label **i** points toward its parent **labels(i)** the root or canonical labels satisfies **labels(i)=i**.

4.3.3.2 integer module_disjoint_set::largest_label

Number of labels currently used.

4.3.3.3 integer, dimension(:), allocatable module_disjoint_set::ranks

Array containing the label trees's ranks. It is used for union by rank (also called weighted union)

The documentation for this module was generated from the following file:

- modules/src/module_disjoint_set.f90

4.4 module_gravity Module Reference

This module contains the functions to account for gravity.

Public Member Functions

- subroutine add_gravity_cubic_lattice (values, nx, ny, nz, dx, dy, dz, sigma, theta-_c, delta_rho, gx, gy, gz)

  Setup sites's invasion potential for cubic lattices.

- subroutine add_gravity_arbitrary_lattice (values, n, x, y, z, sigma, theta_c, delta-_rho, gx, gy, gz)

  Setup sites's invasion potential for arbitrary lattices.

4.4.1 Detailed Description

This module contains the functions to account for gravity.

Author

Yder MASSON

Date

October, 7 2014

The functions in this module compute the invasion potential $P_i = \frac{2\sigma \cos\theta_c}{a_i} - \Delta\rho g(L - z_i)$

Warning

Be careful when using periodic boundaries, gravity gets periodic as well...

4.4.2 Member Function/Subroutine Documentation

4.4.2.1 subroutine module_gravity::add_gravity_arbitrary_lattice (real, dimension(:) *values*, integer *n*, real, dimension(:) *x*, real, dimension(:) *y*, real, dimension(:) *z*, real *sigma*, real *theta_c*, real *delta_rho*, real *gx*, real *gy*, real *gz*)

Setup sites's invasion potential for arbitrary lattices.

Author

Yder MASSON

Date

October, 7 2014

The invasion potential is defined as: $P_i = \frac{2\sigma \cos\theta_c}{a_i} - \Delta\rho g(L - z_i)$

Warning

Be careful when using periodic boundaries, gravity gets periodic as well...

Parameters

n	Total number of sites in the lattice (**input**)
x	Array containing the x coordinates od the sites (**input**)
y	Array containing the y coordinates od the sites (**input**)
z	Array containing the z coordinates od the sites (**input**)
values	Output array containig the sites's invasion potentials (**input/output)** At input time, this array contains the pores's sizes a_i At output time, this array contains the invasion potential $P_i = \frac{2\sigma\cos\theta_c}{a_i} - \Delta\rho g(L-z_i)$
sigma	Surface tension σ (**input**)
theta_c	Equilibrium contact angle θ_c (**input**)
delta_rho	Fluid density contrast $\Delta\rho$ (**input**)
gx	Acceleration of gravity g in the x direction (**input**)
gy	Acceleration of gravity g in the y direction (**input**)
gz	Acceleration of gravity g in the z direction (**input**)

4.4.2.2 subroutine module_gravity::add_gravity_cubic_lattice (real, dimension(nx,ny,nz) *values*, integer *nx*, integer *ny*, integer *nz*, real *dx*, real *dy*, real *dz*, real *sigma*, real *theta_c*, real *delta_rho*, real *gx*, real *gy*, real *gz*)

Setup sites's invasion potential for cubic lattices.

Author

Yder MASSON

Date

October, 7 2014

The invasion potential is defined as: $P_i = \frac{2\sigma\cos\theta_c}{a_i} - \Delta\rho g(L-z_i)$

Warning

Be careful when using periodic boundaries, gravity gets periodic as well...

Parameters

nx	Grid dimension in the x direction (**input**)
ny	Grid dimension in the y direction (**input**)
nz	Grid dimension in the z direction (**input**)
dx	Grid spacing in the x direction Δx (**input**)
dy	Grid spacing in the x direction Δy (**input**)
dz	Grid spacing in the x direction Δz (**input**)
values	Output array containig the sites's invasion potentials (**input/output)** At input time, this array contains the pores's sizes a_i At output time, this array contains the invasion potential $P_i = \frac{2\sigma\cos\theta_c}{a_i} - \Delta\rho g(L-z_i)$

sigma	Surface tension σ (**input**)
theta_c	Equilibrium contact angle θ_c (**input**)
delta_rho	Fluid density contrast $\Delta\rho$ (**input**)
gx	Acceleration of gravity g in the x direction (**input**)
gy	Acceleration of gravity g in the y direction (**input**)
gz	Acceleration of gravity g in the z direction (**input**)

The documentation for this module was generated from the following file:

- modules/src/module_gravity.f90

4.5 module_interpolation Module Reference

Public Member Functions

- real function trilinear_iterpolation (mat, nx, ny, nz, xmin, xmax, ymin, ymax, zmin, zmax, x, y, z)
- real function bilinear_interpolation (mat, nx, ny, xmin, xmax, ymin, ymax, x, y)
- real function linear_interpolation (mat, nx, xmin, xmax, x)

4.5.1 Member Function/Subroutine Documentation

4.5.1.1 real function module_interpolation::bilinear_interpolation (real, dimension(0:nx-1,0:ny-1) *mat*, integer *nx*, integer *ny*, real *xmin*, real *xmax*, real *ymin*, real *ymax*, real *x*, real *y*)

Parameters

mat	2D matrix of real values
nx	Number of grid points in the x direction
ny	Number of grid points in the y direction
xmin	Lower bound of grid extent in the x direction
xmax	Upper bound of grid extent in the x direction
ymin	Lower bound of grid extent in the y direction
ymax	Upper bound of grid extent in the y direction
x	x coordinate of the interpolation point
y	y coordinate of the interpolation point

4.5.1.2 real function module_interpolation::linear_interpolation (real, dimension(0:nx-1) *mat*, integer *nx*, real *xmin*, real *xmax*, real *x*)

Parameters

mat	1D vector of real values
nx	Number of grid points
xmin	Lower bound of grid extent

xmax	Upper bound of grid extent
x	x coordinate of the interpolation point

4.5.1.3 real function module_interpolation::trilinear_iterpolation (real, dimension(0:nx-1,0:ny-1,0:nz-1) *mat,* integer *nx,* integer *ny,* integer *nz,* real *xmin,* real *xmax,* real *ymin,* real *ymax,* real *zmin,* real *zmax,* real *x,* real *y,* real *z*)

Parameters

mat	3D matrix of real values
nx	grid dimension in the x direction
ny	grid dimension in the y direction
nz	grid dimension in the z direction
xmin	Lower bound of grid extent in the x direction
xmax	Upper bound of grid extent in the x direction
ymin	Lower bound of grid extent in the y direction
ymax	Upper bound of grid extent in the y direction
zmin	Lower bound of grid extent in the z direction
zmax	Upper bound of grid extent in the z direction
x	x coordinate of the interpolation point
y	y coordinate of the interpolation point
z	z coordinate of the interpolation point

The documentation for this module was generated from the following file:

- modules/src/module_interpolation.f90

4.6 module_invasion_percolation Module Reference

Public Member Functions

- subroutine invade_cubic_lattice_simple (nx, ny, nz, dx, dy, dz, period_x, period_y, period_z, values, states, n_sites_invaded, invasion_list, gravity, trapping, sigma, theta_c, delta_rho, gx, gy, gz)

  A simple but inefficient implementation of invasion percolation on a 3D cubic lattice.
- subroutine invade_cubic_lattice_fast (nx, ny, nz, dx, dy, dz, period_x, period_y, period_z, values, states, n_sites_invaded, invasion_list, gravity, trapping, sigma, theta_c, delta_rho, gx, gy, gz)

  An efficient implementation of invasion percolation on a 3D cubic lattice.
- subroutine invade_arbitrary_lattice_simple (n_sites, x, y, z, offsets, connectivity, values, states, n_sites_invaded, invasion_list, gravity, trapping, sigma, theta_c, delta_rho, gx, gy, gz)

  A simple but inefficient implementation of invasion percolation on arbitrary lattices.

- subroutine invade_arbitrary_lattice_fast (n_sites, x, y, z, offsets, connectivity, values, states, n_sites_invaded, invasion_list, gravity, trapping, sigma, theta_c, delta_rho, gx, gy, gz)

    An efficient implementation of invasion percolation on arbitrary lattices.

4.6.1 Member Function/Subroutine Documentation

4.6.1.1 subroutine module_invasion_percolation::invade_arbitrary_lattice_fast (integer *n_sites*, real, dimension(:) *x*, real, dimension(:) *y*, real, dimension(:) *z*, integer, dimension(:) *offsets*, integer, dimension(:) *connectivity*, real, dimension(:) *values*, integer, dimension(:) *states*, integer *n_sites_invaded*, integer, dimension(:) *invasion_list*, logical *gravity*, logical *trapping*, real *sigma*, real *theta_c*, real *delta_rho*, real *gx*, real *gy*, real *gz*)

An efficient implementation of invasion percolation on arbitrary lattices.

Author

Yder MASSON

Date

October 21, 2014

Parameters

n_sites	Total number of sites in the lattice (**Input**)
x	Array containing the x coordinates of the sites (**Input**)
y	Array containing the y coordinates of the sites (**Input**)
z	Array containing the z coordinates of the sites (**Input**)
offsets	Array containing the offsets of the data stored in the connectivity array (**Input**)
connectivity	Array containing thelattice connectivity (**input**) For a given site **i**, with offset **j = offsets(i)** : **n=connectivity(j)** is the number of sites neighboring site **i**. **connectivity(j+1, j+2, ... ,j+n)** contains the indices of the sites that are neighboring site **i**.
values	Array containing the sites's sizes a_i (**input/output)** When **gravity==.true.** this array is modified and contains the invasion potentials P_i at output time
states	Array containing the sites's states (**input/output)** Set **state(i)=neighboring** at sites **i** where you would like to inject the fluid Set **state(i)=exit_site** at sites **i** where the defending fluid can escape (the simulation will stop when the invading fluid percolates, i.e. it reaches one of these sites) Set **state(i)=sealed** to prevent sites **i** from being invaded At output time, we have **state(i)=trapped** at trapped sites At output time, we have **state(i)=invaded** at invaded sites

n_sites_-invaded	Number of sites invaded (**output**)
invasion_list	Array containing the list of sites invaded sorted in chronological order (**output**)
gravity	Flag for gravity (**input**) Set **gravity=.true.** to account for gravity Set **gravity=.false.** to ignore gravity
trapping	Flag for trapping (**input**) Set **trapping=.true.** to account for trapping Set **trapping=.false.** to ignore trapping
sigma	Surface tension σ (**input**)
theta_c	Equilibrium contact angle θ_c (**input**)
delta_rho	Fluid density contrast $\Delta\rho$ (**input**)
gx	Acceleration of gravity g in the x direction (**input**)
gy	Acceleration of gravity g in the y direction (**input**)
gz	Acceleration of gravity g in the z direction (**input**)

4.6.1.2 subroutine module_invasion_percolation::invade_arbitrary_lattice_simple (integer *n_sites,* real, dimension(:) *x,* real, dimension(:) *y,* real, dimension(:) *z,* integer, dimension(:) *offsets,* integer, dimension(:) *connectivity,* real, dimension(:) *values,* integer, dimension(:) *states,* integer *n_sites_invaded,* integer, dimension(:) *invasion_list,* logical *gravity,* logical *trapping,* real *sigma,* real *theta_c,* real *delta_rho,* real *gx,* real *gy,* real *gz*)

A simple but inefficient implementation of invasion percolation on arbitrary lattices.

Author

Yder MASSON

Date

October 21, 2014

Parameters

n_sites	Total number of sites in the lattice (**Input**)
x	Array containing the x coordinates of the sites (**Input**)
y	Array containing the y coordinates of the sites (**Input**)
z	Array containing the z coordinates of the sites (**Input**)
offsets	Array containing the offsets of the data stored in the connectivity array (**Input**)
connectivity	Array containing thelattice connectivity (**input**) For a given site **i**, with offset **j = offsets(i)** : **n=connectivity(j)** is the number of sites neighboring site **i**. **connectivity(j+1, j+2, ... ,j+n)** contains the indices of the sites that are neighboring site **i**.

values	Array containing the sites's sizes a_i (**input/output)** When **gravity==.true.** this array is modified and contains the invasion potentials P_i at output time
states	Array containing the sites's states (**input/output)** Set **state(i)=neighboring** at sites **i** where you would like to inject the fluid Set **state(i)=exit_site** at sites **i** where the defending fluid can escape (the simulation will stop when the invading fluid percolates, i.e. it reaches one of these sites) Set **state(i)=sealed** to prevent sites **i** from being invaded At output time, we have **state(i)=trapped** at trapped sites At output time, we have **state(i)=invaded** at invaded sites
n_sites_-invaded	Number of sites invaded (**output**)
invasion_list	Array containing the list of sites invaded sorted in chronological order (**output**)
gravity	Flag for gravity (**input**) Set **gravity=.true.** to account for gravity Set **gravity=.false.** to ignore gravity
trapping	Flag for trapping (**input**) Set **trapping=.true.** to account for trapping Set **trapping=.false.** to ignore trapping
sigma	Surface tension σ (**input**)
theta_c	Equilibrium contact angle θ_c (**input**)
delta_rho	Fluid density contrast $\Delta\rho$ (**input**)
gx	Acceleration of gravity g in the x direction (**input**)
gy	Acceleration of gravity g in the y direction (**input**)
gz	Acceleration of gravity g in the z direction (**input**)

4.6.1.3 subroutine module_invasion_percolation::invade_cubic_lattice_fast (integer *nx*, integer *ny*, integer *nz*, real *dx*, real *dy*, real *dz*, logical *period_x*, logical *period_y*, logical *period_z*, real, dimension(nx∗ny∗nz) *values*, integer, dimension(nx,ny,nz) *states*, integer *n_sites_invaded*, integer, dimension(:) *invasion_list*, logical *gravity*, logical *trapping*, real *sigma*, real *theta_c*, real *delta_rho*, real *gx*, real *gy*, real *gz*)

An efficient implementation of invasion percolation on a 3D cubic lattice.

Author

Yder MASSON

Date

October 21, 2014

Parameters

nx	Grid dimension in the x direction (**input**)
ny	Grid dimension in the y direction (**input**)
nz	Grid dimension in the z direction (**input**)
dx	Grid spacing in the x direction Δx (**input**)
dy	Grid spacing in the x direction Δy (**input**)
dz	Grid spacing in the x direction Δz (**input**)
period_x	Flag for periodic boundaries in the x direction (**input**)
period_y	Flag for periodic boundaries in the y direction (**input**)
period_z	Flag for periodic boundaries in the z direction (**input**)
values	Array containing the sites's sizes a_i (**input**/**output)** When **gravity==.true.** this array is modified and contains the invasion potentials P_i at output time
states	Array containing the sites's states (**input**/**output)** Set **state(i)=neighboring** at sites **i** where you would like to inject the fluid Set **state(i)=exit_site** at sites **i** where the defending fluid can escape (the simulation will stop when the invading fluid percolates, i.e. it reaches one of these sites) Set **state(i)=sealed** to prevent sites **i** from being invaded At output time, we have **state(i)=trapped** at trapped sites At output time, we have **state(i)=invaded** at invaded sites
n_sites_-invaded	Number of sites invaded (**output**)
invasion_list	Array containing the list of sites invaded sorted in chronological order (**output**)
gravity	Flag for gravity (**input**) Set **gravity=.true.** to account for gravity Set **gravity=.false.** to ignore gravity
trapping	Flag for trapping (**input**) Set **trapping=.true.** to account for trapping Set **trapping=.false.** to ignore trapping
sigma	Surface tension σ (**input**)
theta_c	Equilibrium contact angle θ_c (**input**)
delta_rho	Fluid density contrast $\Delta\rho$ (**input**)
gx	Acceleration of gravity g in the x direction (**input**)
gy	Acceleration of gravity g in the y direction (**input**)
gz	Acceleration of gravity g in the z direction (**input**)

4.6.1.4 subroutine module_invasion_percolation::invade_cubic_lattice_simple (integer *nx*, integer *ny*, integer *nz*, real *dx*, real *dy*, real *dz*, logical *period_x*, logical *period_y*, logical *period_z*, real, dimension(nx,ny,nz) *values*, integer, dimension(nx,ny,nz) *states*, integer *n_sites_invaded*, integer, dimension(:) *invasion_list*, logical *gravity*, logical *trapping*, real *sigma*, real *theta_c*, real *delta_rho*, real *gx*, real *gy*, real *gz*)

A simple but inefficient implementation of invasion percolation on a 3D cubic lattice.

Author

Yder MASSON

Date

October 21, 2014

Parameters

nx	Grid dimension in the x direction (**input**)
ny	Grid dimension in the y direction (**input**)
nz	Grid dimension in the z direction (**input**)
dx	Grid spacing in the x direction Δx (**input**)
dy	Grid spacing in the x direction Δy (**input**)
dz	Grid spacing in the x direction Δz (**input**)
period_x	Flag for periodic boundaries in the x direction (**input**)
period_y	Flag for periodic boundaries in the y direction (**input**)
period_z	Flag for periodic boundaries in the z direction (**input**)
values	Array containing the sites's sizes a_i (**input/output)** When **gravity==.true.** this array is modified and contains the invasion potentials P_i at output time
states	Array containing the sites's states (**input/output)** Set **state(i)=neighboring** at sites **i** where you would like to inject the fluid Set **state(i)=exit_site** at sites **i** where the defending fluid can escape (the simulation will stop when the invading fluid percolates, i.e. it reaches one of these sites) Set **state(i)=sealed** to prevent sites **i** from being invaded At output time, we have **state(i)=trapped** at trapped sites At output time, we have **state(i)=invaded** at invaded sites
n_sites_-invaded	Number of sites invaded (**output**)
invasion_list	Array containing the list of sites invaded sorted in chronological order (**output**)
gravity	Flag for gravity (**input**) Set **gravity=.true.** to account for gravity Set **gravity=.false.** to ignore gravity
trapping	Flag for trapping (**input**) Set **trapping=.true.** to account for trapping Set **trapping=.false.** to ignore trapping
sigma	Surface tension σ (**input**)
theta_c	Equilibrium contact angle θ_c (**input**)
delta_rho	Fluid density contrast $\Delta\rho$ (**input**)
gx	Acceleration of gravity g in the x direction (**input**)
gy	Acceleration of gravity g in the y direction (**input**)
gz	Acceleration of gravity g in the z direction (**input**)

The documentation for this module was generated from the following file:

- modules/src/module_invasion_percolation.f90

4.7 module_invasion_percolation_constants Module Reference

This module is used to define and share some constants.

Public Attributes

- INTEGER, parameter NOT_INVADED = 0

  State flag.
- INTEGER, parameter NEIGHBORING = 1

  State flag.
- INTEGER, parameter TRAPPED = 2

  State flag.
- INTEGER, parameter EXIT_SITE = 3

  State flag.
- INTEGER, parameter INVADED = 4

  State flag.

4.7.1 Detailed Description

This module is used to define and share some constants.

Author

Yder Masson

Date

October 22, 2014

4.7.2 Member Data Documentation

4.7.2.1 INTEGER, parameter module_invasion_percolation_constants::EXIT_SITE = 3

State flag.

4.7.2.2 INTEGER, parameter module_invasion_percolation_constants::INVADED = 4

State flag.

4.7.2.3 INTEGER, parameter module_invasion_percolation_constants::NEIGHBORING = 1

State flag.

4.7.2.4 INTEGER, parameter module_invasion_percolation_constants::NOT_INVADED = 0

State flag.

4.7.2.5 INTEGER, parameter module_invasion_percolation_constants::TRAPPED = 2

State flag.

The documentation for this module was generated from the following file:

- modules/src/module_invasion_percolation_constants.f90

4.8 module_label_clusters Module Reference

This module contains functions for cluster labeling, these are used to identify trapped regions.

Public Member Functions

- subroutine label_clusters_cubic (mat, nx, ny, nz, period_x, period_y, period_z)

  Label clusters on a 3D cubic lattice.
- subroutine label_clusters_arbitrary (mat, n_sites, offsets, connectivity)

  Label clusters on arbitrary lattices.
- integer function get_label_mat (n, lc)

  Get label based on the neighbor's label
  If the site has neighbors, the site's label is the union of all neighbors's labels
  If the site has no neighbor, a new label is created.

4.8.1 Detailed Description

This module contains functions for cluster labeling, these are used to identify trapped regions.

Author

Yder MASSON

Date

October 23, 2014

4.8.2 Member Function/Subroutine Documentation

4.8.2.1 integer function module_label_clusters::get_label_mat (integer *n*, integer, dimension(:) *lc*)

Get label based on the neighbor's label

If the site has neighbors, the site's label is the union of all neighbors's labels

If the site has no neighbor, a new label is created.

Author

Yder MASSON

Date

October 23, 2014

Parameters

n	Number of neighbors
lc	Array containing the indices of the neighbors

4.8.2.2 subroutine module_label_clusters::label_clusters_arbitrary (integer, dimension(n_sites) *mat*, integer *n_sites*, integer, dimension(:) *offsets*, integer, dimension(:) *connectivity*)

Label clusters on arbitrary lattices.

Author

Yder MASSON

Date

October 23, 2014

Parameters

n_sites	Number of sites invaded (**output**)
mat	Matrix of clusters labels (**input** /**output**) At input time, **mat(i)=-1** at sites belonging to cluster to be labeled (sites filled with the defending fluid) and **mat(i)= 0** otherwise (sites filled with the invading fluid) At output time, **mat(i)=L** where **L** is the label of the cluster to which the site with index **i** belongs to
offsets	Array containing the offsets of the data stored in the connectivity array (**Input**)

connectivity	Array containing thelattice connectivity (**input**) For a given site **i**, with offset **j = offsets(i)** : **n=connectivity(j)** is the number of sites neighboring site **i**. **connectivity(j+1, j+2, ... ,j+n)** contains the indices of the sites that are neighboring site **i**.

4.8.2.3 subroutine module_label_clusters::label_clusters_cubic (integer, dimension(nx,ny,nz) *mat*, integer *nx*, integer *ny*, integer *nz*, logical *period_x*, logical *period_y*, logical *period_z*)

Label clusters on a 3D cubic lattice.

Author

Yder MASSON

Date

October 23, 2014

Parameters

nx	Grid dimension in the x direction (**input**)
ny	Grid dimension in the y direction (**input**)
nz	Grid dimension in the z direction (**input**)
mat	Matrix of clusters labels (**input** /**output**) At input time, **mat(i,j,k)=-1** at sites belonging to cluster to be labeled (sites filled with the defending fluid) and **mat(i,j,k)= 0** otherwise (sites filled with the invading fluid) At output time, **mat(i,j,k)=L** where **L** is the label of the cluster to which the site with index (i,j,k) belongs to
period_x	Flag for periodic boundaries in the x direction (**input**)
period_y	Flag for periodic boundaries in the y direction (**input**)
period_z	Flag for periodic boundaries in the z direction (**input**)

The documentation for this module was generated from the following file:

- modules/src/module_label_clusters.f90

4.9 module_random_media Module Reference

This module contains function for generating random media having a determined correlation function.

Public Member Functions

- subroutine gen_random_media_3D (mat, nx, ny, nz, dx, dy, dz, H, ak, ag, correlation_function)

  Generates a 3D matrix of random real numbers with zero mean and unity standard deviation and having a given correlation function.
- subroutine gen_random_media_2D (mat, nx, ny, dx, dy, H, ak, ag, correlation_function)

  Generates a 3D matrix of random real numbers with zero mean and unity standard deviation and having a given correlation function.
- subroutine fourn (data, nn, ndim, isign)

4.9.1 Detailed Description

This module contains function for generating random media having a determined correlation function.

Author

Yder MASSON

Date

November 9, 2014

See also

KLIMEŠ, Luděk. Correlation functions of random media. Pure and applied geophysics, 2002, vol. 159, no 7-8, p. 1811-1831.

4.9.2 Member Function/Subroutine Documentation

4.9.2.1 subroutine module_random_media::fourn (real, dimension(∗) *data,* integer, dimension(ndim) *nn,* integer *ndim,* integer *isign*)

4.9.2.2 subroutine module_random_media::gen_random_media_2D (real, dimension(nx,ny) *mat,* integer *nx,* integer *ny,* real *dx,* real *dy,* real *H,* real *ak,* real *ag,* character(len=∗) *correlation_function*)

Generates a 3D matrix of random real numbers with zero mean and unity standard deviation and having a given correlation function.

Author

Yder MASSON

Date

November 8, 2014

The correlation function currently available are :

General : $\hat{f}(k) = \kappa \left[a^{-2} + k^2\right]^{-\frac{d}{4}-\frac{H}{2}} \exp\left(\frac{a_G^2 k^2}{8}\right)$

Gaussian : $\hat{f}(k) = \kappa \exp\left(\frac{a_G^2 k^2}{8}\right)$

Von Karman : $\hat{f}(k) = \kappa \left[a^{-2} + k^2\right]^{-\frac{d}{4}-\frac{H}{2}}$

Exponential : $\hat{f}(k) = \kappa \left[a^{-2} + k^2\right]^{-\frac{d+1}{4}}$

Self-affine : $\hat{f}(k) = \kappa k^{-\frac{d}{2}-H}$

Kummer : $\hat{f}(k) = \kappa k^{-\frac{d}{2}-H} \exp\left(\frac{a_G^2 k^2}{8}\right)$

White noise : $\hat{f}(k) = \kappa$

in the above expression $d = 2$ in 2D

See also

KLIMEŠ, Luděk. Correlation functions of random media. Pure and applied geophysics, 2002, vol. 159, no 7-8, p. 1811-1831.

Parameters

mat	2D matrix containing the random values (**Output**)
nx	Grid dimension in the x direction (**Input**)
ny	Grid dimension in the y directionn (**Input**)
dx	Grid spacing in the x directionn (**Input**)
dy	Grid spacing in the y directionn (**Input**)
H	Hurst exponent H n (**Input**)
ak	Von karman correlation length a n (**Input**)
ag	Gaussian correlation length a_G n (**Input**)
correlation_function	Desired correlation function (character string **Input**) : use **correlation_function='general'** for the most general correlation function use **correlation_function='gaussian'** for a gaussian correlation function use **correlation_function='von_karman'** for a Von Karman correlation function use **correlation_function='self_affine'** for a self-affine correlation function use **correlation_function='kummer'** for a Kummer correlation function use **correlation_function='white_noise'** for a white noise (you may not need this function for that)

4.9.2.3 subroutine module_random_media::gen_random_media_3D (real, dimension(nx,ny,nz) *mat*, integer *nx*, integer *ny*, integer *nz*, real *dx*, real *dy*, real *dz*, real *H*, real *ak*, real *ag*, character(len=*) *correlation_function*)

Generates a 3D matrix of random real numbers with zero mean and unity standard deviation and having a given correlation function.

Author

Yder MASSON

Date

November 8, 2014

The correlation function currently available are :

General : $\hat{f}(k) = \kappa \left[a^{-2} + k^2\right]^{-\frac{d}{4}-\frac{H}{2}} \exp\left(\frac{a_G^2 k^2}{8}\right)$

Gaussian : $\hat{f}(k) = \kappa \exp\left(\frac{a_G^2 k^2}{8}\right)$

Von Karman : $\hat{f}(k) = \kappa \left[a^{-2} + k^2\right]^{-\frac{d}{4}-\frac{H}{2}}$

Exponential : $\hat{f}(k) = \kappa \left[a^{-2} + k^2\right]^{-\frac{d+1}{4}}$

Self-affine : $\hat{f}(k) = \kappa k^{-\frac{d}{2}-H}$

Kummer : $\hat{f}(k) = \kappa k^{-\frac{d}{2}-H} \exp\left(\frac{a_G^2 k^2}{8}\right)$

White noise : $\hat{f}(k) = \kappa$

in the above expression $d = 3$ in 3D

See also

KLIMEŠ, Lulěk. Correlation functions of random media. Pure and applied geophysics, 2002, vol. 159, no 7-8, p. 1811-1831.

Parameters

mat	3D matrix containing the random values (**Output**)
nx	Grid dimension in the x direction (**Input**)
ny	Grid dimension in the y directionn (**Input**)
nz	Grid dimension in the z directionn (**Input**)
dx	Grid spacing in the x directionn (**Input**)
dy	Grid spacing in the y directionn (**Input**)
dz	Grid spacing in the z directionn (**Input**)
H	Hurst exponent H n (**Input**)
ak	Von karman correlation length a n (**Input**)
ag	Gaussian correlation length a_G n (**Input**)

correlation_-function	Desired correlation function (character string **Input**) : use **correlation_function='general'** for the most general correlation function use **correlation_function='gaussian'** for a gaussian correlation function use **correlation_function='von_karman'** for a Von Karman correlation function use **correlation_function='self_affine'** for a self-affine correlation function use **correlation_function='kummer'** for a Kummer correlation function use **correlation_function='white_noise'** for a white noise (you may not need this function for that)

The documentation for this module was generated from the following file:

- modules/src/module_random_media.f90

4.10 module_trapping Module Reference

This module contains functions to identify trapped sites.

Public Member Functions

- integer function get_label_free_clusters (n_sites, states, mat)

  Ths function does the union of the labels of all sites connected to an exit site and return the corresponding label that is the free cluster's label.
- subroutine find_trapped_sites_cubic (nx, ny, nz, states, period_x, period_y, period_z, n_sites_invaded, invasion_list, undo_invasion)

  Identify trapped sites on cubic lattices.
- subroutine find_trapped_sites_arbitrary (n_sites, states, offsets, connectivity, n_sites_invaded, invasion_list, undo_invasion)

  find trapped sites in arbitrary lattices
- subroutine get_trapping_times_arbitrary (n_sites, states, offsets, connectivity, n_sites_invaded, invasion_list, trapping_times)

  This functions returns the times at which the sites got trapped. This function is for arbitrary lattices.

4.10.1 Detailed Description

This module contains functions to identify trapped sites.

Author

Yder MASSON

Date

October 23, 2014

4.10.2 Member Function/Subroutine Documentation

4.10.2.1 subroutine module_trapping::find_trapped_sites_arbitrary (integer *n_sites,* integer, dimension(:) *states,* integer, dimension(:) *offsets,* integer, dimension(:) *connectivity,* integer *n_sites_invaded,* integer, dimension(:) *invasion_list,* logical *undo_invasion*)

find trapped sites in arbitrary lattices

Author

Yder MASSON

Date

October 23, 2014

It uses the fast a posteriori identification of trapped node if the **undo_invasion=.true.**. If **undo_invasion=.false.** cluster labeling is performed at each invasion steps.

Parameters

states	Array containing the sites's states (**input/output)** The state of trapped sites is updated to **state(i)=trapped**
undo_-invasion	Flag: If **undo_invasion==.true.** Then use fast a posteriori method for trapping (**input**)
invasion_list	List of invaded sites sorted in chronological order (**input**)
offsets	Array containing the offsets of the data stored in the connectivity array (**Input**)
connectivity	Array containing thelattice connectivity (**input**) For a given site **i**, with offset **j = offsets(i)** : **n=connectivity(j)** is the number of sites neighboring site **i**. **connectivity(j+1, j+2, ... ,j+n)** contains the indices of the sites that are neighboring site **i**.

4.10.2.2 subroutine module_trapping::find_trapped_sites_cubic (integer *nx,* integer *ny,* integer *nz,* integer, dimension(nx,ny,nz) *states,* logical *period_x,* logical *period_y,* logical *period_z,* integer *n_sites_invaded,* integer, dimension(:) *invasion_list,* logical *undo_invasion*)

Identify trapped sites on cubic lattices.

Author

Yder MASSON

Date

October 23, 2014

It uses the fast a posteriori identification of trapped node if the **undo_invasion=.true.**. If **undo_invasion=.false.** cluster labeling is performed at each invasion steps.

Parameters

nx	Grid dimension in the x direction (**input**)
ny	Grid dimension in the y direction (**input**)
nz	Grid dimension in the z direction (**input**)
states	Array containing the sites's states (**input/output)** The state of trapped sites is updated to **state(i,j,k)=trapped**
period_x	Flag for periodic boundaries in the x direction (**input**)
period_y	Flag for periodic boundaries in the y direction (**input**)
period_z	Flag for periodic boundaries in the z direction (**input**)
undo_-invasion	Flag: If **undo_invasion==.true.** Then use fast a posteriori method for trapping (**input**)
n_sites_-invaded	Number of invaded sites (**input**)
invasion_list	List of invaded sites sorted in chronological order (**input**)

4.10.2.3 integer function module_trapping::get_label_free_clusters (*n_sites,* integer, dimension(n_sites) *states,* integer, dimension(n_sites) *mat*)

Ths function does the union of the labels of all sites connected to an exit site and return the corresponding label that is the free cluster's label.

Author

Yder MASSON

Date

October 23, 2014

Parameters

states	Array containing the sites's states (**input**)
mat	Array containing the clusters's labels (**input**)

4.10.2.4 subroutine module_trapping::get_trapping_times_arbitrary (integer *n_sites*, integer, dimension(:) *states*, integer, dimension(:) *offsets*, integer, dimension(:) *connectivity*, integer *n_sites_invaded*, integer, dimension(:) *invasion_list*, integer, dimension(:) *trapping_times*)

This functions returns the times at which the sites got trapped. This function is for arbitrary lattices.

Author

Yder MASSON

Date

November 11, 2014

Parameters

states	Array containing the sites's states (**input/output**) The state of trapped sites is updated to **state(i)=trapped**
invasion_list	List of invaded sites sorted in chronological order (**input**)
offsets	Array containing the offsets of the data stored in the connectivity array (**Input**)
connectivity	Array containing thelattice connectivity (**input**) For a given site **i**, with offset **j = offsets(i)** : **n=connectivity(j)** is the number of sites neighboring site **i**. **connectivity(j+1, j+2, ... ,j+n)** contains the indices of the sites that are neighboring site **i**.
trapping_-times	Array containing the times at which the sites got trapped. (**Output**) **trapping_times(i) = -1** at sites that have not been trapped

The documentation for this module was generated from the following file:

- modules/src/module_trapping.f90

4.11 module_write_output_files Module Reference

This modulecontains functions for writing the simulations results into data file, e,g., for post-processing and visualization.

Public Member Functions

- subroutine write_arbitrary_lattice_to_vtk (n_sites, states, values, n_sites_-invaded, invasion_list, offsets, connectivity, x, y, z, file_name, unit_vtk)

    write arbitrary lattice info to VTK file (.vtu) for viewing with e.g. Paraview
- subroutine write_cubic_lattice_to_vtk_cells (states, values, n_sites_invaded, invasion_list, nx, ny, nz, dx, dy, dz, file_name, unit_vtk)

write cubic lattice info to VTK file (.vti) for viewing with e.g. Paraview

- subroutine write_cubic_lattice_to_vtk_points (states, values, n_sites_invaded, invasion_list, nx, ny, nz, dx, dy, dz, file_name, unit_vtk)

  write cubic lattice info to VTK file (.vti) for viewing with e.g. Paraview
- subroutine funny_3D (mat, nx, ny, nz, matval)
- subroutine write_invasion_list_cubic_to_csv (invasion_list, n_sites_invaded, nx, ny, dx, dy, dz, file_name)
- subroutine write_invasion_list_arbitrary_to_csv (invasion_list, n_sites_invaded, x, y, z, n_sites, file_name)
- subroutine write_invasion_list_cubic_to_vtk (invasion_list, n_sites_invaded, nx, ny, dx, dy, dz, file_name)

  Write the list of invaded sites to a VTK file (i.e. a polydata xml file with extension .vtp) for viewing with e.g. Paraview (for use with cubic lattices).
- subroutine write_invasion_list_arbitrary_to_vtk (invasion_list, n_sites_invaded, x, y, z, n_sites, file_name)

  Write the list of invaded sites to a VTK file (i.e. a polydata xml file with extension .vtp) for viewing with e.g. Paraview (for use with arbitrary lattices).

4.11.1 Detailed Description

This modulecontains functions for writing the simulations results into data file, e,g., for post-processing and visualization.

Author

Yder MASSON

Date

November 9, 2014

4.11.2 Member Function/Subroutine Documentation

4.11.2.1 subroutine module_write_output_files::funny_3D (integer, dimension(nx,ny,nz) *mat,* integer *nx,* integer *ny,* integer *nz,* integer *matval*)

Author

Yder MASSON

Date

October 30, 2014 Produces a minimalistic 3D rendering of IP clusters inside a terninal

Parameters

nx	grid dimension in the x direction (**Input**)
ny	grid dimension in the y direction (**Input**)
nz	grid dimension in the z direction (**Input**)
mat	input matrix (**Input**)
matval	Value to render, i.e. only the cells where **mat(i,j,k) = matval** will be showed (**Input**)

4.11.2.2 subroutine module_write_output_files::write_arbitrary_lattice_to_vtk (integer *n_sites*, integer, dimension(:) *states*, real, dimension(:) *values*, integer *n_sites_invaded*, integer, dimension(:) *invasion_list*, integer, dimension(:) *offsets*, integer, dimension(:) *connectivity*, real, dimension(:) *x*, real, dimension(:) *y*, real, dimension(:) *z*, character(len=∗) *file_name*, integer *unit_vtk*)

write arbitrary lattice info to VTK file (.vtu) for viewing with e.g. Paraview

Author

Yder MASSON

Date

October 28, 2014

The states and values data are attached to points (i.e. sites), you can view these using a glyph filter, for example.

The bonds linking sites can be visualized by plotting the wireframe.

Parameters

n_sites	Total number of sites in the lattice (**Input**)
states	Sites's states as defiend in invasion percolation module (**Input**)
values	Sites's values as defiend in invasion percolation module (**Input**)
n_sites_-invaded	number of sites invaded
invasion_list	list of invaded sites
offsets	Array containing the offsets of the data stored in the connectivity array (**Input**)
connectivity	Array containing thelattice connectivity (**input**) For a given site **i**, with offset **j = offsets(i)** : **n=connectivity(j)** is the number of sites neighboring site **i**. **connectivity(j+1, j+2, ... ,j+n)** contains the indices of the sites that are neighboring site **i**.
x	Array containing the x coordinates of the sites (**Input**)
y	Array containing the y coordinates of the sites (**Input**)
z	Array containing the z coordinates of the sites (**Input**)
file_name	Output file name, must have the.vtu extension (**Input**)
unit_vtk	logical unit for output file (**Input**)

4.11.2.3 subroutine module_write_output_files::write_cubic_lattice_to_vtk_cells (integer, dimension(nx,ny,nz) *states*, real, dimension(nx,ny,nz) *values*, integer *n_sites_invaded*, integer, dimension(:) *invasion_list*, integer *nx*, integer *ny*, integer *nz*, real *dx*, real *dy*, real *dz*, character(len=∗) *file_name*, integer *unit_vtk*)

write cubic lattice info to VTK file (.vti) for viewing with e.g. Paraview

Author

Yder MASSON

Date

October 30, 2014

The states and values data are attached to cells (i.e. cels represent sites)

Parameters

values	values array as defined in the invasion percolation module (**Input**)
states	states array as defined in the invasion percolation module (**Input**)
n_sites_-invaded	number of sites invaded
invasion_list	list of invaded sites
nx	grid dimension in the x direction (**Input**)
ny	grid dimension in the y direction (**Input**)
nz	grid dimension in the z direction (**Input**)
dx	grid spacing in the x direction (**Input**)
dy	grid spacing in the y direction (**Input**)
dz	grid spacing in the z direction (**Input**)
file_name	Output file name, must have the.vtu extension (**Input**)
unit_vtk	logical unit for output file (**Input**)

4.11.2.4 subroutine module_write_output_files::write_cubic_lattice_to_vtk_points (integer, dimension(nx,ny,nz) *states*, real, dimension(nx,ny,nz) *values*, integer *n_sites_invaded*, integer, dimension(:) *invasion_list*, integer *nx*, integer *ny*, integer *nz*, real *dx*, real *dy*, real *dz*, character(len=∗) *file_name*, integer *unit_vtk*)

write cubic lattice info to VTK file (.vti) for viewing with e.g. Paraview

Author

Yder MASSON

Date

October 30, 2014

The states and values data are attached to points (i.e. points represent sites)

The bonds linking sites can be plotted by viewing the wireframe in paraview.

Parameters

values	values array as defined in the invasion percolation module (**Input**)
states	states array as defined in the invasion percolation module (**Input**)
n_sites_-invaded	number of sites invaded
invasion_list	list of invaded sites
nx	grid dimension in the x direction (**Input**)
ny	grid dimension in the y direction (**Input**)
nz	grid dimension in the z direction (**Input**)
dx	grid spacing in the x direction (**Input**)
dy	grid spacing in the y direction (**Input**)
dz	grid spacing in the z direction (**Input**)
file_name	Output file name, must have the.vtu extension (**Input**)
unit_vtk	logical unit for output file (**Input**)

4.11.2.5 subroutine module_write_output_files::write_invasion_list_arbitrary_to_csv (integer, dimension(:) *invasion_list*, integer *n_sites_invaded*, real, dimension(n_sites) *x*, real, dimension(n_sites) *y*, real, dimension(n_sites) *z*, integer *n_sites*, character (len=*) *file_name*)

Parameters

n_sites_-invaded	number of sites invaded
invasion_list	list of invaded sites
n_sites	number of sites i the lattice
x	x coordinates array
y	y coordinates array
z	z coordinates array
file_name	name of the output file (the extension .csv will be added if not present)

4.11.2.6 subroutine module_write_output_files::write_invasion_list_arbitrary_to_vtk (integer, dimension(:) *invasion_list*, integer *n_sites_invaded*, real, dimension(n_sites) *x*, real, dimension(n_sites) *y*, real, dimension(n_sites) *z*, integer *n_sites*, character (len=*) *file_name*)

Write the list of invaded sites to a VTK file (i.e. a polydata xml file with extension .vtp) for viewing with e.g. Paraview (for use with arbitrary lattices).

Author

Yder MASSON

Date

November 5, 2014

Parameters

n_sites_-invaded	number of sites invaded
invasion_list	list of invaded sites
n_sites	number of sites i the lattice
x	x coordinates array
y	y coordinates array
z	z coordinates array
file_name	name of the output file (the extension .vtp will be added if not present)

4.11.2.7 subroutine module_write_output_files::write_invasion_list_cubic_to_csv (integer, dimension(:) *invasion_list,* integer *n_sites_invaded,* integer *nx,* integer *ny,* real *dx,* real *dy,* real *dz,* character (len=∗) *file_name*)

Parameters

n_sites_-invaded	number of sites invaded
invasion_list	list of invaded sites
nx	grid dimension in the x direction
ny	grid dimension in the y direction
dx	grid spacing in the x direction
dy	grid spacing in the y direction
dz	grid spacing in the z direction
file_name	name of the output file (the extension .csv will be added if not present)

4.11.2.8 subroutine module_write_output_files::write_invasion_list_cubic_to_vtk (integer, dimension(:) *invasion_list,* integer *n_sites_invaded,* integer *nx,* integer *ny,* real *dx,* real *dy,* real *dz,* character (len=∗) *file_name*)

Write the list of invaded sites to a VTK file (i.e. a polydata xml file with extension .vtp) for viewing with e.g. Paraview (for use with cubic lattices).

Author

Yder MASSON

Date

November 5, 2014

Parameters

n_sites_-invaded	number of sites invaded
invasion_list	list of invaded sites
nx	grid dimension in the x direction
ny	grid dimension in the y direction
dx	grid spacing in the x direction
dy	grid spacing in the y direction
dz	grid spacing in the z direction
file_name	name of the output file (the extension .vtp will be added if not present)

The documentation for this module was generated from the following file:

- modules/src/module_write_output_files.f90

Chapter 5

File Documentation

5.1 examples/src/example_1.f90 File Reference

Functions/Subroutines

- program example_1
  a simple example showing how to use invason percolation

5.1.1 Function/Subroutine Documentation

5.1.1.1 program example_1 ( )

a simple example showing how to use invason percolation

a brief description example showing how to use invason percolation

a detailed simple example showing how to use invason percolation

5.2 examples/src/example_2.f90 File Reference

Functions/Subroutines

- program example_2
  a simple example showing how to model invason percolation on arbitrary lattices

5.2.1 Function/Subroutine Documentation

5.2.1.1 program example_2 ( )

a simple example showing how to model invason percolation on arbitrary lattices a brief description example showing how to use invason percolation

a detailed simple example showing how to use invason percolation

5.3 examples/src/test_binary_tree.f90 File Reference

Functions/Subroutines

- program test_binary_tree

  This is to test the binary_tree module.

5.3.1 Function/Subroutine Documentation

5.3.1.1 program test_binary_tree ( )

This is to test the binary_tree module.

Author

Yder MASSON

Date

October 7, 2014

This is to check the binary_tree module is working as expected: 1) generate random values 2) construct the binary tree 3) recursively pick and update the root node and check that the values are well sorted

5.4 examples/src/test_disjoint_set.f90 File Reference

Functions/Subroutines

- program test_disjoint_set

  This is to test the disjoint set module.

5.4.1 Function/Subroutine Documentation

5.4.1.1 program test_disjoint_set ( )

This is to test the disjoint set module.

Author

Yder MASSON

Date

October 7, 2014

1) create a few label or classes 2) check all newly created classes are canonical classes 3) union some label or classes 4) check that the union have been performed correctly

5.5 examples/src/test_invasion_percolation.f90 File Reference

Functions/Subroutines

- program test_invasion_percolation

  This programs makes sure all the invasion functions are producing the same output for a given lattice.

5.5.1 Function/Subroutine Documentation

5.5.1.1 program test_invasion_percolation ( )

This programs makes sure all the invasion functions are producing the same output for a given lattice.

Author

Yder MASSON

Date

November 1, 2014

5.6 modules/src/module_binary_tree.f90 File Reference

Data Types

- module module_binary_tree

  *Defines the binary tree data structure and the associated **add_branch** and **update_-root** procedures.*

5.7 modules/src/module_cubic_indices.f90 File Reference

Data Types

- module module_cubic_indices

  This module contains functions to transform 3D indices to 1D index and vice versa.

5.8 modules/src/module_disjoint_set.f90 File Reference

Data Types

- module module_disjoint_set

  *Define disjoint set data structure and the associated **Union**, **Find** and **Create_set** procedures.*

5.9 modules/src/module_gravity.f90 File Reference

Data Types

- module module_gravity

  This module contains the functions to account for gravity.

5.10 modules/src/module_interpolation.f90 File Reference

Data Types

- module module_interpolation

5.11 modules/src/module_invasion_percolation.f90 File Reference

Data Types

- module module_invasion_percolation

5.12 modules/src/module_invasion_percolation_constants.f90 File Reference

Data Types

- module module_invasion_percolation_constants

This module is used to define and share some constants.

5.13 modules/src/module_label_clusters.f90 File Reference

Data Types

- module module_label_clusters

  This module contains functions for cluster labeling, these are used to identify trapped regions.

5.14 modules/src/module_random_media.f90 File Reference

Data Types

- module module_random_media

  This module contains function for generating random media having a determined correlation function.

5.15 modules/src/module_trapping.f90 File Reference

Data Types

- module module_trapping

  This module contains functions to identify trapped sites.

5.16 modules/src/module_write_output_files.f90 File Reference

Data Types

- module module_write_output_files

  This modulecontains functions for writing the simulations results into data file, e,g., for post-processing and visualization.