wtools Documentation

Bane Sullivan

Nov 16, 2018

Contents

1 Getting Started **3**

2 Contributing **5**
- 2.1 About W-Tools . . . 5
- 2.2 File I/O . . . 5
- 2.3 Mesh Tools . . . 6
- 2.4 Plotting Routines . . . 9
- 2.5 Array Transforms . . . 10
- 2.6 Geostatistics . . . 12

Python Module Index **17**

CHAPTER 1

Getting Started

`wtools` is installable using `pip`. We haven't yet deployed `wtools` on PyPI as it is in its very early stages.

To get started using `wtools`, clone this project:

```
$ git clone https://github.com/csmwteam/wtools.git
```

Then go into that cloned development directory and perform a local installation via `pip` in your active virtual environment:

```
$ cd wtools
$ pip install -e .
```

Then be sure to add the `wtools` repository directory to your *PYTHONPATH*.

CHAPTER 2

Contributing

Do you want to add features? Then go ahead and make commits to the project and push them to GitHub and create a Pull Request!

In your virtual environment, make sure you have all of the proper dependancies installed:

```
$ pip install -r requirements.txt
```

2.1 About W-Tools

- Author: Bane Sullivan
- License: BSD-3-Clause
- Copyright: 2018, Colorado School of Mines W-Team
- Version: 0.0.3

wtools: a Python package for W-Team research needs

2.2 File I/O

This module holds several methods for standard file I/O for the data formats that we work with regularly. Much of this regarding *Grid* objects in inherrited directly into the *Grid* class.

2.2.1 readGSLib

`wtools.fileio.`**`readGSLib`**(*filename*)

This will read a standard GSLib or GeoEAS data file to a pandas `DataFrame`.

Parameters **`filename`** (*`str`*) – the string file name of the data to load. This can be a relative or abslute file path.

Returns A table containing the all data arrays. Note that an attribute called `header` is added to the data frame contianing the string header line of the file.

Return type pandas.DataFrame

2.2.2 saveGSLib

`wtools.fileio.`**`saveGSLib`**(*filename*, *dataframe*, *header=None*)
This will save a pandas dataframe to a GSLib file

2.3 Mesh Tools

This provides a class for discretizing data in a convienant way that makes sense for our spatially referenced data/models.

2.3.1 Grid

`class` `wtools.mesh.`**`Grid`**(***kwargs*)
Bases: `properties.base.base.HasProperties`, `wtools.fileio.GridFileIO`

A data structure to store a model space discretization and different attributes of that model space.

Example:

```
>>> import wtools
>>> import numpy as np
>>> models = {
    'rand': np.random.random(1000).reshape((10,10,10)),
    'spatial': np.arange(1000).reshape((10,10,10)),
    }
>>> grid = wtools.Grid(models=models)
>>> grid.validate() # Make sure the data object was created successfully
True
```

Note: See Jupyter notebooks under the `examples` directory

Required Properties:

- **origin** (`Vector3`): The lower southwest corner of the data volume., a 3D Vector of <type 'float'> with shape (3), Default: [0.0, 0.0, 0.0]
- **xtensor** (`Array`): Tensor cell widths, x-direction, a list or numpy array of <type 'float'>, <type 'int'> with shape (*)
- **ytensor** (`Array`): Tensor cell widths, y-direction, a list or numpy array of <type 'float'>, <type 'int'> with shape (*)
- **ztensor** (`Array`): Tensor cell widths, z-direction, a list or numpy array of <type 'float'>, <type 'int'> with shape (*)

Optional Properties:

- **models** (`Dictionary`): The volumetric data as a 3D NumPy arrays in <X,Y,Z> or <i,j,k> coordinates. Each key value pair represents a different model for the gridded model space. Keys will be treated as the string name of the model., a dictionary (keys: a unicode string; values: a list or numpy array of <type 'float'>, <type 'int'> with shape (*, *, *))

bounds
The bounds of the grid

classmethod deserialize(*value*, *trusted=False*, *strict=False*, *assert_valid=False*, ***kwargs*)
Creates **HasProperties** instance from serialized dictionary

This uses the Property deserializers to deserialize all JSON-compatible dictionary values into their corresponding Property values on a new instance of a **HasProperties** class. Extra keys in the dictionary that do not correspond to Properties will be ignored.

Parameters:

- **value** - Dictionary to deserialize new instance from.
- **trusted** - If True (and if the input dictionary has `'__class__'` keyword and this class is in the registry), the new **HasProperties** class will come from the dictionary. If False (the default), only the **HasProperties** class this method is called on will be constructed.
- **strict** - Requires `'__class__'`, if present on the input dictionary, to match the deserialized instance's class. Also disallows unused properties in the input dictionary. Default is False.
- **assert_valid** - Require deserialized instance to be valid. Default is False.
- Any other keyword arguments will be passed through to the Property deserializers.

display(*plt*, *key*, *plane='xy'*, *slc=None*, *showit=True*, ***kwargs*)
Display a 2D slice of this grid.

Parameters

- **plt** (*handle*) – the active plotting handle to use
- **key** (*str*) – the string name of the model to view
- **plane** (`'xy'`, `'xz'`, `'yz'`) – The plane to slice upon
- **slc** (*float*) – the coordinate along the sliced dimension
- **showit** (*bool*) – A flag for whether or not to call `plt.show()`

equal(*other*)
Compare this Grid to another Grid

getDataRange(*key*)
Get the data range for a given model

getNodePoints()
Get ALL nodes in the gridded volume as an XYZ point set

keys
List of the string names for each of the models

models
a unicode string; values: a list or numpy array of <type 'float'>, <type 'int'> with shape (*, *, *))

Type **models** (Dictionary)

Type The volumetric data as a 3D NumPy arrays in <X,Y,Z> or <i,j,k> coordinates. Each key value pair represents a different model for the gridded model space. Keys will be treated as the string name of the model., a dictionary (keys

num_cells
Number of cells

num_nodes
Number of nodes (vertices)

nx
Number of cells in the X direction

ny
Number of cells in the Y direction

nz
Number of cells in the Z direction

origin
[0.0, 0.0, 0.0]

Type **origin** (`Vector3`)

Type The lower southwest corner of the data volume., a 3D Vector of <type 'float'> with shape (3), Default

classmethod readSGeMSGrid(*fname, origin=[0.0, 0.0, 0.0], spacing=[1.0, 1.0, 1.0]*)
Reads an SGeMS grid file where grid shape is defined in the header as three integers seperated by whitespace. Data arrays are treated as 3D and given in <x, y, z> indexing to a `Grid` object.

Parameters

- **fname** (*str*) – the string file name of the data to load. This can be a relative or abslute file path.
- **origin** (*iter(float)*) – the southwest-bottom corner of the grid.
- **spacing** (*iter(float)*) – the cell spacings for each axial direction

Returns The SGeMS data loaded onto a `Grid` object.

Return type *Grid*

classmethod readUBC(*name*, *directory=''*)

saveSGeMS(*filename*)
This will save the grid in the SGeMS gridded data file format

saveUBC(*fname*)
Save the grid in the UBC mesh format.

serialize(*include_class=True*, *save_dynamic=False*, ***kwargs*)
Serializes a **HasProperties** instance to dictionary

This uses the Property serializers to serialize all Property values to a JSON-compatible dictionary. Properties that are undefined are not included. If the **HasProperties** instance contains a reference to itself, a `properties.SelfReferenceError` will be raised.

Parameters:

- **include_class** - If True (the default), the name of the class will also be saved to the serialized dictionary under key `'__class__'`
- **save_dynamic** - If True, dynamic properties are written to the serialized dict (default: False).
- Any other keyword arguments will be passed through to the Property serializers.

shape
3D shape of the grid (number of cells in all three directions)

classmethod tableToGrid(*df, shp, origin=[0.0, 0.0, 0.0], spacing=[1.0, 1.0, 1.0], order='F'*)
Converts a pandas `DataFrame` table to a `Grid` object.

Parameters

- **shp** (*tuple(int)*) – length 3 tuple of integers sizes for the data grid dimensions.
- **origin** (*iter(float)*) – the southwest-bottom corner of the grid.
- **spacing** (*iter(float)*) – the cell spacings for each axial direction.
- **order** (`'C'`, `'F'`, `'A'`) – the reshape order.

Returns The data table loaded onto a `Grid` object.

Return type *Grid*

toDataFrame(*order='C'*)
Returns the models in this Grid to a Pandas DataFrame with all arrays flattened in the specified order. A header attribute is added to the DataFrame to specified the grid extents. Much metadata is lost in this conversion.

validate()

xcenters
The cell center coordinates along the X-axis

xnodes
The node coordinates along the X-axis

xtensor
Tensor cell widths, x-direction, a list or numpy array of <type 'float'>, <type 'int'> with shape (*)

Type **xtensor** (`Array`)

ycenters
The cell center coordinates along the Y-axis

ynodes
The node coordinates along the Y-axis

ytensor
Tensor cell widths, y-direction, a list or numpy array of <type 'float'>, <type 'int'> with shape (*)

Type **ytensor** (`Array`)

zcenters
The cell center coordinates along the Z-axis

znodes
The node coordinates along the Z-axis

ztensor
Tensor cell widths, z-direction, a list or numpy array of <type 'float'>, <type 'int'> with shape (*)

Type **ztensor** (`Array`)

2.4 Plotting Routines

`plots`: This module provides various plotting routines that ensure we display our spatially referenced data in logical, consistant ways across projects.

2.4.1 display

`wtools.plots.`**display**(*plt*, *arr*, *x=None*, *y=None*, ***kwargs*)
This provides a convienant class for plotting 2D arrays that avoids treating our data like images. Since most

datasets we work with are defined on Cartesian coordinates, <i,j,k> == <x,y,z>, we need to transpose our arrays before plotting in image plotting libraries like `matplotlib`.

Parameters

- **plt** (*handle*) – your active plotting handle
- **arr** (*np.ndarray*) – A 2D array to plot
- **kwargs** (*dict*) – Any kwargs to pass to the `pcolormesh` plotting routine

Returns plt.pcolormesh

Example

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> arr = np.arange(1000).reshape((10,100))
>>> wtools.display(plt, arr)
>>> plt.title('What we actually want')
>>> plt.colorbar()
>>> plt.show()
```

2.4.2 plotStructGrid

`wtools.plots.`**plotStructGrid**(*plt*, *outStruct*, *gridspecs=None*, *imeas=None*)

Plot a semivariogram or covariogram produced from raster2structgrid

Parameters

- **plt** (*handle*) – An active plotting handle. This allows us to use the plotted result after the routine.
- **outStruct** (*np.ndarray*) – the data to plot
- **gridspecs** (*list (GridSpec)*) – the spatial reference of your gdata
- **imeas** (*str*) – key indicating which structural measure to label: `'var'` for semi-variogram or `'covar'` for covariogram. This simply adds a few labels to the active figure. If semi-variance use `True`. If covariance, use `False`.

Returns plt.plot or plt.pcolor

2.5 Array Transforms

`transform`: This module provides several conveinance methods for transforming NumPy arrays in a Cartesian coordinate system.

2.5.1 emptyArray

`wtools.transform.`**emptyArray**(*shp*)

Creates a NumPy ndarray of the given shape that is gaurnteed to be initialized to all NaN values

Parameters **shp** (*tuple (int)*) – A tuple of integers specifiy the shape of the array

2.5.2 meshgrid

wtools.transform.**meshgrid**(*x*, *y*, *z=None*)

Use this convienance method for your meshgrid needs. This ensures that we always use <ij> indexing to stay consistant with Cartesian grids.

This simply provides a wrapper for `np.meshgrid` ensuring we always use `indexing='ij'` which makes sense for typical Cartesian coordinate systems (<x,y,z>).

Note: This method handles 2D or 3D grids.

Example

```
>>> import wtools
>>> import numpy as np
>>> x = np.arange(20, 200, 10)
>>> y = np.arange(20, 500, 20)
>>> z = np.arange(0, 1000, 50)
>>> xx, yy, zz = wtools.meshgrid(x, y, z)
>>> # Now check that axii are ordered correctly
>>> assert(xx.shape[0] == len(x))
>>> assert(xx.shape[1] == len(y))
>>> assert(xx.shape[2] == len(z))
```

2.5.3 transpose

wtools.transform.**transpose**(*arr*)

Transpose matrix from Cartesian to Earth Science coordinate system. This is useful for UBC Meshgrids where +Z is down.

Note: Works forward and backward.

Parameters **arr** (*ndarray*) – 3D NumPy array to transpose with ordering: <i,j,k>

Returns same array transposed from <i,j,k> to <j,i,-k>

Return type ndarray

Example

```
>>> import wtools
>>> import numpy as np
>>> model = np.random.random(1000).reshape((10, 20, 5))
>>> wtools.transpose(model).shape
(20, 10, 5)
```

2.6 Geostatistics

2.6.1 Grids

GridSpec

class wtools.geostats.grids.**GridSpec**(***kwargs*)

Bases: properties.base.base.HasProperties

A GridSpec object provides the details of a single axis along a grid. If you have a 3D grid then you will have 3 GridSpec objects.

Required Properties:

- **min** (Integer): The minimum value along this dimension. The origin., an integer, Default: 0
- **n** (Integer): The number of components along this dimension., an integer
- **sz** (Integer): The uniform cell size along this dimension., an integer, Default: 1

Optional Properties:

- **nnodes** (Integer): The number of grid nodes to consider on either side of the origin in the output map, an integer

classmethod deserialize(*value*, *trusted=False*, *strict=False*, *assert_valid=False*, ***kwargs*)

Creates **HasProperties** instance from serialized dictionary

This uses the Property deserializers to deserialize all JSON-compatible dictionary values into their corresponding Property values on a new instance of a **HasProperties** class. Extra keys in the dictionary that do not correspond to Properties will be ignored.

Parameters:

- **value** - Dictionary to deserialize new instance from.
- **trusted** - If True (and if the input dictionary has '__class__' keyword and this class is in the registry), the new **HasProperties** class will come from the dictionary. If False (the default), only the **HasProperties** class this method is called on will be constructed.
- **strict** - Requires '__class__', if present on the input dictionary, to match the deserialized instance's class. Also disallows unused properties in the input dictionary. Default is False.
- **assert_valid** - Require deserialized instance to be valid. Default is False.
- Any other keyword arguments will be passed through to the Property deserializers.

equal(*other*)

Determine if two **HasProperties** instances are equivalent

Equivalence is determined by checking if all Property values on two instances are equal, using Property.equal.

min

0

Type **min** (Integer)

Type The minimum value along this dimension. The origin., an integer, Default

n

The number of components along this dimension., an integer

Type **n** (Integer)

nnodes
The number of grid nodes to consider on either side of the origin in the output map, an integer

Type **nnodes** (Integer)

serialize(*include_class=True*, *save_dynamic=False*, ***kwargs*)
Serializes a **HasProperties** instance to dictionary

This uses the Property serializers to serialize all Property values to a JSON-compatible dictionary. Properties that are undefined are not included. If the **HasProperties** instance contains a reference to itself, a properties.SelfReferenceError will be raised.

Parameters:

- **include_class** - If True (the default), the name of the class will also be saved to the serialized dictionary under key '__class__'
- **save_dynamic** - If True, dynamic properties are written to the serialized dict (default: False).
- Any other keyword arguments will be passed through to the Property serializers.

sz
1

Type **sz** (Integer)

Type The uniform cell size along this dimension., an integer, Default

validate()
Call all registered class validator methods

These are all methods decorated with @properties.validator. Validator methods are expected to raise a ValidationError if they fail.

geoeas2numpy

wtools.geostats.grids.**geoeas2numpy**(*datain*, *nx*, *ny=None*, *nz=None*)
Transform GeoEas array into np.ndarray to be treated like image. Function to transform a SINGLE GoeEas-formatted raster (datain) i.e., a single column, to a NumPy array that can be viewed using imshow (in 2D) or slice (in 3D).

Parameters

- **datain** (*np.ndarray*) – 1D input GeoEas-formatted raster of dimensions:
- **nx** (*int*) – the number of dimensions along the 1st axis
- **ny** (*int, optional*) – the number of dimensions along the 2nd axis
- **nz** (*int, optional*) – the number of dimensions along the 3rd axis

Returns

If only nx given: 1D array. If only nx and ny given: 2D array. If nx, ny, and nz given: 3D array.

Return type np.ndarray

Note: In 3D, z increases upwards

References

Originally implemented in MATLAB by: Phaedon Kyriakidis, Department of Geography, University of California Santa Barbara, May 2005

Reimplemented into Python by: Bane Sullivan and Jonah Bartrand, Department of Geophysics, Colorado School of Mines, October 2018

geoeas2numpyGS

wtools.geostats.grids.**geoeas2numpyGS**(*datain*, *gridspecs*)

A wrapper for `geoeas2numpy` to handle a list of `GridSpec` objects

Parameters **gridspecs** (*list(GridSpec)*) – array with grid specifications using `GridSpec` objects

2.6.2 Rasters

This module provides useful methods for operating on 1D and 2D rasters such as making variogram or covariograms.

raster2structgrid

wtools.geostats.raster.**raster2structgrid**(*datain*, *imeas='covar'*, *rtol=1e-10*)

Create an auto-variogram or auto-covariance map from 1D or 2D rasters. This computes auto-variogram or auto-covariance maps from 1D or 2D rasters. This function computes variograms/covariances in the frequency domain via the Fast Fourier Transform (`np.fftn`).

Note: For viewing the results, please use the `plotStructGrid` method from the `plots` module.

Note: Missing values, flagged as `np.nan`, are allowed.

Parameters

- **datain** (*np.ndarray*) – input arrray with raster in GeoEas format
- **imeas** (*str*) – key indicating which structural measure to compute: `'var'` for semi-variogram or `'covar'` for covariogram.
- **gridspecs** (*list(GridSpec)*) – array with grid specifications using `GridSpec` objects
- **rtol** (*float*) – the tolerance. Default is 1e-10

Returns

output array with variogram or covariogram map, depending on variogram choice, with size: in 1D: (2*nxOutHalf+1) or in 2D: (2*nxOutHalf+1 x 2*nxOutHalf+1).

output array with number of pairs available in each lag, of same size as outStruct

Return type tuple(np.ndarray, np.ndarray)

References

Originally implemented in MATLAB by: Phaedon Kyriakidis, Department of Geography, University of California Santa Barbara, May 2005

Reimplemented into Python by: Jonah Bartrand, Department of Geophysics, Colorado School of Mines, October 2018

Algorith based on: Marcotte, D. (1996): Fast Variogram Computation with FFT, Computers & Geosciences, 22(10), 1175-1186.

suprts2modelcovFFT

wtools.geostats.raster.**suprts2modelcovFFT**(*CovMapExtFFT*, *ind1Ext*, *sf1Ext*, *ind2Ext*, *sf2Ext*)

Integrated model covariances between 1 or 2 sets of arbitrary supports. Function to calculate array of TOTAL or AVERAGE model covariances between 1 or 2 sets of irregular supports, using convolution in the frequency domain (FFT-based). Integration or averaging is IMPLICIT in the pre-computed sampling functions (from discrsuprtsFFT).

Parameters

- **CovMapExtFFT** (*np.ndarray*) – Fourier transform of model covariance map evaluated at nodes of an extended MATLAB grid
- **ind1Ext** – (nSup1 x 1) cell array with MATLAB indices of non-zero sampling function values for support set #1 in extended MATLAB grid
- **sf1Ext** – (nSup1 x 1) cell array with sampling function values for support set #1
- **ind2Ext** – Optional (nSup2 x 1) cell array with MATLAB indices of non-zero sampling function values for support set #2 in extended MATLAB grid
- **sf2Ext** – Optional (nSup2 x 1) cell array with sampling function values for support set #2

Returns (nSup1 x nSup[1,2]) array with integrated covariances

Return type np.ndarray

References

Originally implemented in MATLAB by: Phaedon Kyriakidis, Department of Geography, University of California Santa Barbara, May 2005

Reimplemented into Python by: Bane Sullivan and Jonah Bartrand, Department of Geophysics, Colorado School of Mines, October 2018

Docs Automatically Generated

Learn more about automatic documentation

Python Module Index

w

wtools.fileio, 5
wtools.geostats.grids, 12
wtools.geostats.raster, 14
wtools.mesh, 6
wtools.plots, 9
wtools.transform, 10

Index

B

bounds (wtools.mesh.Grid attribute), 6

D

deserialize() (wtools.geostats.grids.GridSpec class method), 12
deserialize() (wtools.mesh.Grid class method), 7
display() (in module wtools.plots), 9
display() (wtools.mesh.Grid method), 7

E

emptyArray() (in module wtools.transform), 10
equal() (wtools.geostats.grids.GridSpec method), 12
equal() (wtools.mesh.Grid method), 7

G

geoeas2numpy() (in module wtools.geostats.grids), 13
geoeas2numpyGS() (in module wtools.geostats.grids), 14
getDataRange() (wtools.mesh.Grid method), 7
getNodePoints() (wtools.mesh.Grid method), 7
Grid (class in wtools.mesh), 6
GridSpec (class in wtools.geostats.grids), 12

K

keys (wtools.mesh.Grid attribute), 7

M

meshgrid() (in module wtools.transform), 11
min (wtools.geostats.grids.GridSpec attribute), 12
models (wtools.mesh.Grid attribute), 7

N

n (wtools.geostats.grids.GridSpec attribute), 12
nnodes (wtools.geostats.grids.GridSpec attribute), 12
num_cells (wtools.mesh.Grid attribute), 7
num_nodes (wtools.mesh.Grid attribute), 7
nx (wtools.mesh.Grid attribute), 7
ny (wtools.mesh.Grid attribute), 8
nz (wtools.mesh.Grid attribute), 8

O

origin (wtools.mesh.Grid attribute), 8

P

plotStructGrid() (in module wtools.plots), 10

R

raster2structgrid() (in module wtools.geostats.raster), 14
readGSLib() (in module wtools.fileio), 5
readSGeMSGrid() (wtools.mesh.Grid class method), 8
readUBC() (wtools.mesh.Grid class method), 8

S

saveGSLib() (in module wtools.fileio), 6
saveSGeMS() (wtools.mesh.Grid method), 8
saveUBC() (wtools.mesh.Grid method), 8
serialize() (wtools.geostats.grids.GridSpec method), 13
serialize() (wtools.mesh.Grid method), 8
shape (wtools.mesh.Grid attribute), 8
suprts2modelcovFFT() (in module wtools.geostats.raster), 15
sz (wtools.geostats.grids.GridSpec attribute), 13

T

tableToGrid() (wtools.mesh.Grid class method), 8
toDataFrame() (wtools.mesh.Grid method), 9
transpose() (in module wtools.transform), 11

V

validate() (wtools.geostats.grids.GridSpec method), 13
validate() (wtools.mesh.Grid method), 9

W

wtools.fileio (module), 5
wtools.geostats.grids (module), 12
wtools.geostats.raster (module), 14
wtools.mesh (module), 6
wtools.plots (module), 9
wtools.transform (module), 10

X

xcenters (wtools.mesh.Grid attribute), 9
xnodes (wtools.mesh.Grid attribute), 9
xtensor (wtools.mesh.Grid attribute), 9

Y

ycenters (wtools.mesh.Grid attribute), 9
ynodes (wtools.mesh.Grid attribute), 9
ytensor (wtools.mesh.Grid attribute), 9

Z

zcenters (wtools.mesh.Grid attribute), 9
znodes (wtools.mesh.Grid attribute), 9
ztensor (wtools.mesh.Grid attribute), 9