wtools Documentation

Bane Sullivan

Oct 29, 2018

Contents

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 .
```

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.

2.2.1 readGSLib

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

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

Parameters **`fname`** (*`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 readSGeMSGrid

`wtools.fileio.`**`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 `GriddedData` 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 `GriddedData` object.

Return type *GriddedData*

2.2.3 tableToGrid

`wtools.fileio.`**`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 `GriddedData` 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 `GriddedData` object.

Return type *GriddedData*

2.3 Mesh Tools

`mesh`: This module provides numerous methods and classes for discretizing data in a convienant way that makes sense for our spatially referenced data/models.

2.3.1 GriddedData

`class` `wtools.mesh.`**`GriddedData`**(***kwargs*)

Bases: `properties.base.base.HasProperties`

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.GriddedData(models=models)
>>> grid.validate() # Make sure the data object was created successfully
True
```

Note: See Jupyter notebooks under the `examples` directory

Required 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 (*, *, *))
- **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 (*)

`bounds`
The bounds of the grid

`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()`

`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

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

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

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)

2.3.2 meshgrid

wtools.mesh.**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))
```

2.3.3 saveUBC

wtools.mesh.**saveUBC**(*fname*, *x*, *y*, *z*, *models*, *header='Data'*, *widths=False*, *origin=(0.0, 0.0, 0.0)*)

Saves a 3D gridded array with spatail reference to the UBC mesh/model format. Use PVGeo to visualize this data. For more information on the UBC mesh format, reference the GIFtoolsCookbook website.

Warning: This method assumes your mesh and data are defined on a normal cartesian system: <x,y,z>

Parameters

- **fname** (*str*) – the string file name of the mesh file. Model files will be saved next to this file.
- **x** (*ndarray or float*) – a 1D array of unique coordinates along the X axis, float for uniform cell widths, or an array with `widths==True` to treat as cell spacing on X axis
- **y** (*ndarray or float*) – a 1D array of unique coordinates along the Y axis, float for uniform cell widths, or an array with `widths==True` to treat as cell spacing on Y axis
- **z** (*ndarray or float*) – a 1D array of unique coordinates along the Z axis, float for uniform cell widths, or an array with `widths==True` to treat as cell spacing on Z axis
- **models** (*dict*) – a dictionary of models. Key is model name and value is a 3D array with dimensions <x,y,z> containing cell data.
- **header** (*str*) – a string header for your mesh/model files

- **widths** (*bool*) – flag for whether to treat the (x, y, z) args as cell sizes/widths
- **origin** (*tuple(float)*) – optional origin value used if widths==True, or used on a component basis if any of the x, y, or z args are scalars.

Yields Saves out a mesh file named {fname}.msh and a model file for every key/value pair in the models argument (key is file extension for model file and value is the data.

Examples

```
>>> import numpy as np
>>> # Create the unique coordinates along each axis : 11 nodes on each axis
>>> x = np.linspace(0, 100, 11)
>>> y = np.linspace(220, 500, 11)
>>> z = np.linspace(0, 50, 11)
>>> # Create some model data: 10 cells on each axis
>>> arr = np.array([i*j*k for i in range(10) for j in range(10) for k in␣
↪range(10)]).reshape(10, 10, 10)
>>> models = dict( foo=arr )
>>> # Define the name of the file
>>> fname = 'test'
>>> # Perfrom the write out
>>> saveUBC(fname, x, y, z, models, header='A simple model')
>>> # Two files saved: 'test.msh' and 'test.foo'
```

```
>>> import numpy as np
>>> # Uniform cell sizes
>>> d = np.random.random(1000).reshape((10, 10, 10))
>>> v = np.random.random(1000).reshape((10, 10, 10))
>>> models = dict(den=d, vel=v)
>>> saveUBC('volume', 25, 25, 2, models, widths=True, origin=(200.0, 100.0, 500.
↪0))
>>> # Three files saved: 'volume.msh', 'volume.den', and 'volume.vel'
```

2.3.4 transpose

wtools.mesh.**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)
```

2.4 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 OrthographicSlicer

class wtools.plots.**OrthographicSlicer**(*plt*, *grid*, *model*, *xslice=None*, *yslice=None*, *zslice=None*)

Plot slices of the 3D volume.

Use `x`, `y`, and `z` keyword arguments to specify the constant value to plot agianst. If none given, will use center of volume.

Parameters

- **plt** (*handle*) – An active plotting handle. This allows us to use the plotted result after the routine.
- **grid** (`GriddedData`) – The grid to plot
- **model** (*str*) – The model name to plot (the attribute data)
- **y, or z** (*x,*) – the constant values to slice against.

Returns None

adjust_x(*plt*, *x*)

adjust_y(*plt*, *y*)

adjust_z(*plt*, *z*)

clear_element(*name*)

Remove element <name> from plot if it exists.

static find_nearest_idx(*a*, *a0*)

update_xy(*plt*)

Update plot for change in Z-index.

update_xz(*plt*)

Update plot for change in Y-index.

update_zy(*plt*)

Update plot for change in X-index.

2.4.2 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()
```

2.4.3 plotStructGrid

`wtools.plots.plotStructGrid`(*plt*, *outStruct*, *gridspecs*, *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 Geostatistics

2.5.1 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
- **n** (`Integer`): The number of components along this dimension., an integer
- **sz** (`Integer`): The uniform cell size along this dimension., an integer

Optional Properties:

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

min
The minimum value along this dimension. The origin., an integer

Type **min** (`Integer`)

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`)

sz
The uniform cell size along this dimension., an integer

Type **sz** (`Integer`)

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

2.5.2 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

Index

V

W

X

Y

Z