DHI Tools Documentation

Release 1.1.0

Rob Wall

Apr 04, 2019

Contents

1 Install **3**

2 Quickstart **5**

3 API Documentation **9**
- 3.1 Mesh . . . 9
- 3.2 Dfsu . . . 13
- 3.3 Dfs0 . . . 20
- 3.4 Dfs1 . . . 21
- 3.5 Dfs2 . . . 22
- 3.6 Units . . . 24

4 Features **27**

5 Examples **29**

Python Module Index **31**

Python tools for working with DHI MIKE21.

See also:

CHAPTER 1

Install

Requirements

- MIKE SDK 2019
- GDAL/OGR
- Geopandas
- Pythonnet

Due to depending on the **MIKE SDK DLL** libraries only Windows is supported.

Install

Recommended that Anaconda is used to install **GDAL** and **geopandas**. Alternatively, see here and here for installation instructions of these packages.

First, install **MIKE software development kit**:

Download installer from here

After installing the MIKE SDK:

```
conda install gdal
conda install geopandas
pip install pythonnet
pip install dhitools
```

CHAPTER 2

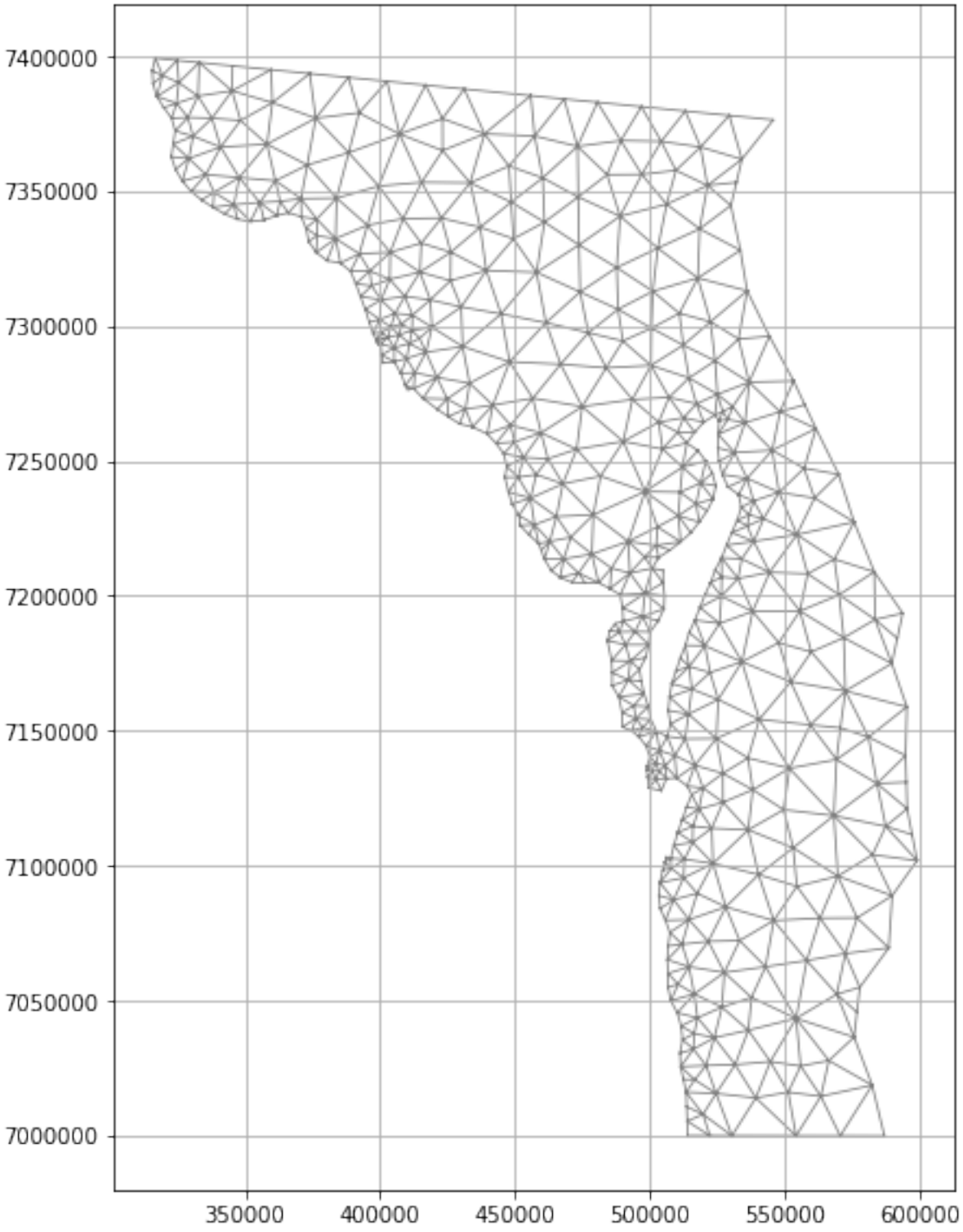
Quickstart

Read in **.mesh** file and inspect mesh

```
from dhitools import mesh
mesh_f = "path/to/mesh/file"
m = mesh.Mesh(mesh_f)

 # Plotting accepts matplotlib.triplot kwargs
kwargs = dict(color='grey', linewidth=0.8)
f1, a1 = m.plot_mesh(kwargs=kwargs)

f1.set_size_inches(10,10)
a1.grid()
a1.set_aspect('equal')
plt.show()
```



Read in **.dfsu** file and plot surface elevation at timestep 500

```
from dhitools import dfsu
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

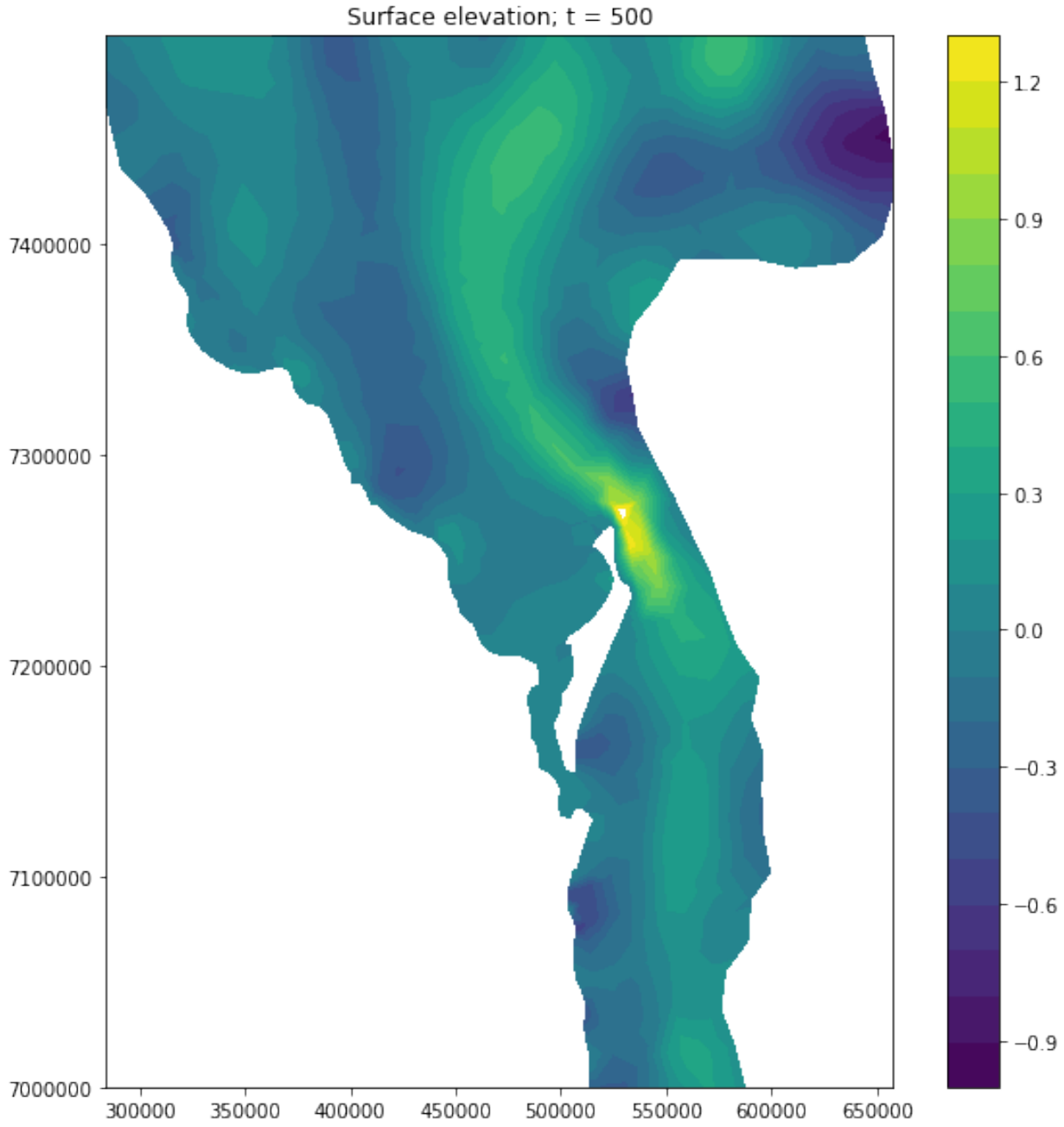
```
dfsu_f = "path/to/dfsu/file"
area = dfsu.Dfsu(dfsu_f)

plot_dict = dict(levels = np.arange(-1,1.4,0.1))
fig_se, ax_se, tf_se = area.plot_item(item_name='Surface elevation', tstep=400,
                                                    kwargs=plot_dict)

fig_se.set_size_inches(10,10)
ax_se.set_aspect('equal')

fig_se.colorbar(tf_se)

ax_se.set_title('Surface elevation; t = 500')
plt.show()
```


Surface elevation; t = 500
7400000
7300000
7200000
7100000
7000000
300000
350000
400000
450000
500000
550000
600000
650000
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6
−0.9

CHAPTER 3

API Documentation

3.1 Mesh

class dhitools.mesh.**Mesh**(*filename=None*)

MIKE21 mesh class. Contains many attributes read in from the input *.mesh* file.

Parameters **filename** (*str*) – Path to .mesh

filename

Path to .mesh

Type str

nodes

(x,y,z) coordinate for each node

Type ndarray, shape (num_nodes, 3)

elements

(x,y,z) coordinate for each element

Type ndarray, shape (num_ele, 3)

element_table

Defines for each element the nodes that define the element.

Type ndarray, shape (num_ele, 3)

node_table

Defines for each node the element adjacent to this node. May contain padded zeros

Type ndarray, shape (num_nodes, n)

node_ids

Ordered node ids

Type ndarray, shape (num_nodes,)

node_boundary_code
Each nodes boundary code

Type ndarray, shape (num_nodes,)

element_ids
Ordered element ids

Type ndarray, shape (num_elements,)

num_nodes
Number of nodes elements

Type int

num_elements
Number of mesh elements

Type int

projection
.mesh spatial projection string in WKT format

Type str

zUnitKey
EUM unit designating quantity of Z variable:

- 1000 = metres
- 1014 = U.S. feet

Type int

lyrs
Stores additional layers from *lyr_from_shape()*

Type dict

See also:

- Many of these methods have been adapated from the DHI MATLAB Toolbox
- Method *grid_res()*: Grid interpolation paramters which have additional attributes if calculated

boolean_mask(*res=1000*, *mesh_mask=None*)
Create a boolean mask of a regular grid at input resolution indicating if gridded points are within the model mesh.

Parameters

- **res** (*int*) – Grid resolution
- **mesh_mask** (*shapely Polygon object, optional*) – Mesh domain mask output from the method *mask()*. If this is not provided, it will be created by *mask()*. *mesh_mask* will be used to determine gridded points that are within the polygon.

Returns **bool_mask** – Boolean mask covering the regular grid for the mesh domain

Return type ndarray, shape (len_xgrid, len_ygrid)

grid_res(*res*, *nodes=True*)
Calculate grid parameters at specified resolution for either nodes or element coordinates. These parameters are used for interpolating node or element values to regular spaced grids efficiently.

Parameters

- **res** (*int*) – grid resolution
- **nodes** (*bool*) – If True, use node coordinates as input Else, use element coordinates

Returns

- *Updates the following class attributes*
- **grid_x** (*ndarray, shape (len_xgrid, len_ygrid)*) – x grid at specified resolution
- **grid_y** (*ndarray, shape (len_xgrid, len_ygrid)*) – y grid at specified resolution
- **grid_vertices** (*ndarray, shape (num_nodes/elements, 3)*) – vertices for triangulation applied to (x, y) for input to interpolation
- **grid_weights** (*ndarray, shape (num_nodes/elements, 3)*) – weights for grid_x and grid_y based on unstructured node/element (x,y). Input for interpolation.

interpolate_rasters(*raster_list*, *method='nearest'*)

Interpolate multiple raster elevations to mesh nodes

Parameters

- **raster_list** (*list*) – List of filepaths to each raster to interpolate from Order listed will be order in which interpolation is performed
- **method** (*str*) – 'nearest' or 'linear'

Returns **Updates mesh.nodes z coordinates**

Return type *Mesh.nodes* z update

See also:

Interpolation methods:

- NearestNDInterpolator
- LinearNDInterpolator

lyr_from_shape(*lyr_name*, *input_shp*, *field_attribute*, *output_shp=None*)

Create a model input layer at mesh element coordinates.

For example, input_shp is a roughness map containing polygons with roughness values. A spatial join is performed for mesh element points within input_shp polygons and returns field_attributeat element points.

Parameters

- **lyr_name** (*str*) – Layer name as key to *lyrs* attribute dictionary
- **input_shp** (*str*) – Path to input shape file
- **field_attributes** (*str*) – Attribute in *input_shp* to extract at mesh elements
- **output_shp** (*str, optional*) – output path to write to .shp file

Returns

- Inserts *lyr_name* into the *lyrs* attribute dictionary as an ndarray,
- shape (num_elements,) with extracted *field_attributes* value for each
- *mesh element*

lyr_to_dfsu(*lyr_name*, *output_dfsu*, *item_type=<sphinx.ext.autodoc.importer._MockObject object>*, *unit_type=<sphinx.ext.autodoc.importer._MockObject object>*)

Create model layer .dfsu file *lyr* attribute. References *lyrs* attribute dictionary as value at element coordinates to write to .dfsu file.

See also *lyr_from_shape()*.

Parameters

- **lyr_name** (*str*) – Layer name as key to *lyrs* attribute dictionary
- **output_dfsu** (*str*) – Path to output .dfsu file
- **item_type** (*int*) – MIKE21 item code. See *get_item()*. Default is "Mannings M"
- **unit_type** (*int*) – MIKE21 unit code. See *get_unit()*. Default is "Mannings M" unit "cube root metre per second"

Returns **Creates a new dfsu file at output_dfsu**

Return type dfsu file

mask()

Create a Shapely polygon mesh domain mask.

Determines mesh boundary from node boundary codes.

Returns **poly_mask** – Polygon of the mesh domain.

Return type shapely Polygon object

mesh_details()

Get min and max for input x and y ndarrays; shape (num_nodes,)

Returns

- **min_x** (*float*)
- **max_x** (*float*)
- **min_y** (*float*)
- **max_y** (*float*)

meshgrid(*res*)

Create X and Y meshgrid covering node coordinates

Parameters **res** (*int*) – grid resolution

Returns

- **grid_x** (*ndarray, shape (len_xgrid, len_ygrid)*) – x grid at specified resolution
- **grid_y** (*ndarray, shape (len_xgrid, len_ygrid)*) – y grid at specified resolution

plot_mesh(*fill=False*, *kwargs=None*)

Plot triangular mesh with triplot or tricontourf.

See matplotlib kwargs for respective additional plot arguments.

Warning: if mesh is large performance will be poor

Parameters

- **fill** (*boolean*) – if True, plots filled contour mesh (tricontourf) if False, plots (x, y) triangular mesh (triplot)
- **kwargs** (*dict*) – Additional arguments supported by triplot/tricontourf

Returns

- **fig** (*matplotlib figure obj*) – Figure object
- **ax** (*matplotlib axis obj*) – Axis object
- If *fill* is True –

  tf [matplotlib tricontourf obj] Tricontourf object

See also:

- Triplot
- Tricontourf

read_mesh (*filename=None*)
Read in .mesh file

Parameters **filename** (*str*) – File path to .mesh file

summary ()
Prints a summary of the mesh

to_gpd (*elements=True*, *output_shp=None*)
Export mesh elements or nodes to GeoDataFrame with option to write to shape file

Parameters

- **elements** (*boolean*) – if True, export element points if False, export nodes points
- **output_shp** (*str, optional*) – output path to write to .shp file

Returns **mesh_df** – Geopandas df with field for element or node id if specified

Return type GeoDataFrame, shape (nrows, 2)

write_mesh (*output_name*)
Write new mesh file

Parameters **output_name** (*str*) – File path to write node (x, y, z) to .mesh file Include .mesh at the end of string

3.2 Dfsu

class dhitools.dfsu.**Dfsu** (*filename=None*)
Bases: *dhitools.mesh.Mesh*

MIKE21 dfsu class. Contains many attributes read in from the input *.dfsu* file. Uses *dhitools.mesh.Mesh* as a base class and inherits its methods.

Parameters **filename** (*str*) – Path to .dfsu

filename
Path to .dfsu

Type str

items
List .dfsu items (ie. surface elevation, current speed), item index to lookup in .dfsu, item units and counts of elements, nodes and time steps.

Type dict

projection
.dfsu spatial projection string in WKT format

Type str

ele_table
Defines for each element the nodes that define the element.

Type ndarray, shape (num_ele, 3)

node_table
Defines for each node the element adjacent to this node. May contain padded zeros

Type ndarray, shape (num_nodes, n)

nodes
(x,y,z) coordinate for each node

Type ndarray, shape (num_nodes, 3)

elements
(x,y,z) coordinate for each element

Type ndarray, shape (num_ele, 3)

start_datetime_str
Start datetime (as a string)

Type str

start_datetime
Start datetime (datetime object)

Type datetime

end_datetime
End datetime (datetime object)

Type datetime

timestep
Timestep delta in seconds

Type float

number_tstep
Total number of timesteps

Type int

time
Sequence of datetimes between start and end datetime at delta timestep

Type ndarray, shape (number_tstep,)

See also:

- Many of these methods have been adapated from the DHI MATLAB Toolbox

boolean_mask(*mesh_mask*, *res=1000*)
Create a boolean mask of a regular grid at input resolution indicating if gridded points are within the model mesh.

This is slightly different to the mesh method which will automatically create the mask if it isn't provided. This will not automatically create the mask and the mask method has been disabled. See mask() for further details.

Parameters

- **res** (*int*) – Grid resolution
- **mesh_mask** (*shapely Polygon object*) – Mesh domain mask output from the *mask()* or any shapely polygon. *mesh_mask* will be used to determine gridded points that are within the polygon.

Returns **bool_mask** – Boolean mask covering the regular grid for the mesh domain

Return type ndarray, shape (len_xgrid, len_ygrid)

create_dfsu(*arr*, *item_name*, *output_dfsu*, *start_datetime=None*, *timestep=None*, *item_type=<sphinx.ext.autodoc.importer._MockObject object>*, *unit_type=<sphinx.ext.autodoc.importer._MockObject object>*)

Create a new *dfsu* file based on the underlying *Dfsu()* for some new non-temporal or temporal layer.

Parameters

- **arr** (*ndarray, shape (num_elements, num_timesteps)*) – Array to write to dfsu file. Number of rows must equal the number of elements in the *Dfsu()* object and the order of the array must align with the order of the elements. Can create a non-temporal *dfsu* layer of a single dimensional input *arr*, or a temporal *dfsu* layer at *timestep* from *start_datetime*.
- **item_name** (*str*) – Name of item to write to *dfsu*
- **output_dfsu** (*str*) – Path to output .dfsu file
- **start_datetime** (*datetime*) – Start datetime (datetime object). If *None*, use the base *Dfsu() start_datetime*.
- **timestep** (*float*) – Timestep delta in seconds. If *None*, use the base *Dfsu() timestep*.
- **item_type** (*str*) – MIKE21 item code. See *get_item()*. Default is "Mannings M"
- **unit_type** (*str*) – MIKE21 unit code. See *get_unit()*. Default is "Mannings M" unit "cube root metre per second"

Returns **Creates a new dfsu file at output_dfsu**

Return type dfsu file

ele_to_node(*z_element*)

Convert data at element coordinates to node coordinates

Parameters **z_element** (*ndarray, shape (num_elements,)*) – Data corresponding to order and coordinates of elements

Returns **z_node** – Data corresponding to order and coordinates of nodes

Return type ndarray, shape (num_nodes,)

gridded_item(*item_name=None*, *tstep_start=None*, *tstep_end=None*, *res=1000*, *node=True*, *node_data=None*)

Calculate gridded item data, either from nodes or elements, at specified grid resolution and for a range of time steps. Allows for downsampling of high resolution mesh's to a more manageable size.

The method *grid_res()* needs to be run before this to calculate the grid parameters needed for interpolation. Pre-calculating these also greatly improves run-time. res and node must be consistent between grid_res() and gridded_item().

Parameters

- **item_name** (*str*) – Specified item to return node data. Item names are found in the *Dfsu.items* attribute.
- **tstep_start** (*int or None, optional*) – Specify time step for node data. Timesteps begin from 0. If *None*, returns data from 0 time step.
- **tstep_end** (*int or None, optional*) – Specify last time step for node data. Allows for range of time steps to be returned, where *tstep_end* is included.Must be positive int <= number of timesteps If *None*, returns single time step specified by *tstep_start* If *-1*, returns all time steps from *tstep_start*:end
- **res** (*int*) – Grid resolution
- **node** (*bool*) – If true, interpolate from node data, Else, interpolate from element data
- **node_data** (*ndarray or None, shape (num_nodes,), optional*) – Provide data at node coordinates to create grid from. Will take precedence over *item_name*.

Returns **z_interp** – Interpolated z grid for each timestep

Return type ndarray, shape (num_timsteps, len_xgrid, len_ygrid)

gridded_stats(*item_name*, *tstep_start=None*, *tstep_end=None*, *node=True*, *max=True*, *res=1000*)

Calculate gridded item maximum or minimum across time range, either from nodes or elements, at specified grid resolution. Allows for downsampling of high resolution mesh's to a more manageable size.

The method grid_res() needs to be run before this to calculate the grid parameters needed for interpolation. Pre-calculating these also greatly improves run-time. res and node must be consistent between grid_res() and gridded_item().

Parameters

- **item_name** (*str*) – Specified item to return element data. Item names are found in the *Dfsu.items* attribute.
- **tstep_start** (*int or None, optional*) – Specify time step for data considered in determining maximum. Timesteps begin from 0. If *None*, returns data from 0 time step.
- **tstep_end** (*int or None, optional*) – Specify last time step for data considered in determining maximum Must be positive int <= number of timesteps If *None*, returns all time steps from *tstep_start*:end
- **node** (*boolean, optional*) – If True, returns item data at node rather than element
- **max** (*boolean, optional*) – If True, returns max (see method max_item()) else returns min

Returns **z_interp** – Interpolated z grid

Return type ndarray, shape (len_xgrid, len_ygrid)

item_element_data(*item_name*, *tstep_start=None*, *tstep_end=None*, *element_list=None*)

Get element data for specified item with option to specify range of timesteps.

Parameters

- **item_name** (*str*) – Specified item to return element data. Item names are found in the *Dfsu.items* attribute.

- **tstep_start** (*int or None, optional*) – Specify time step for element data. Timesteps begin from 0. If *None*, returns data from 0 time step.
- **tstep_end** (*int or None, optional*) – Specify last time step for element data. Allows for range of time steps to be returned, where *tstep_end* is included.Must be positive int <= number of timesteps If *None*, returns single time step specified by *tstep_start* If *-1*, returns all time steps from *tstep_start*:end
- **element_list** (*list, optional*) – Provide list of elements. Element numbers are as seen by MIKE programs and adjusted for Python indexing.

Returns **ele_data** – Element data for specified item and time steps *element_list* will change num_elements returned in *ele_data*

Return type ndarray, shape (num_elements,[tstep_end-tstep_start])

item_node_data(*item_name*, *tstep_start=None*, *tstep_end=None*)

Get node data for specified item with option to specify range of timesteps.

Parameters

- **item_name** (*str*) – Specified item to return node data. Item names are found in the *Dfsu.items* attribute.
- **tstep_start** (*int or None, optional*) – Specify time step for node data. Timesteps begin from 0. If *None*, returns data from 0 time step.
- **tstep_end** (*int or None, optional*) – Specify last time step for node data. Allows for range of time steps to be returned, where *tstep_end* is included.Must be positive int <= number of timesteps If *None*, returns single time step specified by *tstep_start* If *-1*, returns all time steps from *tstep_start*:end

Returns **node_data** – Node data for specified item and time steps

Return type ndarray, shape (num_nodes,[tstep_end-tstep_start])

mask()

Method disabled for dfsu class since the node boundary codes for dfsu files are not consistent with mesh boundary codes particularly when dfsu output is a subset of the mesh

max_ampltiude(*item_name='Maximum water depth'*, *datum_shift=0*, *nodes=True*)

Calculate maximum amplitude from MIKE21 inundation output.

Specifically, takes the MIKE21 output for *Maximum water depth* across the model run, adjusted for *datum_shift* and calculates maximum amplitude by the difference between the depth and mesh elevation

Datum shift applies are different water level to a model run and the mesh elevation values saved within the *dfsu* file will be adjusted by the datum shift. So, providing the datum shift is necessary to calculate the correct amplitudes.

Parameters

- **item_name** (*str*) – Default is 'Maxium water depth' which is the default output name from MIKE21. Can parse an alternative string if a different name has been used.
- **datum_shift** (*float*) – Adjust for datum_shift value used during model run. Only necessary if a datum shift was applied to the model. Default is 0.
- **nodes** (*boolean*) – If True, return data at node coordinates If False, return data at element coordinates

Returns

- if *node* is True

- **max_amplitude** (*ndarray, shape (num_nodes,)*) – Max amplitude across entire model run at node coordinates
- if *node* is False
- **max_amplitude** (*ndarray, shape (num_elements,)*) – Max amplitude across entire model run at element coordinates

max_item(*item_name*, *tstep_start=None*, *tstep_end=None*, *current_dir=False*, *node=False*)

Calculate maximum element value for specified item over entire model or within specific range of timesteps.

Parameters

- **item_name** (*str*) – Specified item to return element data. Item names are found in the *Dfsu.items* attribute.
- **tstep_start** (*int or None, optional*) – Specify time step for data considered in determining maximum. Timesteps begin from 0. If *None*, returns data from 0 time step.
- **tstep_end** (*int or None, optional*) – Specify last time step for data considered in determining maximum Must be positive int <= number of timesteps If *None*, returns all time steps from *tstep_start*:end
- **current_dir** (*boolean*) – If True, returns corresponding current direction value occuring at the maxmimum of specified *item_name*.
- **node** (*boolean, optional*) – If True, returns item data at node rather than element

Returns

- If *current_dir* is False
- **max_ele** (*ndarray, shape (num_elements,)*) – Maximum elements values for specified item
- If *current_dir* is True
- **max_ele** (*ndarray, shape (num_elements,)*) – Maximum elements values for specified item
- **max_current_dir** (*ndarray, shape (num_elements,)*) – Current direction corresponding to *max_ele*
- if *node* is True
- **min_node** (*ndarray, shape (num_nodes,)*) – Minimum node values for specified item
- If *node* and *current_dir* are True
- **min_node** (*ndarray, shape (num_nodes,)*) – Minimum node values for specified item
- **min_current_dir** (*ndarray, shape (num_elements,)*) – Current direction corresponding to *min_node*

min_item(*item_name*, *tstep_start=None*, *tstep_end=None*, *current_dir=False*, *node=False*)

Calculate minimum element value for specified item over entire model or within specific range of timesteps.

Parameters

- **item_name** (*str*) – Specified item to return element data. Item names are found in the *Dfsu.items* attribute.
- **tstep_start** (*int or None, optional*) – Specify time step for data considered in determining minimum. Timesteps begin from 0. If *None*, returns data from 0 time step.

- **tstep_end** (*int or None, optional*) – Specify last time step for data considered in determining minimum Must be positive int <= number of timesteps If *None*, returns all time steps from *tstep_start*:end
- **current_dir** (*boolean*) – If True, returns corresponding current direction value occuring at the maxmimum of specified *item_name*.
- **node** (*boolean, optional*) – If True, returns item data at node rather than element

Returns

- If *current_dir* is False –

  min_ele [ndarray, shape (num_elements,)] Minimum elements values for specified item

- If *current_dir* is True –

  min_ele [ndarray, shape (num_elements,)] Minimum elements values for specified item

  min_current_dir [ndarray, shape (num_elements,)] Current direction corresponding to *min_ele*

- if *node* is True –

  min_node [ndarray, shape (num_nodes,)] Minimum node values for specified item

- If *node* and *current_dir* are True –

  min_node [ndarray, shape (num_nodes,)] Minimum node values for specified item

  min_current_dir [ndarray, shape (num_elements,)] Current direction corresponding to *min_node*

plot_item(*item_name=None*, *tstep=None*, *node_data=None*, *kwargs=None*)

Plot triangular mesh with tricontourf for input item and timestep

Warning: if mesh is large performance will be poor

Parameters

- **item_name** (*str*) – Specified item to return element data. Item names are found in the *Dfsu.items* attribute.
- **tstep** (*int*) – Specify time step for node data. Timesteps begin from 0.
- **node_date** (*ndarray or None, shape (num_nodes,), optional*) – Provide data at node coordinates to plot. Will take precedence over *item_name* and *tstep*.
- **kwargs** (*dict*) – Additional arguments supported by tricontourf

Returns

- **fig** (*matplotlib figure obj*)
- **ax** (*matplotlib axis obj*)
- **tf** (*tricontourf obj*)

read_dfsu(*filename*)

Read in .dfsu file and read attributes

Parameters **filename** (*str*) – File path to .dfsu file

summary()

Prints a summary of the dfsu

3.3 Dfs0

class dhitools.dfs.**Dfs0**(*filename*)

Bases: dhitools.dfs._Dfs

MIKE21 dfs0 class. Contains many attributes read in from the input *.dfs0* file.

Parameters **filename** (*str*) – Path to .dfs0

filename
Path to .dfs0

Type str

data
Pandas DataFrame containing .dfs0 item data. Indexed by time. Columns are each .dfs0 item name.

Type pandas.DataFrame, shape (num_timesteps, num_items)

num_items
Total number of .dfs0 items

Type int

items
List .dfs0 items (ie. surface elevation, current speed), item names, item indexto lookup in .dfs0, item units and counts of elements, nodes and time steps.

Type dict

start_datetime
Start datetime (datetime object)

Type datetime

end_datetime
End datetime (datetime object)

Type datetime

timestep
Timestep delta in seconds

Type float

number_tstep
Total number of timesteps

Type int

time
Sequence of datetimes between start and end datetime at delta timestep

Type ndarray, shape (number_tstep,)

dfs_info(*dfs_object*)
Make a dictionary with .dfs items and other attributes.

See class attributes

dfs_time()
Create a time sequency between start and end datetime

summary()
Prints a summary of the dfs

3.4 Dfs1

class dhitools.dfs.**Dfs1**(*filename*)
Bases: dhitools.dfs._Dfs

MIKE21 dfs1 class. Contains many attributes read in from the input *.dfs1* file.

Parameters **filename** (*str*) – Path to .dfs1

filename
Path to .dfs1

Type str

num_items
Total number of .dfs1 items

Type int

num_points
Total number of .dfs1 profile points within each item

Type int

items
List .dfs1 items (ie. surface elevation, current speed), item names, item indexto lookup in .dfs1, item units and counts of elements, nodes and time steps. Contains item data, accessed by dict key *item_name*. This is more easily accessed by *item_data()*.

Type dict

start_datetime
Start datetime (datetime object)

Type datetime

end_datetime
End datetime (datetime object)

Type datetime

timestep
Timestep delta in seconds

Type float

number_tstep
Total number of timesteps

Type int

time
Sequence of datetimes between start and end datetime at delta timestep

Type ndarray, shape (number_tstep,)

dfs_info(*dfs_object*)
Make a dictionary with .dfs items and other attributes.

See class attributes

dfs_time()
Create a time sequency between start and end datetime

item_data(*item_name*)
Return pandas DataFrame of *dfs1* item data.

Parameters **item_name** (*str*) – Specified item to return element data. Item names can be found in *items* attribute or by *summary()*.

Returns **data** – Pandas DataFrame containing .dfs1 item data. Indexed by time. Columns are each of the profile points.

Return type pandas.DataFrame, shape (num_timesteps, num_points)

summary()
Prints a summary of the dfs

3.5 Dfs2

class dhitools.dfs.**Dfs2**(*filename*)
Bases: dhitools.dfs._Dfs

MIKE21 dfs2 class. Contains many attributes read in from the input *.dfs2* file.

Parameters **filename** (*str*) – Path to .dfs2

filename
Path to .dfs2

Type str

num_items
Total number of .dfs2 items

Type int

num_points
Total number of .dfs2 profile points within each item

Type int

items
List .dfs2 items (ie. surface elevation, current speed), item names, item indexto lookup in .dfs2, item units and counts of elements, nodes and time steps. Contains item data, accessed by dict key *item_name*. This is more easily accessed by *item_data()*.

Type dict

start_datetime
Start datetime (datetime object)

Type datetime

end_datetime
End datetime (datetime object)

Type datetime

timestep
Timestep delta in seconds

Type float

number_tstep
Total number of timesteps

Type int

time
Sequence of datetimes between start and end datetime at delta timestep

Type ndarray, shape (number_tstep,)

projection
.mesh spatial projection string in WKT format

Type str

X
X meshgrid

Type ndarray, shape (y_count, x_count)

Y
Y meshgrid

Type ndarray, shape (y_count, x_count)

gridshape
.dfs2 grid shape

Type tuple, length 2

x_count
Number of x points

Type int

y_count
Number of y points

Type int

del_x
X grid step

Type int

del_y
Y grid step

Type int

x_max
Max x value

Type int

x_min
Min x value

Type int

y_max
Max y value

Type int

y_min
Min y value

Type int

nodata_float
Nodata value for type float data

Type float

nodata_double
Nodata value for type double data

Type float

nodata_int
Nodata value for type int data

Type int

dfs_info(*dfs_object*)
Make a dictionary with .dfs items and other attributes.

See class attributes

dfs_time()
Create a time sequency between start and end datetime

item_data(*item_name*, *tstep_start=None*, *tstep_end=None*)
Function description...

Parameters

- **item_name** (*str*) – Specified item to return data. Item names are found in the *Dfs2.items* attribute.
- **tstep_start** (*int or None, optional*) – Specify time step for element data. Timesteps begin from 0. If *None*, returns data from 0 time step.
- **tstep_end** (*int or None, optional*) – Specify last time step for element data. Allows for range of time steps to be returned, where *tstep_end* is included.Must be positive int <= number of timesteps If *None*, returns single time step specified by *tstep_start* If *-1*, returns all time steps from *tstep_start*:end

Returns **item_data** – Data for specified item and time steps.

Return type ndarray, shape (y_count, x_count, [tstep_end-tstep_start])

summary()
Prints a summary of the dfs

3.6 Units

Access the DHI MIKE21 item and unit objects

dhitools.units.**get_item**(*item*)
Return MIKE21 item code. See *available_items()* for all available MIKE21 items.

Parameters **item** (*str*) – Item name from *available_items()*

Returns **item_num** – Item number relating to input item

Return type int

dhitools.units.**get_unit**(*unit*)
Return MIKE21 unit code. See *available_units()* for all available MIKE21 units.

Parameters **unit** (*str*) – Unit name from *available_units()*

Returns **unit_num** – Unit number relating to input unit

Return type int

dhitools.units.**available_items**()
Return list of available MIKE21 items

Returns **available_items** – Available MIKE21 item names. Get the correct item code when name is used in *get_item()*.

Return type list

dhitools.units.**available_units**()
Return list of available MIKE21 units

Returns **available_units** – Available MIKE21 unit names. Get the correct unit code when name is used in *get_unit()*.

Return type list

CHAPTER 4

Features

- Interpolate multiple raster DEMS directly to **.mesh** file
- Read and analyse **.dfsu** model files
- Create **.dfsu** roughness map (or any other map) directly from **.shp** and **.mesh**
- Read and analyse **.dfs0**, **.dfs1** and **.dfs2** files

Due to depending on the **MIKE SDK DLL** libraries only Windows is supported.

CHAPTER 5

Examples

See the following Jupyter notebooks for detailed examples:

- Interpolate mesh
- Create roughness map
- Dfsu analysis - reading items, calculating statistics, plotting, interpolating to regular grid, creating new dfsu files
- Dfs012 analysis

Python Module Index

d

dhitools.units, 24

Index

A

available_items() (*in module dhitools.units*), 25
available_units() (*in module dhitools.units*), 25

B

boolean_mask() (*dhitools.dfsu.Dfsu method*), 14
boolean_mask() (*dhitools.mesh.Mesh method*), 10

C

create_dfsu() (*dhitools.dfsu.Dfsu method*), 15

D

data (*dhitools.dfs.Dfs0 attribute*), 20
del_x (*dhitools.dfs.Dfs2 attribute*), 23
del_y (*dhitools.dfs.Dfs2 attribute*), 23
Dfs0 (*class in dhitools.dfs*), 20
Dfs1 (*class in dhitools.dfs*), 21
Dfs2 (*class in dhitools.dfs*), 22
dfs_info() (*dhitools.dfs.Dfs0 method*), 20
dfs_info() (*dhitools.dfs.Dfs1 method*), 21
dfs_info() (*dhitools.dfs.Dfs2 method*), 24
dfs_time() (*dhitools.dfs.Dfs0 method*), 20
dfs_time() (*dhitools.dfs.Dfs1 method*), 21
dfs_time() (*dhitools.dfs.Dfs2 method*), 24
Dfsu (*class in dhitools.dfsu*), 13
dhitools.units (*module*), 24

E

ele_table (*dhitools.dfsu.Dfsu attribute*), 14
ele_to_node() (*dhitools.dfsu.Dfsu method*), 15
element_ids (*dhitools.mesh.Mesh attribute*), 10
element_table (*dhitools.mesh.Mesh attribute*), 9
elements (*dhitools.dfsu.Dfsu attribute*), 14
elements (*dhitools.mesh.Mesh attribute*), 9
end_datetime (*dhitools.dfs.Dfs0 attribute*), 20
end_datetime (*dhitools.dfs.Dfs1 attribute*), 21
end_datetime (*dhitools.dfs.Dfs2 attribute*), 22
end_datetime (*dhitools.dfsu.Dfsu attribute*), 14

F

filename (*dhitools.dfs.Dfs0 attribute*), 20
filename (*dhitools.dfs.Dfs1 attribute*), 21
filename (*dhitools.dfs.Dfs2 attribute*), 22
filename (*dhitools.dfsu.Dfsu attribute*), 13
filename (*dhitools.mesh.Mesh attribute*), 9

G

get_item() (*in module dhitools.units*), 24
get_unit() (*in module dhitools.units*), 24
grid_res() (*dhitools.mesh.Mesh method*), 10
gridded_item() (*dhitools.dfsu.Dfsu method*), 15
gridded_stats() (*dhitools.dfsu.Dfsu method*), 16
gridshape (*dhitools.dfs.Dfs2 attribute*), 23

I

interpolate_rasters() (*dhitools.mesh.Mesh method*), 11
item_data() (*dhitools.dfs.Dfs1 method*), 21
item_data() (*dhitools.dfs.Dfs2 method*), 24
item_element_data() (*dhitools.dfsu.Dfsu method*), 16
item_node_data() (*dhitools.dfsu.Dfsu method*), 17
items (*dhitools.dfs.Dfs0 attribute*), 20
items (*dhitools.dfs.Dfs1 attribute*), 21
items (*dhitools.dfs.Dfs2 attribute*), 22
items (*dhitools.dfsu.Dfsu attribute*), 13

L

lyr_from_shape() (*dhitools.mesh.Mesh method*), 11
lyr_to_dfsu() (*dhitools.mesh.Mesh method*), 11
lyrs (*dhitools.mesh.Mesh attribute*), 10

M

mask() (*dhitools.dfsu.Dfsu method*), 17
mask() (*dhitools.mesh.Mesh method*), 12
max_ampltiude() (*dhitools.dfsu.Dfsu method*), 17
max_item() (*dhitools.dfsu.Dfsu method*), 18

Mesh (*class in dhitools.mesh*), 9
mesh_details() (*dhitools.mesh.Mesh method*), 12
meshgrid() (*dhitools.mesh.Mesh method*), 12
min_item() (*dhitools.dfsu.Dfsu method*), 18

N

nodata_double (*dhitools.dfs.Dfs2 attribute*), 24
nodata_float (*dhitools.dfs.Dfs2 attribute*), 23
nodata_int (*dhitools.dfs.Dfs2 attribute*), 24
node_boundary_code (*dhitools.mesh.Mesh attribute*), 9
node_ids (*dhitools.mesh.Mesh attribute*), 9
node_table (*dhitools.dfsu.Dfsu attribute*), 14
node_table (*dhitools.mesh.Mesh attribute*), 9
nodes (*dhitools.dfsu.Dfsu attribute*), 14
nodes (*dhitools.mesh.Mesh attribute*), 9
num_elements (*dhitools.mesh.Mesh attribute*), 10
num_items (*dhitools.dfs.Dfs0 attribute*), 20
num_items (*dhitools.dfs.Dfs1 attribute*), 21
num_items (*dhitools.dfs.Dfs2 attribute*), 22
num_nodes (*dhitools.mesh.Mesh attribute*), 10
num_points (*dhitools.dfs.Dfs1 attribute*), 21
num_points (*dhitools.dfs.Dfs2 attribute*), 22
number_tstep (*dhitools.dfs.Dfs0 attribute*), 20
number_tstep (*dhitools.dfs.Dfs1 attribute*), 21
number_tstep (*dhitools.dfs.Dfs2 attribute*), 22
number_tstep (*dhitools.dfsu.Dfsu attribute*), 14

P

plot_item() (*dhitools.dfsu.Dfsu method*), 19
plot_mesh() (*dhitools.mesh.Mesh method*), 12
projection (*dhitools.dfs.Dfs2 attribute*), 23
projection (*dhitools.dfsu.Dfsu attribute*), 13
projection (*dhitools.mesh.Mesh attribute*), 10

R

read_dfsu() (*dhitools.dfsu.Dfsu method*), 19
read_mesh() (*dhitools.mesh.Mesh method*), 13

S

start_datetime (*dhitools.dfs.Dfs0 attribute*), 20
start_datetime (*dhitools.dfs.Dfs1 attribute*), 21
start_datetime (*dhitools.dfs.Dfs2 attribute*), 22
start_datetime (*dhitools.dfsu.Dfsu attribute*), 14
start_datetime_str (*dhitools.dfsu.Dfsu attribute*), 14
summary() (*dhitools.dfs.Dfs0 method*), 20
summary() (*dhitools.dfs.Dfs1 method*), 22
summary() (*dhitools.dfs.Dfs2 method*), 24
summary() (*dhitools.dfsu.Dfsu method*), 19
summary() (*dhitools.mesh.Mesh method*), 13

T

time (*dhitools.dfs.Dfs0 attribute*), 20
time (*dhitools.dfs.Dfs1 attribute*), 21
time (*dhitools.dfs.Dfs2 attribute*), 23
time (*dhitools.dfsu.Dfsu attribute*), 14
timestep (*dhitools.dfs.Dfs0 attribute*), 20
timestep (*dhitools.dfs.Dfs1 attribute*), 21
timestep (*dhitools.dfs.Dfs2 attribute*), 22
timestep (*dhitools.dfsu.Dfsu attribute*), 14
to_gpd() (*dhitools.mesh.Mesh method*), 13

W

write_mesh() (*dhitools.mesh.Mesh method*), 13

X

X (*dhitools.dfs.Dfs2 attribute*), 23
x_count (*dhitools.dfs.Dfs2 attribute*), 23
x_max (*dhitools.dfs.Dfs2 attribute*), 23
x_min (*dhitools.dfs.Dfs2 attribute*), 23

Y

Y (*dhitools.dfs.Dfs2 attribute*), 23
y_count (*dhitools.dfs.Dfs2 attribute*), 23
y_max (*dhitools.dfs.Dfs2 attribute*), 23
y_min (*dhitools.dfs.Dfs2 attribute*), 23

Z

zUnitKey (*dhitools.mesh.Mesh attribute*), 10