CMG(Flowgrid)

For parsing CMG input and output files to VTK or Numpy
Initializes grid object with Grid = flowgrid.CMG()

CORNER(fname, cp_type)

Builds corner point grid (*GRID *CORNER) geometry from a CMG file

Parameters

fname : str
Input file
cp_type : array_like of str
*CORNER subkeyword(s) describing how to read corner points
User should refer to input file to determine appropriate keywords
Available options are ['CORNERS'] or ['ZCORN', subkey_x, subkey_y]
subkey_x options : 'DI' ('XCORN' being added)
subkey_y options : 'DJ' ('YCORN' being added)

CART(fname)

Builds cartesian grid (*GRID *CART) from a CMG file

Parameters

fname : str
Input file

read_prop(fname, prop, add=True, mult=1)

Reads input property from .DAT file

Parameters

fname : str
Input file
prop : str
CMG keyword to read
add : bool
If grid geometry has been constructed, optionally add
False will just return array_like of property data
mult : float
Multiplicative factor for scaling data
Up to the user to determine when data needs to be scaled

Returns

data : array_like
Property array

read_ext_prop(fname, prop_title, mult=1)

Reads a property from an external file denoted by INCLUDE in .DAT
Assumes that only data is contained in file (no keywords!)

Parameters

fname : str
Input file

prop_title : str
Name for the data

mult : float
Multiplicative factor for scaling data
Up to the user to determine when data needs to be scaled

Returns

data : array_like
Property array

read_outputs(fname, out_props, refined_blocks=None)

Reads per-cell output properties for every time step in .OUT file
Properties available for reading determined by *OUTPRN *GRID
If a cell property is empty in .OUT, this will set it to null

Parameters

fname : str
File containing output properties (.OUT)

out_props : array_like of array_like of str
Specify multiple properties to read from GEM .OUT file
Sub-arrays should contain two strings:
str 1 : An output property name as it appears at a timestep declaration in a .OUT file (Time = 0 … property name)
str 2 : A custom name for that property- what if exported to Numpy or VTK files
ex) [['Pressure (kpa)', 'My pressure'], …]

refined_blocks : dict
Refinement blueprint from get_refined_blocks
Only supports 2D refinements
(under development/lightly tested, may be issues)

add_data(d, prop_title)

Adds data to a VTS structured grid

Parameters

d : numpy_array
dimensions should match that of the grid adding to

prop_title : str
Name for the data

get_refined_blocks(fname)

To be used in grids with local grid refinement (LGR)
Reads refinement blueprint for each LGR cell, which can be fed into refine_outputs

Parameters

fname : str

Input file

Returns

refine_blocks : dict

{cell idx : subdivision info, … } for REFINE keywords in fname

refine_outputs(fp, refined_blocks)

***UNDER DEVELOPMENT/LIGHTLY TESTED, MAY HAVE ISSUES
Only works with 2D refinements

```
Refined grid in 39,27,81    Refined grid in 10,28,81
-------------------------   ----------------------------------------
All values are 0.113              I =  1        2        3
                             J=   1   0.182    0.182    0.182
                             J=   2   0.182    0.183    0.182
                             J=   3   0.182    0.182    0.186
```

Above is an example .OUT refinement
We assume that the number of dashes is >= number of data items per line below
This allows us to split data blocks based on number of dashes

Parameters

fp : textIOWrapper

File object to read lines from
ex) with open(fname, “r”) as fp:

```
    # move file pointer
    …
    refine_outputs(fp, refined_blocks)
```

refined_blocks : dict

Refinement blueprint from get_refined_blocks

get_wells(fname)

This should be ran before any other well operations are performed
Builds a Wells object that contains dict of well init properties

Parameters

fname : str

File containing well init information

Returns

Wells : Wells object

Well info read here assigned to Wells.wells

export_grid(vtk_fname='GRID', toVTK=True, toNumpy=True)

Export grid information to Numpy or VTK files after all data has been read

Parameters

vtk_fname : str

Output file name for generated VTK files

toVTK : bool

Optionally export grid data to VTK files

toNumpy : bool

Optionally export grid data to Numpy files

export_wells(w, title)

Exports well dictionaries as a Numpy file

Parameters

w : dict

Well data, can be fetched from two places:

Wells.wells for well location, type, constraints

Wells.read_outputs() for well response/outputs over time

title : str

Name for the data

Wells

Contains useful methods for working with wells
Should not be called explicitly, use Grid.get_wells() to create

read_output(fname, keys, subkeys)

Reads well response/output information for all time steps
The properties being read are located at the 'GEM FIELD SUMMARY' at each time step

Parameters

fname : str
File containing well output properties (.OUT)

keys : array_like of str
These are the property titles, which act as a header for the properties (subkeys) which are to be read
These can be gathered by locating a 'GEM FIELD SUMMARY' section
ex) ['Well Pressures', 'Inst Surface Production Rates']

subkeys : array_like of array_like of str
These are the actual properties for which to read data
These can be gathered by locating a 'GEM FIELD SUMMARY' section
A sub-array index should match its corresponding key index
(make sure this order is correct!)
ex) This example goes along with the keys example above:
[['Bottom Hole', 'Drawdown'], ['Oil', 'Water', 'Gas']]

Returns

well_out : dict
Output data for all keys/subkeys for all wells for all time steps

build_cylinders(vtk_fname, zscale=1, radius=20)

***Lightly tested, might be issues
Prepares Paraview script for automatically creating cylinders to represent wells
Generated Paraview script can be executed in Paraview Python Shell after VTK grid has been loaded
Grid.get_wells() must be called first to gather well info and create wells object
Does not currently support complex well geometry

Parameters

vtk_fname : str
Should be the same as that provided to export_grid(vtk_fname)

zscale : float
Should match any Z-scaling (planning to be) applied to the grid in Paraview

radius : float
Cylinder radius

Returns

Python script for creating cylinders at well locations in Paraview