Groundwater Data Utilities

Part C: Programs Written for Unstructured Grid Models

Watermark Numerical Computing

March, 2016

PREFACE

The utilities described herein were originally documented in Part B of the Groundwater Data Utilities manual. However as their number increased, it was decided to include them in a manual of their own. This manual can be regarded, therefore, as an extension of Part B of the manual to the Groundwater Data Utility suite. It follows the same protocols, and the file types referred to herein are described in Part A of the manual.

Programs are discussed in alphabetical order. Most were written by me. However a number were written by Mark Gallagher. Hopefully his contribution will increase in the future.

The PLPROC utility, described in its own manual, comprises part of the software toolkit that is being developed to expedite use of PEST in an unstructured grid environment. It replaces many of the traditional Groundwater Data Utilities, particularly those associated with pilot points. It can be downloaded from the PEST web site.

While the writing of PLPROC, and of the utilities described herein, was prompted by the release of MODFLOW-USG, their use is not restricted to MODFLOW-USG. They can be used in conjunction with regularly-gridded MODFLOW models, and with other models (structured or unstructured) such as TOUGH and FEFLOW.

Note also that many of the utilities described in Part B of this manual are just as useful in the unstructured grid context as in the structured grid context. These include:

- Programs that manipulate data contained in bore sample files.
- Utilities such as PESTPREP, PESTPREP1 and PESTPREP2 which automate PEST input dataset preparation.
- Programs such as GENREG and PPCOV which assist in provision of regularisation constraints to pilot point parameters.

I wish to gratefully acknowledge support that was provided to me for the writing of some of the utilities documented herein. In particular, I would like to acknowledge the following.

- The National Centre for Groundwater Research and Training, Flinders University, Australia.
- The Queensland Hydrology Unit, Science Delivery Division, Department of Science, Information Technology, Innovation and the Arts (DSITIA).
- Office of Groundwater Impact Assessment, Queensland Government Department of Natural Resources and Mines.
- Australian Groundwater and Environmental Consultants, Brisbane, Australia.

Ongoing support from EMS-I is also gratefully acknowledged and appreciated.

John Doherty

UNSTRUCTURED GRID GROUNDWATER DATA UTILITIES
See Part B of the manual for the Groundwater Data Utilities for listing and documentation of other members of the suite.

arr2ndtf	Reads a list of numbers (i.e. an array) from a text file. After associating these numbers with nodes of an unstructured grid, it re-writes them in node data table format.
gridgen2gsf	Writes an unstructured grid specification file based on a layer quadtree grid produced by the USGS GRIDGEN program.
pt2vtk	Reads coordinates of points, and data pertaining to points, from a tabular data file. Writes a VTK file for visualization of point data.
tab2ndtf	Reads a tabular file containing data for at least some nodes of an unstructured grid. Writes data in node data table format, so that it is readable by USG2VTK.
usg2vtk	Reads a MODFLOW-USG GWF grid or CLN network specification file and writes a VTK file for viewing of the grid/network and (optionally) data associated with grid/network nodes.
usg2vtk1	Similar to USG2VTK but represents unstructured cells using the more flexible VTK POLYHEDRON scheme.
usgaddcoord	Adds coordinates to a tabular data file in which nodes feature as the first column.
usgaddzcoord	Adds *z* coordinates to a tabular data file which includes *x* and *y* coordinates. *z* coordinates are obtained through interpolation from a MODFLOW-USG grid.
usgarrdet	Lists the contents of a MODFLOW-USG head/drawdown or cell-by-cell flow term binary output file.
usgarrdet_dbl	Same as USGARRDET, but reads double precision binary files.
usgbin2csv	Reads a binary head/drawdown file generated by MODFLOW-USG. Tabulates heads for user-specified nodes or layers as a series of CSV files that can be imported to PARAVIEW and prepared for animation.
usgbin2tab_h	Reads a binary head/drawdown file generated by MODFLOW-USG. Tabulates values of head or drawdown at all nodes at a user-specified simulation time.
usgbin2tab_h1	Identical to USGBIN2TAB_H1 but adds a column containing node layer numbers.
usgbud2smp	Reads a binary cell-by-cell flow term file written by MODFLOW-USG. Records flow rates in each of a number of user-specified zones in bore sample file format.

usgbud2smp1	Performs similar function as usgbud2smp but obtains information on which budget terms to extract from a MODFLOW-USG binary budget file from a different type of text input file.
usgdbl2sgl	Reads an unstructured grid binary head/drawdown or budget file written by a double precision compiled version of MODFLOW-USG. Converts to single precision.
usgdualmodel	Reads unstructured grid specification files for a coarse and a fine model. The two are assumed to be compatible, in that fine model cells "tile" coarse model cells and do not overlap coarse model cell boundaries. Writes a file containing node associations between the two models for use by upscaling and downscaling software.
usggridlay	Reads an unstructured grid specification file. Writes a MIF/MID file, BLN file and XYZ file for a user-specified model layer.
usgmod2array	Similar to MOD2ARRAY but works with MODFLOW-USG instead of MODFLOW. Modifies a MODFLOW-USG input dataset so that user-specified arrays are read from an external file.
usgmod2obs	Undertakes spatial and temporal interpolation from a binary head/drawdown file written by MODFLOW-USG to the locations and times of observations.
usgmod2obs1	Similar to USGMOD2OBS, but can read a binary file written in double precision. Its method of identifying dry/inactive cells is also slightly different.
usgmod2smp	Undertakes spatial interpolation from a binary head/drawdown file written by MODFLOW-USG to user-specified sites.
usgndtf2mif	Reads an unstructured grid specification file and a node data table file containing information on the nodes of an unstructured grid. Writes node data in MIF/MID format, ready for import into a GIS.
usgorthchek	Checks whether all model cells are square or rectangular.
usgprop2tab1	Reads a series of layer property arrays from a MODFLOW-USG input file. Rewrites these in tabular format in nodal sequence.
usgprop2tab2	Reads a series of layer property arrays from a MODFLOW-USG input file. Rewrites these in tabular format in nodal sequence. Allows more flexible representation of array data in MODFLOW-USG input file than USGPROP2TAB1.
usgptingrid	Finds unstructured grid cells in which user-supplied points lie. An (x,y) coordinate and layer number is associated with each point.
usgquadfac	Calculates interpolation factors from a regular or quadtree-refined grid to a set of observation wells for the use of USGMOD2OBS and USGMOD2SMP.

ARR2NDTF

Function of ARR2NDTF

ARR2NDTF (which stands for "ARRay to Node Data Table File) reads a list of numbers from a text input file. These are read using the free format protocol, and so can be arranged in a list or as sequential numbers on one or many wrapped lines. The numbers can be real or integers. On the assumption that these numbers can be assigned to sequential nodes of an unstructured grid (starting at an arbitrary node), ARR2NDTF rewrites them in node data table file format. As such, they can be used by programs such as USG2VTK for visualization and display purposes, of by PLPROC for unstructured grid parameterization purposes.

Using ARR2NDTF

As for all other programs in the Groundwater Data Utilities suite, a response of "e" or "E" (for "escape") to any prompt takes the user back to the previous prompt.

ARR2NDTF commences execution by asking for the name of the file which it must read, and then asking how many lines of this file must be ignored before encountering the start of the list of numbers that must actually be read. The prompts are:

```
Enter name of file containing data array:
Skip how many lines at the top of this file?
```

ARR2NTDF next acquires information that allows it to place the numbers which it is about to read into an array that pertains to a MODFLOW-USG or other model. It asks:

```
Enter starting node number in array:
Enter final node number in array:
Enter total number of nodes in grid:
```

It is the user's responsibility to ensure the integrity of these numbers. The last is important, because every node in the entire grid must feature in a node data table file, even if many are associated with default numbers that imply missing data.

```
Enter a label for this data (12 characters of less):
```

The label must be 15 characters or less in length and should contain no spaces. This label will head the column of the node data table file which ARR2NDTF writes.

ARR2NDTF's next question is:

```
Does the array contain integer or real data?  [i/r]:
```

If the starting node number is not 1 and the final node number is not equal to the number of nodes in the grid, ARR2NDF needs to know how it should fill elements of the nodal array that are missing from the list of numbers that it is about to read. So it asks:

```
Enter data value to assign to uncited nodes:
```

ARR2NDTF's final prompt is for the name of the node data table file which it must write:

```
Enter name for node data table file:
```

After acquiring this name, it reads its input file and writes its output file. If any errors are encountered during this process it reports these to the screen and then ceases execution.

Numbers read by ARR2NDTF from its input file must be separated from each other by whitespace (including a new-line character) or commas. They are read using the FORTRAN free-field protocol. Hence they can be arranged sequentially over one or many lines, or in a single list with one number to a line. No characters other than whitespace or a comma must separate them. Also, they must pertain to sequential nodes of an unstructured grid.

Uses of ARR2NDTF

The format of a node data table file is described in Part A of this manual. It is read by programs such as USG2VTK in order to associate an integer or real value with every node of the grid. It can also be read by programs such as PLPROC (a parameter pre-processor for unstructured grid models).

Specifications of a node data table file allow it to contain more than one column of integer and/or real data. Different columns can contain data of different types. ARR2NDTF writes a node data table file containing only two columns, the first being node numbers and the second being the data read from its input file. Multi-column node data table files can easily be created from ARR2NDTF-generated files through column cutting and pasting, as is allowed by many text editors.

See Also

See also TAB2NDTF, USG2VTK, USGPROP2TAB1, USGPROP2TAB2, USGBIN2TAB_H and USGBIN2TAB_H1.

GRIDGEN2GSF

Function of GRIDGEN2GSF

GRIDGEN2GSF facilitates three-dimensional visualization of a MODFLOW-USG model whose grid is constructed through quadtree refinement of a traditional structured grid MODFLOW model using the USGS GRIDGEN program. See Lien et al (2014) for a description of the latter program.

GRIDGEN2GSF writes an unstructured grid specification file using two different methodologies for definition of grid cell vertices. It also writes a node data table file linking nodes within the quadtree-refined unstructured grid generated by GRIDGEN to rows, columns and layers of the original MODFLOW grid. These files can be used by programs such as USG2VTK and USGGRIDLAY for building VTK and SURFER files through which the new quadtree-refined grid can be visualized and displayed. Programs such as PLPROC can also use the grid specification file to assist in pilot point parameterization of the quadtree-refined grid.

Using GRIDGEN2GSF

General

GRIDGEN2GSF reads a GRIDGEN-produced file (and files cited therein) which contains specifications for a quadtree-refined grid. This file must contain a single QUADTREE block and a single MODFLOW_GRID block, with the latter citing the former through its MODFLOW_GRID keyword. These blocks can be provided in any order. If other blocks appear in this file they are ignored.

Both of these blocks will probably cite other files using the OPEN/CLOSE protocol associated with various keywords. These files provide data such as structured or unstructured cell or node elevations. GRIDGEN2GSF reads these files too. Of particular importance is the quadtree structure file cited with the QUADTREE_FILE keyword of the QUADTREE block. This file relates quadtree-refined nodes to cells of the original structured MODFLOW model.

As is discussed by Lien et al (2014), GRIDGEN allows specification of quadtree-refined grid node top and bottom elevations in three different ways, namely through the REPLICATE, INTERPOLATE and ASCIIGRID options associated with the TOP LAYER and BOTTOM LAYER keywords of the QUADTREE_BUILDER block. Elevations assigned by GRIDGEN to grid nodes using any of these options are stored in files cited in the TOP LAYER and BOTTOM LAYER keywords of the QUADTREE block. GRIDGEN2GSF reads these latter files. From the data contained within these files it calculates the *z* coordinates of all unstructured grid nodes as the average of the bottom and top elevations calculated for each pertinent cell. It then records these node *z* coordinates in the second section of the unstructured grid specification file which it writes.

The assignment of elevations to the *vertices* of unstructured cells is a little more complicated than this however. Visualization of cells requires that the *x*, *y*, and *z* coordinates of all cell vertices be provided; this indeed is an important component of the unstructured grid specification file, vertex coordinates being listed in the first part of this file. GRIDGEN2GSF calculates *z* coordinates for these vertices in one of two user-specified ways.

If the first option is employed, vertex elevations are calculated through bilinear interpolation from structured model cell top and bottom elevations provided in the MODFLOW_GRID block. In writing the unstructured grid specification file, GRIDGEN2GSF then links all cells which use a common vertex to that vertex. This strategy is "parsimonious", as it drastically reduces the number of vertices that must be recorded in the unstructured grid specification file, as no vertex is recorded more than once, regardless of the number of cells to which that vertex belongs. This, in turn, induces USG2VTK to write a smaller VTK file, and allows visualization software to display the grid with a smaller numerical burden.

If the second option for vertex *z* coordinate calculation is adopted, each cell in the quadtree-refined unstructured grid is assumed to be a rectangular prism with a horizontal top and bottom surface. The elevations of these surfaces are read from TOP LAYER and BOTTOM LAYER data cited in the QUADTREE block. With adoption of this protocol, vertex definition becomes "non-parsimonious", in that each unstructured model cell is assigned eight individual vertices, regardless of whether a vertex is shared by a neighbouring cell. For large and complex grids, this can result in the production of very large unstructured grid specification files by GRIDGEN2GSF, very large VTK files by USG2VTK and slow display and manipulation of the unstructured grid by three-dimensional visualization software.

GRIDGEN2GSF also writes a "node data table file". As is described elsewhere in this manual, this contains node data in tabular format. In the node data table file written by GRIDGEN2GSF the first column contains node numbers, arranged in increasing order. The second, third and fourth columns contain the column, row and layer number of the original structured MODFLOW grid in which each cell of the quadtree-refined grid lies.

Prompts and Responses

Typical GRIDGEN2GSF prompts and responses are as follows.

```
Program GRIDGEN2GSF writes an unstructured grid specification file based on
   a quadtree-refined grid generated using the USGS GRIDGEN program.

 Enter file containing MODFLOW_GRID and QUADTREE blocks: model.dfn
 Inactive threshold for layer top/bot elevs: (<Enter> if none): 999.0

 Name for vertex-parsimonious USG spec file (<Enter> if none): model1.spc
 Name for non-vertex-parimonious USG spec file (<Enter> if none): model2.spc
 Name for ICOL/IROW/ILAY node data table file (<Enter> if none): nodes.ndt
 - file SS18_2_L1top.DAT read ok.
 - file Layer1.DAT read ok.
 - file Layer2.DAT read ok.
 - file Layer3.DAT read ok.
 - file Layer4.DAT read ok.
 - file Layer5.DAT read ok.
```

```
- file Layer6.DAT read ok.
- file Layer7.DAT read ok.
- file Layer8.DAT read ok.
- file temp.dfn read ok.

- reading structure file for first time...
- maximum grid refinement level = 4
- assigning coordinates to grid of maximum refinement...
- reading structure file for second time...
- file grid02qtg.tsf read ok
- reading USG nodal top/bottom elevation arrays...
- file grid02qtg.top1.dat read ok.
- file grid02qtg.top2.dat read ok.
- file grid02qtg.top3.dat read ok.
- file grid02qtg.top4.dat read ok.
- file grid02qtg.top5.dat read ok.
- file grid02qtg.top6.dat read ok.
- file grid02qtg.top7.dat read ok.
- file grid02qtg.top8.dat read ok.
- file grid02qtg.bot1.dat read ok.
- file grid02qtg.bot2.dat read ok.
- file grid02qtg.bot3.dat read ok.
- file grid02qtg.bot4.dat read ok.
- file grid02qtg.bot5.dat read ok.
- file grid02qtg.bot6.dat read ok.
- file grid02qtg.bot7.dat read ok.
- file grid02qtg.bot8.dat read ok.
- eliminating unused vertices...
- writing parimonious USG spec file...
- file model1.spc written ok.
- writing non-parsimonious USG spec file...
- file model2.spc written ok.
- writing NCOL/NROW/NLAY node data table file...
- file nodes.ndt written ok.
```

The following should be noted.

- If the original structured MODFLOW grid has inactive cells, and if top and bottom elevations assigned to these cells are incorrect, then the number supplied in response to the second of GRIDGEN2GSF's prompt should allow the user to identify such cells. Any cell in which the absolute value of the bottom or top elevation is greater than the absolute value of the user-supplied threshold is not used in interpolating structured MODFLOW cell top and bottom elevations to the vertices of the quadtree-refined grid if the parsimonious vertex option is employed. (As stated above, for the non-parsimonious option, unstructured cell vertex elevations are equated to the cell top and bottom elevations read from files cited in the QUADTREE block.) It is the user's responsibility to ensure that no active cells of the structured MODFLOW model are assigned elevations that are above this threshold.

- Inactive nodes, and the vertices associated with them, are omitted from the grid specification and node data table files written by GRIDGEN2GSF.

- The node numbering sequence recorded in the grid specification file written by GRIDGEN2GSF is the same as that defined in the quadtree grid structure file written by the GRIDGEN program.

- If the parsimonious vertex option is employed, then vertices in quadtree-refined areas of the model grid may be calculated with reduced accuracy as

these are interpolated from original structured MODFLOW grid cell centres rather than from node elevations that GRIDGEN may read if the ASCIIFILE option is adopted for assignment of refined grid node elevations. Furthermore, if a highly refined cell borders a larger, less refined cell, vertices for the former are not decreed to be shared by the latter. This strategy allows all refined cells to possess 8 vertices regardless of their refinement level, and of the refinement level of their (possibly more refined) neighbours. (This is a requirement of the USG2VTK program.) This may result in slight vertical misalignment of differently-refined cell edges where layer topography is variable.

Uses of GRIDGEN2GSF

GRIDGEN2GSF is used primarily to facilitate 3D visualization of quadtree-refined grids constructed by the GRIDGEN program. However the unstructured grid specification file produced by GRIDGEN2GSF can also be used by programs which perform tasks other than visualization. One such program is the PLPROC parameter list processor, which can be used for implementation of pilot points parameterization in an unstructured grid model.

See Also

See also USG2VTK, USGGRIDLAY

References

Lien, J-M., Liu, G. and Langevin, C.D., 2014. GRIDGEN Version 1.0: A Computer Program for Generating Unstructured Finite-Volume Grids. U.S. Geological Survey Open-File Report 2014-1109.

PT2VTK

Function of PT2VTK

Program PT2VTK reads a tabular data file. One of the data columns within this file should contain the *x* (i.e. east) coordinates of a set of points, while another should contain the *y* (i.e. north) coordinates of these points. Optionally, another column can contain point *z* coordinates, while other columns may contain integer or real data pertaining to the points. PT2VTK writes a VTK file which can be imported into appropriate visualization software for displaying the points and the data associated with them.

Using PT2VTK

PT2VTK commences execution by asking for the name of the tabular data file which it must read:

```
Enter name of tabular data file:
```

Next it asks how many lines should be skipped before the point coordinates and data can actually be read. More often than not, the answer to this question will be "1", as the first line of a tabular data file contains column headers. All skipped lines are ignored by PT2VTK. The prompt is:

```
Skip how many lines at the top of this file?
```

Next it asks:

```
Enter column containing X coordinates:
Enter column containing Y coordinates:
Enter column containing Z coordinates (<Enter> if none):
```

If no *z* coordinates are provided for the points, simply press <Enter> in response to the last of the above prompts (or "e" to return to the previous prompt). If <Enter> is indeed the response, then PT2VTK will assign the same *z* coordinate to all points featured in the tabular data file, this being supplied in response to the question:

```
Enter Z coordinate of all points:
```

Next PT2VTK offers to read other columns of the tabular data file, these presumably containing data pertaining to the points. The offer can be declined by responding with <Enter> only.

```
Enter column number from which to read point data (<Enter> if no more):
```

If a column number is provided, two further questions follow:

```
Enter a label for this data (12 characters of less):
Is this integer or real data?  [i/r]:
```

This three-question cycle is then repeated until the user indicates that all necessary data have been read by responding to the first of these questions with <Enter>.

Finally PT2VTK asks for the name of the "legacy" VTK file which it must write:

```
Enter name for VTK file:
```

It then writes the file.

Uses of PT2VTK

As is described above, PT2VTK can be used to provide the means of adding point-based data to a picture produced by software which can read a VTK file. This can be particularly useful where the picture is of a model, and the points are pilot points used to parameterize the model.

See Also

See also USG2VTK.

TAB2NDTF

Function of TAB2NDTF

Program TAB2NDTF reads a tabular data file containing node numbers together with data associated with those nodes. Note that there is no need for this file to cite all nodes pertaining to a particular model. It re-writes node data that is recorded in selected columns of this table in "node data table" format (see part 1 of this manual for a description of this file type). The latter cites all nodes of an unstructured grid; default values are provided for nodes that are not cited in the original file. A node data table file is readable by USG2VTK. Hence the nodes and data in the original tabular data file can be displayed in conjunction with other aspects of the unstructured grid.

Using TAB2NDTF

As for any other member of the Groundwater Data Utilities suite, a user may respond to any of TAB2NDTF's prompts with "e" (for "Escape") followed by <Enter>. This will take TAB2NDTF execution back to the previous prompt.

TAB2NDTF commences execution with the prompt:

```
Enter name of tabular data file:
```

The "tabular data file" may, in fact, be part of a MODFLOW-USG input file for a package that requires that its data be supplied in tabular format. These packages include the DRAIN, RIVER, GHB and many other packages. One of the columns in this file must contain node numbers. These do not need to be in sequential order; nor do all nodes within the grid need to be cited in this column. Other columns contain data associated with the cited nodes.

Initial lines within the tabular data file can be skipped if an integer greater than zero is supplied in response to TAB2NDTF's next prompt, which is:

```
Skip how many lines at the top of this file:
```

Then it asks:

```
Enter column containing node numbers:
```

For most MODFLOW-USG package input files this will be the first column. However TAB2NDTF makes no assumptions about this matter, so that it can be used to read other types of files containing nodal data.

Then follows a cycle of four prompts through which TAB2NDTF is informed of what nodal data it must read from the its tabular data input file. This cycle is repeated until broken. It is broken by responding with <Enter> to the first prompt in the cycle. However at least one cycle must be completed, so that at least one type of node-specific data must be read from the input file. The cycle of prompts is as follows:

```
Enter column number from which to read node data (<Enter> if no more):
Enter a label for this data (12 characters of less):
Is this integer or real data?  [i/r]:
Enter data value to assign to uncited nodes:
```

Next TAB2NDTF asks for the name of the node data table file which it must write:

```
Enter name for node data table file:
```

Its final prompt is:

```
Enter total number of nodes in model grid:
```

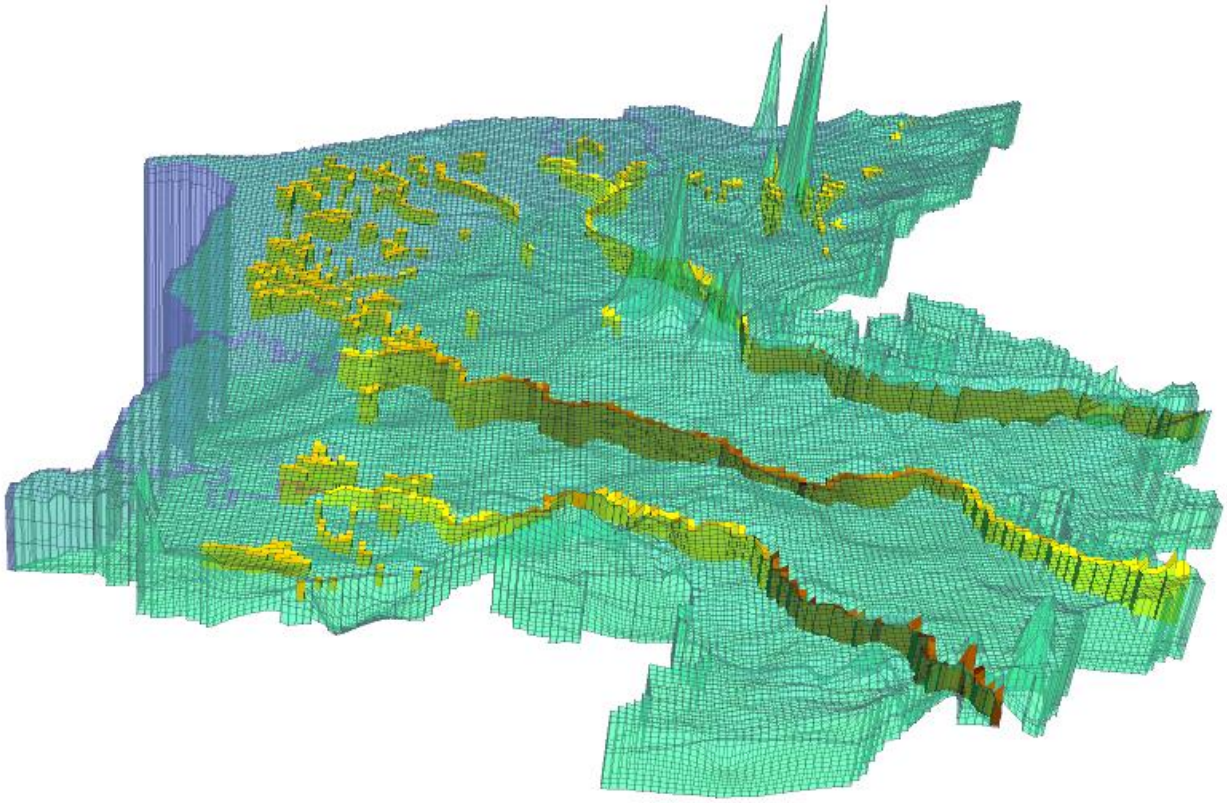
It is very important that this number be supplied correctly, for correct writing of the node table data file depends on it. This is because very node of the grid must be cited in this file. If unsure of this number, it can be found in the DISU file (see MODFLOW-USG documentation). If the "grid" is in fact a CLN network, the number of CLN nodes can be found in the CLN package input file.

After being informed of this number, TAB2NDTF reads its input tabular data file and writes its output node data table file. If any errors are found in the former, these are reported to the screen.

Uses of TAB2NDTF

TAB2NDTF was written in order to facilitate display of MODFLOW-USG data associated with certain of its packages, namely those which read data in tabular format. Input data associated with a certain stress period can be extracted from the MODFLOW-USG input file associated with that package and placed into a file of its own using the cutting and pasting functionality of a text editor. TAB2NDTF can then read this file, extracting stress-period-specific data from it. The data can then be recorded in a format ready for the use of USG2VTK.

The figure below shows the domain of a model with fixed head cells coloured blue and most other cells coloured green. Both are displayed as transparent. Within the grid MODFLOW-USG RIVER cells are coloured as brown to yellow, the colour reflecting the DRAIN conductance assigned to each such cell. The locations and properties of these cells are readily visible from this picture.



See Also

See also PT2VTK, USG2VTK, USGPROP2TAB1, USGPROP2TAB2, USGBIN2TAB_H and USGBIN2TAB_H1.

USG2VTK

Function of USG2VTK

Program USG2VTK writes a "legacy" VTK file based on an unstructured grid or CLN network. Optionally this file can include data associated with nodes comprising the grid or network. This allows the grid or network to be coloured according to values associated with these nodes when the grid is visualized using a modeller's chosen visualization platform.

"VTK" stands for "visualization toolkit". The visualization toolkit is an open-source, freely available software system for 3D computer graphics, image processing and visualization. Files used by this software are supported by many visualization packages which are built from this toolkit. One of these packages is the public domain PARAVIEW package. See www.paraview.org for full details of this package. It can import VTK files written by USG2VTK, thereby allowing three dimensional display of MODFLOW-USG model components, as well as grid/CLN properties and system states, the latter having been computed by MODFLOW-USG.

The protocols for MODFLOW-USG grid and CLN specification files are presented in part A of this manual.

Using USG2VTK

USG2VTK begins execution by asking for the name of a MODFLOW-USG grid specification file. This file can specify the geometry of a MODFLOW-USG grid; alternatively it can specify the geometry of a MODFLOW-USG CLN network. USG2VTK knows the difference between the two from the contents of the grid specification file; see part A of this manual. It asks:

```
Enter name of MODFLOW-USG grid specification file:
```

Next USG2VTK reads part of the file, establishing its specifications and checking its integrity. It then asks whether any so-called "scalar" data is to be recorded in the VTK file. "Scalar" is VTK terminology. In the present context it means data associated with the nodes of the unstructured grid or CLN network. Its prompt is:

```
Record scalar data in VTK file? [y/n]:
```

If the answer to this question is "n", then USG2VTK simply writes the VTK file and ceases execution. Alternatively, if the answer is "y", USG2VTK asks for the name of the file from which it must read data pertaining to grid or CLN nodes

```
Enter name of tabular file from which to read node data:
```

The contents of the file whose name is supplied in response to this prompt must meet certain specifications. These are:

- It must contain columns of space- or tab-delimited data.
- The first line of the file must contain column headers.
- There must be as many elements in each column as there are nodes of the grid or CLN.
- Data elements within each column are associated with nodes starting with node number 1, and with an increment of 1 proceeding down the column.

Part of a tabular data file is presented in the figure below.

```
ZONE_INDEX     Kh
   1          1.034e-1
   1          2.518e-1
   1          3.322e-1
   1          2.981e-1
   1          1.850e-1
   1          5.200e-1
   2          7.453e-1
   2          9.564e-1
   3          3.817e-1
   3          4.339e-1
   3          5.964e-1
   3          6.072e-1
   3          5.834e-1
   3          2.558e-1
   etc
```

Part of a tabular data file containing nodal data.

After opening the file named by the user, USG2VTK prompts:

```
Enter column number of data to read (<Enter> if no more):
Is this integer or real data? [i/r]:
```

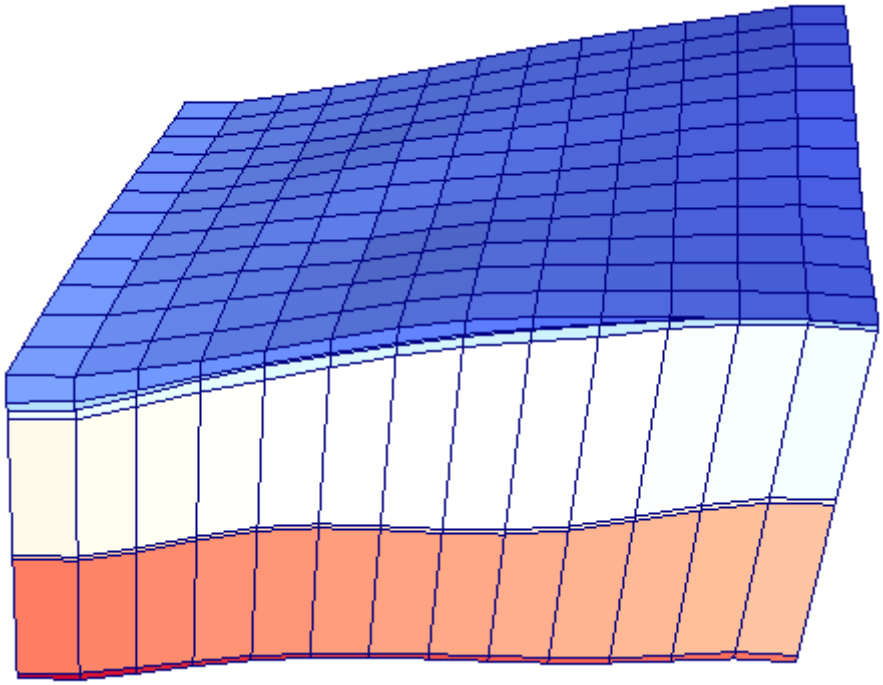
Columns are numbered from 1, starting at the left. Provide the indices of as many columns as desired. These indices can be in any order. Once enough indices have been provided, press <Enter> instead of a column number. USG2VTK will then write the VTK file.

As for all other members of the Groundwater Data Utilities suite, a response of "E" or "e" to any prompt instructs USG2VTK to return to the preceding prompt.

Uses of USG2VTK

The figure below shows a picture generated by PARAVIEW. PARAVIEW obtained the grid design specifications and nodal data required for generation of this plot from a USG2VTK-generated VTK file. PARAVIEW provides high level display functionality, including sectioning, selection-by-threshold, display of multiple data types, etc.

Tables of USG2VTK-readable node data can be generated by programs such as USGBIN2TAB_H, USGBIN2TAB_H1, USGPROP2TAB1 and USGPROP2TAB2.



See Also

See also USG2VTK, USGBIN2TAB_H, USGBIN2TAB_H1, USGPROP2TAB1 and USGPROP2TAB2.

USG2VTK1

Function of USG2VTK1

USG2VTK1 is very similar in its specifications and operations to USG2VTK. However it differs in two important respects. These are as follows.

- USG2VTK1 cannot process data contained in an unstructured grid specification file which contains specifications for a CLN network; CLN data must be processed by USG2VTK.

- In writing a legacy VTK file, USG2VTK1 represents cells using VTK POLYHEDRA (cell type 42). These are more complicated than VTK HEXADRA (cell type 12) used by USG2VTK. However, unlike USG2VTK, USG2VTK1 can accommodate cells produced by software packages such as ALGOMESH which generate prismatic cells of arbitrary cross-sectional shape.

Using USG2VTK1

Prompts and responses of USG2VTK1 are identical to those of USG2VTK.

Uses of USG2VTK1

Where cell types are not rectangular in cross section, then USG2VTK1 should be used instead of USG2VTK. USG2VTK1 can accommodate cells whose upper and lower boundaries are polygons of any shape, as long as the upper and lower polygons have the same number of sides. The number of sides can differ between cells. These specifications allow it to read and process unstructured grid specification files written by the ALGOMESH mesh generator. See

http://www.hydroalgorithmics.com/index.php/software/algomesh

for details of this software package.

For USG2VTK1 to work properly with polyhedral cells, there are certain specifications which a MODFLOW-USG grid specification file must possess which are not listed in Part A of this manual. These additional specifications are now provided.

The last (and often longest) part of an unstructured grid specification file lists every node in the unstructured grid, one to a line. Following the node number on each line are the node's coordinates, and then the vertices that define the cell in which the node lies. The protocol for each such line is:

inode x y z lay m (ivertex(i),i=1,m)

where

inode	is the node number (these should be listed in increasing order in the file);
x, *y* and *z*	are the coordinates of the node;
lay	is the layer number to which the node belongs;
m	is the number of vertices defining the cell which contains the node; and
vertex(i)	is the vertex index of the cell's *i*'th vertex; vertex coordinates are provided in the first part of the MODFLOW-USG grid specification file.

USG2VTK1 requires that *m* be an even number. This follows from the fact that the upper and lower polygons defining a polyhedral cell must have the same number of vertices. The following vertex list must cite vertices pertaining to the cell's upper polygon first, and then those pertaining to the cell's lower polygon next. Vertices must be cited in either clockwise or anti-clockwise order; thus neighbouring nodes in the list must be neighbouring nodes in respective polygons. Vertices in the upper and lower polygon sublists must correspond to each other; that is, the first-listed upper vertex is joined to the first-listed lower vertex, the second-listed upper vertex is joined to the second-listed lower vertex, etc.

See Also

See also USG2VTK.

USGADDCOORD

Function of USGADDCOORD

USGADDCOORD reads a tabular data file, such as a node table data file in which nodes comprise the first column. A header line is also expected in this file. It adds four columns to the file between the existing first and second columns; the latter then becomes the fifth column. These new columns comprise the *x*, *y*, *z* and layer coordinates of respective nodes featured in the first column. This table can then be easily imported into display, GIS or visualization software. Note that nodes listed in the node data table data file do not need to be complete, and can be listed in any order.

Using USGADDCOORD

USGADDCOORD is easy to use. Prompts are as follows:

```
Enter name of MODFLOW-USG grid specification file:

Enter name of node data table file:
Enter name for new node data table file:
```

As stated above, the tabular data file read by USGADDCOORD must possess a text header. Entries in this header after the first are pushed to the right and the headers "NODE_X", "NODE_Y", "NODE_Z" and "LAYER" are inserted. In subsequent lines the *x*, *y*, *z* coordinates and the layer number of the node featured in the first column are inserted. Once again, other columns are pushed to the right.

Function of USGADDCOORD

Once nodal data is accompanied by node coordinates, this data can be visualized and displayed. Suppose, for example, that you place DRAIN data for a particular stress period into a text file of its own, while adding appropriate headers to this data. USGADDCOORD can then be used to add coordinates and the layer number corresponding to each drain node. The resulting table can then be imported into PARAVIEW where so-called PARVIEW "filters" can be applied to create points with drain elevation as the *z*-coordinate for each point. Drains can then be added to a model visualization in their correct position. Colouring can be by layer, conductance, or whatever is most fitting for the plot.

See Also

See also USG2VTK, USG2VTK1 and PT2VTK.

USGADDZCOORD

Function of USGADDZOORD

Program USGADDZCOORD reads a tabular data file containing information pertaining to a set of points. The first column of this file is assumed to be comprised of point identifiers; USGADDZCOORD treats these as character strings, but they can, of course, be numbers. A length limit of 20 characters is assumed. The second and third columns are assumed to be comprised of point *x* and *y* coordinates. If any columns follow that, then one of them can optionally include model layer numbers with which respective points are associated. (The points may, for example, be two-dimensional pilot points that are used to parameterize one or a number of model layers.)

USGADDZCOORD adds a column to this tabular data file. This will become the fourth column of the file – the column that follows that which contains point *y* coordinates. If the existing file has only three columns, this fourth column will then be the last column of the file. However if the existing file has more than three columns, then the fourth column is inserted after the third, and all other columns are then shifted to the right. This fourth column ascribes *z* coordinates to the points represented in the file. These *z* coordinates are obtained by *x*-*y* inverse-power-of-distance interpolation between vertices of an unstructured grid. Vertex *x*, *y*, and *z* coordinates are read from an unstructured grid specification file.

Using USGADDZCOORD

Program USGADDZCOORD begins execution by prompting for the name of the unstructured grid specification file that it must read. The prompt is:

```
Enter name of MODFLOW-USG grid specification file:
```

Optionally a user can supply a node data table file which informs USGADDZCOORD which nodes are inactive. Interpolation from vertices associated with inactive nodes to points in the tabular data file does not take place. The node data table file expected by USGADDZCOORD must have a header. Node numbers must appear in the first column of the file; all nodes in the model grid must be represented. The prompt is:

```
Enter name of activity node data table file (<Enter> if none):
```

Then, if a filename is provided:

```
Enter column number containing activities:
```

The column number must be 2 or greater. Integers must comprise the column. An integer value of zero signifies an inactive node.

Next USGADDZCOORD prompts for the name of an existing tabular data file:

```
Enter name of tabular data file:
```

The following are expected of this file:

- On any line, columns should be delimited by spaces or a tab.
- The first line of the file should be comprised of a header line which indicates the contents of each column. Each header should be 20 characters or less in length and not include a space.

To be sure that a header is indeed present in the tabular data file, USGADDZCOORD asks the user:

```
Does this file have a header line? [y/n]:
```

An answer of "n" to this prompt results in termination of USGADDZCOORD execution with an appropriate error message. Otherwise USGADDZCOORD next prompts:

```
Enter column number containing layer data (<Enter> if none):
```

If the response to this prompt is <Enter>, then USGADDZCOORD asks the user for a single layer number which is associated with all points in the file. The prompt is:

```
Enter layer number to which this file pertains:
```

Next USGADDZCOORD asks the user how it must interpolate from vertex coordinates provided in the grid specification file to each point that is represented in the tabular data file. The prompts are:

```
Enter power of inverse distance to use in interpolation:
Enter search radius:
Enter maximum number of points to use in interpolation:
```

A value of "2" normally constitutes a suitable response to the first of the above prompts.

Where a grid is large USGADDZCOORD's run time can be large. This run time can be shortened by not using an unnecessarily large search radius.

A small number is normally adequate for the last of the above prompts – only 4 if a grid is comprised of uniform square cells (and hence is not really unstructured at all). It needs to be slightly larger than this where grid cells change size, or can increase in number, over a short distance. It should not be too large however, for then the averaging that is involved in inverse-power-of-distance interpolation can render the *z* coordinate assigned to a point unrepresentative of its local position with respect to the grid. Note that USGADDZCOORD multiplies the above number by 2 internally to account for the fact that it uses both the upper and lower vertices associated with each model cell in performing inverse-power-of-distance interpolation. Hopefully the location ascribed to a point is then somewhere near the middle of the layer.

If a point lies outside the finite difference grid, then assignment of a *z* coordinate can become problematical. USGADDZCOORD does not detect this condition; it simply applies inverse-power-of-distance interpolation to all points cited in the tabular data

file regardless of their position. If the number of points used in the interpolation process is not large, then the *z* coordinate ascribed to an outside-of-grid point will approximate those of the closest grid vertices to it.

Uses of USGADDZCOORD

USGADDZCOORD can facilitate display of two-dimensional pilot points in a three-dimensional visualization package such as PARAVIEW. The tabular data file that it produces can be imported into PARAVIEW as a table. This table can then be subjected to the "table to points" filter, after which the points can then be plotted in three dimensions. Though the *z* coordinates of two-dimensional pilot points play no part in their usage (this being to interpolate properties to the nodes of a grid), it is visually pleasing to ascribe them to locations at which they "conceptually situated" in performing this interpolation when viewing them and the model grid together.

See Also

See also USG2VTK, USG2VTK1 and USGADDCOORD.

USGARRDET

Function of USGARRDET

USGARRDET reads a binary output file written by MODFLOW-USG. This can be a head/drawdown or cell-by-cell flow term output file; however the grid must be of the unstructured type. USGARRDET lists the contents of this file to an ASCII file whose name is provided by the user.

USGARRDET does for MODFLOW-USG what ARRDET does for MODFLOW. However it is more powerful than ARRDET in that it is able to read cell-by-cell flow term (i.e. budget) files as well as head/drawdown files.

Using USGARRDET

Use of USGARRDET is simple. Typical prompts and responses are as follows.

```
Enter name of unformatted MFUSG-generated file: haughton.cbb
Is this a head/drawdown or cell-by-cell flow file?  [h/c]: c

Enter name for output file: temp.dat

- 136 arrays read from file haughton.cbb.
- file temp.dat written ok.
```

Though simple to use, a number of things can go wrong when running USGARRDET. For example the user may ask USGARRDET to read a binary MODFLOW-USG output file that pertains to a structured rather than an unstructured grid. He/she may also inform USGARRDET that the file contains head or drawdown data when, in fact, it contains budget data, or vice versa. In any of these cases the headers to the various data arrays stored in the binary file will not be as USGARRDET expects. If USGARRDET reads what seems like unusual data from an array header it will write the contents of this header to the screen and then cease execution with an appropriate error message. On other occasions its behaviour may be more erratic. For example it may misread the dimensions of the grid from the header and try to allocate memory for a grid which is far larger than that actually employed by the model, perhaps asking for more memory than the computer can give it; an obscure compiler-generated error message may then appear.

Uses of USGARRDET

The contents of a binary data file cannot be read directly by a human being. USGARRDET allows a modeller to be informed of the entire contents of a binary head/drawdown or cell-by-cell flow term file written by MODFLOW-USG. He/she can then use other programs documented herein (such as USGBIN2TAB_H) to extract these arrays and subject them to further processing.

See Also

See also ARRDET, USGARRDET_DBL, USGBIN2TAB_H and USGBIN2TAB_H1.

USGARRDET_DBL

Function of USGARRDET_DBL

USGARRDET_DBL is identical to USGARRDET except for the fact that it reads a binary MODFLOW-USG output file wherein real variables are recorded in double precision.

If USGARRDET ceases execution with an error message which states that binary file contents or headers do not make sense, then try USGARRDET_DBL. If the problem persists, then it may be rooted in the fact that different FORTRAN compilers use different methods for storage of unformatted (as distinct from binary) files.

USGBIN2CSV

Function of USGBIN2CSV

USGBIN2CSV reads a binary head/drawdown file written by MODFLOW-USG. It records heads/drawdowns over user-specified output times for user-specified nodes as a series of CSV files, one for each output time. These have a common root name. Because of this they can be imported into PARAVIEW as a block. Using standard PARAVIEW filters, these can be translated into a series of three-dimensional surfaces that can be displayed, contoured and animated.

Using USGBIN2CSV

As for all other members of the Groundwater Data Utilities suite, a response to any prompt which is comprised solely of the "e" character takes the user back to the previous prompt.

USGBIN2CSV begins execution by reading an unstructured grid specification file. From this it obtains the eastings and northings of all nodes in the unstructured grid. The prompt is:

```
Enter name of MODFLOW-USG grid specification file:
```

It then prompts for the name of a binary head/drawdown file, and for the inactive threshold in this file. Heads/drawdowns whose absolute values are above this threshold are not represented in files written by USGBIN2CSV as they are presumed to pertain to inactive or dry cells. The prompts are:

```
Enter name of MODFLOW-USG binary head/drawdown file:
Enter inactive threshold for array data in this file:
```

Cells represented in USGBIN2CSV binary output files can be selected by layer, or by providing a list of nodes in a prepared file. The choice between the two is made through responding appropriately to the following prompt:

```
Enter layer of interest (<Enter> to read list of nodes from a file):
```

If the user simply presses <Enter> in response to this prompt, he/she is then prompted:

```
Enter file to read list of nodes from:
Skip how many lines at top of file?
```

Nodes must be arranged one-to-a-line. Other data can follow the node number on each line of this file; however the node number must be the first item on each line of the file.

Then, regardless of whether node selection takes place by layer number or file, the user is prompted:

```
Enter starting simulation time of interest:
Enter ending simulation time of interest:
```

The simulation time associated with MODFLOW-USG array is recorded as the TOTIM variable in the header to each such array. Only heads/drawdowns between and including the two user-specified times will be represented in USGBIN2CSV output files. (Note that a listing of the contents of a MODFLOW-USG binary file can be obtained using the USGARRDET utility.)

USGBIN2CSV's next prompt is:

```
Enter filename base for CSV output files:
```

Suppose that the response to this prompt is "*filename*". Then output files will be named *filename1.csv*, *filename2,csv* etc. When PARAVIEW is asked to read a series of files whose names follow this protocol it assumes that they pertain to a series of sequential times. Its functionality is such as to animate such data.

Finally USGBIN2CSV asks:

```
Enter name for time index table file:
```

This file contains two columns. These associate a simulation time (i.e. TOTIM) with each CSV file.

CSV files written by USGBIN2CSV contain four columns. The first is node number while the second and third are node eastings and northings (i.e. *x* and *y* coordinates). The label of the fourth is taken from the array headers in the MODFLOW-USG binary output file. Normally this will be "HEADU".

Uses of USGBIN2CSV

USGBIN2CSV was written primarily to expedite use of PARAVIEW in visualizing MODFLOW-USG outputs. The following protocol can be adopted to import files written by USGBIN2CSV and then view them as surfaces.

1. Read them into PARAVIEW as a CSV file set.

2. Use the "table to points" filter to convert to points. Assign the X column to data labelled "X", the Y column do data labelled "Y" and the Z column to data whose label pertains to the MODFLOW-USG data type (normally "HEADU"). Check the "Keep all data arrays" box.

3. Use the "Delaunay 2D: filter to create a surface from these points. Colour as appropriate.

4. If desired, use the "Contour" filter to add animated contours to the surfaces.

See Also

See also USGBIN2TAB_H, USGBIN2TAB_H1 and USGARRDET.

USGBIN2TAB_H

Function of USGBIN2TAB_H

USGBIN2TAB_H reads a binary head/drawdown file written by MODFLOW-USG. It tabulates in an ASCII file heads or drawdowns calculated at a user-nominated simulation time at all nodes in the grid. The contents of this file can be read by other MODFLOW-USG related utilities which undertake various kinds of model postprocessing and display.

Using USGBIN2TAB_H

Optionally, heads and drawdowns computed by MODFLOW-USG can be recorded in binary output files. Heads and drawdowns can pertain to grid or connected linear network (CLN) nodes. Simulation times at which these are recorded can be set by the modeller through MODFLOW-USG's output control functionality. As is documented in the MODFLOW-USG manual, heads and drawdowns for model grid nodes are recorded as layer-specific arrays, each preceded by a header which describes the contents of the following array, including the layer and simulation time to which the array pertains. In contrast, CLN heads and drawdowns at any particular output time are recorded in a single array, also following an appropriate header. The contents of all array headers contained within any MODFLOW-USG binary head/drawdown file can be ascertained using the USGARRDET utility described elsewhere in this manual.

Program USGBIN2TAB_H extracts heads or drawdowns pertaining to a particular simulation time. It writes this data as a table wherein each head or drawdown value is clearly associated with the node to which it pertains.

USGBIN2TAB_H obtains the dimensions of the unstructured grid, and the number of nodes within each layer of the grid, from the binary output file which it reads. For it to do this correctly, heads/drawdowns for all layers must be represented within the output file. (In the case of CLN data no such precautions need to be taken as CLN-calculated heads or drawdowns are recorded in a single array at each output time; CLN nodes are not associated with model layers.)

It is important to note that use of USGBIN2TAB_H is predicated on the assumption that MODFLOW-USG employs an unstructured grid. If this is not the case, binary array headers, and ensuing binary array contents, will not be intelligible to it. If it is asked to read a binary output file associated with a structured grid, USGBIN2TAB_H will record either a self-generated or compiler-generated error message that makes reference to an unintelligible array header. Programs such as MANY2ONE or GETMULARR must be used for binary data extraction in the structured grid context.

USGBIN2TAB_H's prompts, and typical responses, are as follows.

```
Enter name of MFUSG binary head/drawdown file: umodel_usg.cln.hds
Enter simulation time (i.e. TOTIM) of interest: 7305

Enter name for node data table file: test3.dat
```

```
 - file test3.dat written ok.
```

Part of a tabular data file written by USGBIN2TAB_H appears in the following figure.

```
    NODE_NUMBER       HEADU
              1       859.3523
              2       859.3523
              3       859.3518
              4       859.3481
              5       859.3445
              6       859.2001
              7       859.2001
              8       859.2000
              9       859.1982
             10       859.1965
             11       859.2421
             12       859.2421
             13       859.2419
             14       859.2405
             15       859.2390
             16       859.3923
             17       859.3923
             etc
```

Part of the ASCII tabular data file written by USGBIN2TAB_H.

As is apparent from the above figure, a USGBIN2TAB_H tabular output file contains two columns of data. The first contains node numbers. These are numbered sequentially from 1 to the largest node number in the unstructured finite-difference grid (or the largest CLN node number if appropriate). The second column records the head or drawdown associated with each such node. The header to the second column is taken directly from the MODFLOW-USG binary file; however underscores replace any spaces that are present within the header.

USGBIN2TAB_H requires that the user choose a simulation time for data extraction from the MODFLOW-USG binary output file. The times for which output are available can be obtained using the USGARRDET utility; these output times are listed as the MODFLOW-USG TOTIM variable.

A single MODFLOW-USG head/drawdown binary output file may contain different types of data. For example, it may contain both head and drawdown data for both grid nodes and CLN nodes. If this is the case only the first data type recorded at the user-specified simulation time is transferred to the tabular output file. (It is better practice to instruct MODFLOW-USG to record data of different types in different binary output files.)

Uses of USGBIN2TAB_H

The ASCII file generated by USGBIN2TAB_H can be used by other utility software for post-processing of model-generated quantities. Such postprocessing may include visualization, or generation of secondary quantities that are functions of model-generated quantities.

See Also

See also USGBIN2TAB_H1, USGARRDET, USGPROP2TAB1, USGPROP2TAB2.

USGBIN2TAB_H1

USGBIN2TAB_H1 is identical to USGBIN2TAB_H except for the fact that its output file includes an extra column. This column lists the layer number associated with each node.

USGBUD2SMP

Function of USGBUD2SMP

USGBUD2SMP functions in much the same way as does the BUD2SMP utility for conventional, structured, MODFLOW; however USGBUD2SMP reads an unstructured budget file produced by MODFLOW-USG. As for BUD2SMP, the binary cell-by-cell flow term file (i.e. the budget file) must be written using the COMPACT protocol by MODFLOW-USG.

USGBUD2SMP extracts flow rates accumulated over one or a number of user-defined zones within the model domain for all times represented in the cell-by-cell flow term file. It then records these flow rates in bore sample file format. A file written by USGBUD2SMP can then be processed using the SMP2HYD utility for plotting of flows against time in each zone. Alternatively (or as well) SMP2SMP can be employed to write another bore sample file in which flow rates are matched to those recorded in a measurement bore sample file. PESTPREP, PESTPREP1 or PESTPREP2 can then be used to automate construction of a PEST input dataset in which modelled flows are matched to their observed counterparts.

Using USGBUD2SMP

As for many of the Groundwater Data Utilities, immediately upon commencement of execution USGBUD2SMP looks for a file named *settings.fig* in the subdirectory from which it was invoked. If this file is not found, USGBUD2SMP terminates execution with an appropriate error message. As is explained in Part A of this manual, the contents of file *settings.fig* inform those utilities that read it of the format to use in representing dates.

USGBUD2SMP's first prompt is:

```
Enter name of MFUSG unformatted BUDGET output file:
```

USGBUD2SMP can only process cell-by-cell flow term files produced by MODFLOW-USG if MODFLOW-USG employs an unstructured grid. Furthermore, USGBUD2SMP can only read such files if they are stored in COMPACT form. This is because files stored in this form contain timing information (lacking in other forms of MODFLOW-USG budget storage) which is essential for the recording of flow data in a manner that allows plotting against elapsed simulation time. If USGBUD2SMP discovers that the cell-by-cell flow term file whose name was provided above is not, in fact, stored in COMPACT form, it terminates execution with an appropriate error message. (If your MODFLOW-USG pre-processor does not provide an option for file storage in this manner this is not a problem, for a user can easily create a MODFLOW-USG OUTPUT CONTROL file him/herself which directs MODFLOW-USG to store files in COMPACT form. See MODFLOW-USG documentation for details.)

It is important to note that for if a MODFLOW-USG model employs a Connected Linear Network (CLN) process, *budget output from the CLN process should be written to a different file from that used to record the Groundwater Flow (GWF) process budget terms.*

USGBUD2SMP next prompts:-

```
Enter maximum number of output times:
```

Through its OUTPUT CONTROL input dataset, MODFLOW-USG is directed to provide cell-by-cell flow term output at the end of certain time steps, stress periods, or at particular simulation times. In response to the above prompt, you should inform USGBUD2SMP of the total number of times for which there is such cell-by-cell output. If you are unsure, simply enter a number that is likely to exceed the number of output times; USGBUD2SMP uses this number solely to dimension internal arrays. If the number is too large, it does not matter (except if it is so large that USGBUD2SMP runs out of memory). If it is too small, USGBUD2SMP will inform you of this later in its processing and ask that you run it again, supplying a higher number for this parameter.

USGBUD2SMP's next prompt is:-

```
Enter text to identify MFUSG flow type:-
```

Whenever it writes an array to its cell-by-cell flow term file, MODFLOW-USG first records an array header. The header contains timing information as well as a 16-character identifier of the flow type represented in the following array. Some of these identifiers are set out in the tables below for the GWF and CLN processes respectively. Note that the flow term "FLOW JA FACE", which refers to flow between connected cells, is not accommodated by USGBUD2SMP at present as this flow term is inter-zonal rather than zone-specific; arrays of this type are simply ignored. Computation of zone-to-zone flows based on this type of cell-by-cell flow information can be undertaken using the USGS ZONBUDUSG utility (refer http://water.usgs.gov/ogw/mfusg/).

Package	**Text Identifier**
bcf/lpf	STORAGE
bcf/lpf	CONSTANT HEAD
bcf/lpf	FLOW JA FACE
flow and head boundary	SPECIFIED FLOWS
general head boundary	HEAD DEP BOUNDS
well	WELLS

drain	DRAINS
river	RIVER LEAKAGE
evapotranspiration	ET
recharge	RECHARGE

Some text identifiers contained in cell-by-cell flow term array headers for the GWF process.

Package	**Text Identifier**
cln	CLN STORAGE
cln	CLN CONST HEAD
cln	FLOW CLN FACE
cln	GWF TO CLN

Some text identifiers contained in cell-by-cell flow term array headers for the CLN process.

The user should supply at least part of an appropriate text identifier in response to the above prompt. If the user-supplied text occurs within an array identifier, that array will be processed by USGBUD2SMP. Otherwise the array is ignored. Only enough text needs to be provided in response to the above prompt to uniquely identify one particular flow type. If the user wishes to plot more than one flow term type against time, he/she should run USGBUD2SMP more than once, supplying a different text string in response to the above prompt on each occasion. (Note that, as is explained below, the user may ascertain the text identifiers pertaining to various MODFLOW-USG packages by reading USGBUD2SMP's record file after USGBUD2SMP has finished execution. Alternatively, USGARRDET can be used for this purpose.)

USGBUD2SMP next prompts:-

```
Enter simulation starting date [dd/mm/yyyy]:
Enter simulation starting time [hh:mm:ss]:
Enter time units employed by model [y/d/h/m/s]:
```

(Note that date representation will be in the format "mm/dd/yyyy" instead of "dd/mm/yyyy" if this is appropriately denoted in the settings file *settings.fig.*) USGBUD2SMP requires the above information so that it can record the date and time pertaining to every flow rate on its output file.

The user is then requested to provide the nodal dimension of the model:-

```
Enter total number of nodes in grid/network:
```

If USGBUD2SMP is reading a budget file associated with the CLN process, the number of nodes supplied in response to the above prompt must pertain to the CLN network rather than to the MODFLOW-USG flow grid. This information allows USGBUD2SMP to check the integrity of the MODFLOW-USG budget file; it also

allows USGBUD2SMP to know the size of the integer zone list array that it must read. This happens next.

As for the BUD2SMP utility (of which USGBUD2SMP is the unstructured grid equivalent), flows in and out of nodes are accumulated over zones. A zone is defined as the collection of nodes to which the same integer value is assigned. Note, however, that nodes which are assigned an integer value of zero are presumed to belong to no zone at all, and hence are ignored. This allows a user to ascertain flows within a small part of the model domain by supplying a series of zone numbers which are everywhere zero except within that part of the domain which is of interest.

Zonation is defined through an integer zone list read from a file. This list must provide the zone number ascribed to each node in the MODFLOW-USG flow grid or CLN network. An integer must be supplied for each node within the GWF or CLN grid. Integers in the list must be supplied in node sequential order. They are read using free-field format; hence each number must be separated by whitespace (including blanks, tabs or a new line) or a comma. Thus the file can be tabular, with zone numbers recorded on consecutive lines of the file; alternatively the integer zonation list can be written in traditional array-type format, with many integers arranged on each of many consecutive lines. The "n*I" convention can be used in this context; this represents a string of *n* integers of value *I*.

USGMOD2SMP prompts for the name of the file containing the integer zonation list as follows:

```
Enter name of integer zonation list file:
```

USGBUD2SMP does not insist that the list of zone-defining integers be the only occupant of the file from which the list is read. Hence it asks how to find the integer list on the file. Two options are available, namely an indicator text or a line number. The prompt is:

```
Locate integer list using text string or line number? [t/l]:
```

If text is chosen, the user is next prompted for the pertinent text string:-

```
Enter text string:-
```

Supply enough text to uniquely locate a header in the input file (you need only supply a substring of the header text). Note, however, that if the text does not uniquely demarcate a header in response to the above prompt, then the first match to this text will be assumed to be correct. Note also that the text search is case-insensitive. USGBUD2SMP assumes that the integer list begins on the line following the identified text string.

Alternatively, if the line number option was selected, MODFLOW-USG prompts:-

```
Enter line number:
```

in response to which an appropriate line number must be supplied. Free-field reading of the integer list will begin at that line.

By way of example, suppose that the file containing the integer zonation list is as follows. USGBUD2SMP could be guided to the array using either the text string "zones" or the line number "4".

```
Line #1
Line #2
MFUSG NODAL ZONES
1 2 2 2 2
1 1 0 2 2
1 1 0 3 3
2*3
1
1
1
```

Example of a file containing an integer zonation list for a MODFLOW-USG groundwater flow or connected linear network grid.

When writing a bore sample file, USGBUD2SMP needs to know the "bore identifier" to assign to each zone. So for each different non-zero zone that it finds in the zone-defining integer list it asks:-

```
Enter identifier for flows in zone n:
```

where *n* is a zone-defining number occurring within the nodal integer list. Supply a name comprised of 10 characters or less; no two zones should be supplied with the same identifier. No spaces are allowed in this name.

USGBUD2SMP writes two files. One is a bore sample file, the other is a file recording the details of all arrays found in the cell-by-cell MODFLOW-USG output file. If you wish to simply know what arrays are present in this file without necessarily creating a bore sample file, run USGBUD2SMP, supplying a flow identification text string which does not match any of the array identifiers used by MODFLOW-USG.

Next USGBUD2SMP prompts for the name of its principal output file:-

```
Enter name for bore sample output file:
```

For each time step at which cell-by-cell flow term data was accumulated, USGBUD2SMP calculates the total flow within each of the non-zero zones defined in the integer zone definition list; however the zero-valued zone (if it exists) is ignored as far as flow term accumulation is concerned. USGBUD2SMP records accumulated flows to the user-nominated bore sample file. Within this file, one line is recorded for each zone for each time for which flow data was recorded by MODFLOW-USG.

Next USGBUD2SMP asks:

```
Enter flow rate factor:
```

A flow rate factor different from unity may be required to convert MODFLOW-USG flow rates to more appropriate units. Flow rates recorded in the MODFLOW-USG cell-by-cell flow term file employ the length and time units used by MODFLOW-USG

itself. Flow rates calculated for each MODFLOW-USG zone are multiplied by the flow rate factor supplied above before being written to the bore sample file. The MODFLOW-USG protocol of negative values representing model outflows and positive values represent model inflows is respected in writing this file.

Flow rates can be referenced to the time pertaining to the beginning, middle or end of the time step with which they are associated. USGBUD2SMP prompts the user for his/her wish:-

```
Assign flows to beginning, middle or finish of time step? [b/m/f]:
```

If the response to the above prompt is "`f`", the time associated with each flow term (i.e. the time written to each line of the bore sample file) will be the same as the time of model output, this being the end of a particular model time step. However in some instances a user may consider that flow rates should be plotted at a time corresponding to the middle of the time step in which they are evaluated. This will occur if "`m`" is supplied in response to the above prompt. By typing "`b`", flow terms can be ascribed to the beginning of pertinent MODFLOW-USG time steps.

USGBUD2SMP's final prompt is:-

```
Enter name for run record file:
```

Upon completion of execution USGBUD2SMP writes a record of every array encountered in the MODFLOW-USG cell-by-cell flow term file which it has just read. Part of such a file is shown below.

Stress_period	Time_step	Elapsed_time	Flow_type	Flow_processed_by_USGBUD2SMP
1	5	31.000	STORAGE	no
1	5	31.000	CONSTANT HEAD	no
1	5	31.000	FLOW JA FACE	no
1	5	31.000	WELLS	no
1	5	31.000	DRAINS	no
1	5	31.000	RIVER LEAKAGE	yes
1	5	31.000	ET	no
1	5	31.000	RECHARGE	no
2	5	62.000	STORAGE	no
2	5	62.000	CONSTANT HEAD	no
2	5	62.000	FLOW JA FACE	no
2	5	62.000	WELLS	no
2	5	62.000	DRAINS	no
2	5	62.000	RIVER LEAKAGE	yes
2	5	62.000	ET	no
2	5	62.000	RECHARGE	no

Part of a USGBUD2SMP run record file.

Uses of USGBUD2SMP

USGBUD2SMP can form part of a composite model run by PEST. The commands to run individual model components are placed in a batch or script file. These will include the command to run MODFLOW-USG; the commands to run USGBUD2SMP (and probably SMP2SMP) will follow that. Recall that SMP2SMP (documented elsewhere in this manual) generates a bore sample file in which model-generated flows are time-interpolated to the dates and times of measured flows. If this

protocol is adopted, input file preparation for the PEST run in which flow terms comprise part of a calibration dataset can be easily accomplished through use of PESTPREP, PESTPREP1 or PESTPREP2 (all of which are documented herein).

See Also

See also USGARRDET, USGARRDET_DBL USGBUD2SMP1, PESTPREP, PESTPREP1, PESTPREP2, SMP2SMP and SMP2HYD.

USGBUD2SMP1

Function of USGBUD2SMP1

USGBUD2SMP1 performs a similar role to USGBUD2SMP in that it reads a binary cell-by-cell flow file written by MODFLOW-USG and records budget terms of interest in bore sample file format. Data recorded in this format is amenable to further processing by Groundwater Data Utility suite software. Interpolation to the times of measured flows can be done using the SMP2SMP utility. Programs PESTPREP, PESTPREP1 and PESTPREP2 can be used to prepare PEST input files in which budget terms comprise part of the calibration dataset.

USGBUD2SMP1 can detect with a MODFLOW-USG budget file records data in single or double precision. It adjusts its reading of the file accordingly.

Using USGBUD2SMP1

Bore-To-Budget File

USGBUD2SMP1 obtains specifications for extraction of budget terms from a "bore-to-budget" file. It is the user's responsibility to prepare this file prior to running USGBUD2SMP1. An example of a bore-to-budget file follows.

```
# Example of a bore-to-budget file

ch1 CONSTANT HEAD NODEGRP 5 4 0.75 12        &
                          1.0 15 1.0 18 1.0  &
                                             &
                          25 0.15
ch2 CONSTANT HEAD NODEGRP 5 14 0.75  32 1.0  85 1.0  78 1.0  95 0.25

ja_flow FLOW JA FACE NODEGRP 2 443 1.0 122 1.0

cc1 COMPOUND 2 ch1 ch2
cc2 COMPOUND 2 springck12 springck11

SpringCk12 RIVER LEAKAGE AUX CELLGRP 0
SpringCk11 RIVER LEAKAGE AUX CELLGRP 2
```

Example of a bore-to-budget file.

Each line (except for continuation lines – see below) of a bore-to-budget file begins with a name, this being the name of a "flow target". This name must be a maximum of 10 characters in length. This 10 character restriction is imposed in order to accommodate the specifications of a bore sample file (see Part 1 of this manual) whose task it is for USGBUD2SMP1 to write. Each such target name is transferred to the bore sample file, along with pertinent flows extracted from the MODFLOW-USG budget file for all output times found in the latter file. In the above example, flow targets are named "ch1", "ch2", "ja_flow", "cc1", "cc2", "SpringCk12" and "SpringCk11". Note that, along with all other content of a bore-to-budget file, these names are case insensitive.

Following the name of each flow target must be either the name of a MODFLOW-USG flow type ("constant head", "ja face", "river leakage" in the above example), or the "compound" specification. In the former case, the text characterizing the flow type must exactly match corresponding text which appears in the headers to MODFLOW unformatted budget records (including spaces between words); however, as stated above, the case does not matter. (Note that text headers appearing in a budget file can be listed to an ASCII file using the USGARRDET and USGARRDET_DBL utilities documented herein.)

If the flow target is of "compound" type, then the flow associated with that target is specified to be the sum of flows associated with other targets. Following the word "compound" must be an integer ("2" in both of the above COMPOUND examples) which specifies the number of target flows to be summed, followed by the names of the targets whose flows are thus summed. Note that these latter targets cannot be COMPOUND targets themselves (for this could lead to recursive summation).

Alternatively, if the string NODEGRP follows the flow type specification associated with a certain target, then this indicates to USGBUD2SMP1 that a list of MODFLOW-USG model nodes will be provided (or connections between nodes if JA FACE flows are requested); flows into these nodes (or along these links) are summed to compute the target flow. Following the flow type text must be an integer which specifies the number of nodes (or links) for which flows are to be summed in assigning a flow to the target. Suppose that this number it *n* (it is "5" and "2" in the above example file). Then *n* pairs of numbers follow, the first of each pair being an integer (a node or JA linkage number) and the second being a factor (a real number). At each MODFLOW-USG output time, the flow assigned to the flow target will be the sum of the flows pertaining to the cited nodes (or JA linkages), with each such flow multiplied by the associated factor during the summation.

Alternatively, the string AUX CELLGRP can follow the name of the flow target type. Use of this flow target specification protocol is predicated on the assumption that auxiliary data is stored in the budget file along with the budget data itself; this can be done for flow types such as those associated with DRAIN, RIVER, WELL and some other boundary conditions. One of these auxiliary data types must be named CELLGRP. Selection of which cells to include in the summed flow that is assigned to the flow target is made on the basis of values assigned to the CELLGRP auxiliary variable as it pertains to nodes listed in the budget file for that flow type. In the bore-to-budget file, this "selection variable" must be an integer (and therefore acts as a kind of zonation), despite the fact that auxiliary data is real in the MODFLOW-USG budget file; USGBUD2SMP1 simply takes the nearest integer corresponding to each CELLGRP value, and compares this with selection integers provided in the bore-to-budget file. In the above example, a CELLGRP value of 0 causes flows to be assigned to the "SpringCk12" flow target whereas a CELLGRP value of 2 selects flows for the "SpringCk11" target.

Other salient features of the bore-to-budget file follow.

1. Blank lines and comment lines can be inserted between flow target specification lines. Comment lines begin with the "#" character.

2. Specifications for COMPOUND and NODEGRP flow targets can continue onto subsequent lines. This is done by terminating an unfinished line with the "@" character. However this character cannot be placed too early in the first line of the flow target specification; it must be placed after the integer which specifies the number of nodes or targets that follow. As many continuation lines can be provided as is necessary to complete a flow target specification.

3. All parts of all flow target specification lines are case-insensitive.

4. The width of the bore-to-budget file must be 2000 characters or less. Continuation lines can accommodate NODEGRP specifications which include any number of cells.

It is worthy of note that the NODEGRP option can be slow where flows in many nodes must be summed. This is because the matching of nodes in a NODEGRP list to those in a list provided in the binary budget file requires many operations, especially where no particular ordering is assumed in the NODEGRP list and where node repetition is allowed. In this case zonation selection options provided by USGBUD2SMP may be quicker.

Prompts and Responses

Typical USGBUD2SMP1 prompts and responses are as follows.

```
Enter name of MFUSG unformatted budget output file: calib.ccf
Enter maximum number of output times featured in this file: 100

Enter name of bore-to-budget file: calib.b2b
- file calib.b2b read ok.

Enter simulation starting date [dd/mm/yyyy]: 1/1/2010
Enter simulation starting time [hh:mm:ss]: 20.0
Enter time units employed by model [y/d/h/m/s]: d

Enter name for bore sample output file: model.smp
Enter flow rate factor: 1.0
Assign flows to beginning, middle or finish of time step?  [b/m/f]: f

Enter name for run record file: temp.rec

- data for 1 model output times written to file model.smp
- see file temp.rec for a record of arrays found in file calib.ccf
```

See documentation of USGBUD2SMP for a description of other aspects of USGBUD2SMP1 usage.

Uses of USGBUD2SMP1

USGBUD2SMP1 can be run as a MODFLOW-USG postprocessor as part of a batch file run by PEST as "the model". Model-calculated flows corresponding to measured flows which comprise part of a calibration dataset can thereby be extracted from a MODFLOW-USG budget file. Interpolation to the times at which flow measurements comprising the calibration dataset were made can be effected using the SMP2SMP utility as a USGBUD2SMP1 postprocessor, provided that measured flows are also stored in bore sample file format.

Alternatively, USGBUD2SMP1 does not necessarily need to be employed as part of a calibration process. It can be used simply to extract flows of interest from a MODFLOW-USG budget file. The SMP2HYD utility can then be used to plot these flows against time.

See Also

See also USGARRDET, USGARRDET_DBL USGBUD2SMP, PESTPREP, PESTPREP1, PESTPREP2, SMP2SMP and SMP2HYD.

USGDBL2SGL

Function of USGDBL2SGL

If MODFLOW-USG is compiled such that all variables are double precision, then the numbers represented in its binary output files will be recorded as double precision numbers. Such a file will not be readable by standard MODFLOW-USG post-processors (such as those documented herein). USGDBL2SGL ("DBL2SGL" stands for "double-to-single") rewrites a binary MODFLOW-USG output file using the single precision protocol so that normal MODFLOW-USG postprocessors can read it.

Using USGDBL2SGL

Typical USGDBL2SGL prompts and responses are as follows:

```
Enter name of double precision unformatted MFUSG file: usg_coarse.cbb
Is this a head/drawdown or cell-by-cell flow file?  [h/c]: c

Enter name for equivalent single precision file: temp.dat
```

USGDBL2SGL can read binary head/drawdown (including those pertaining to the CLN package) and cell-by-cell flow term output files. However these must be written for an unstructured grid model (i.e. a model for which node numbers, rather than row, column and layer numbers, are used to identify cells.). The file written by USGDBL2SGL will be immediately readable by programs such as USGARRDET and other post-processing programs described herein.

Uses of USGDBL2SGL

For some models, a double precision version of MODFLOW-USG will run faster and have better convergence properties than a single precision version. However binary files produced by this version of MODFLOW-USG must be available for processing by standard MODFLOW-USG postprocessors. The USGDBLE2SGL utility undertakes the processing required to make this happen.

See Also

See also USGARRDET.

USGDUALMODEL

Function of USGDUALMODEL

USGDUALMODEL writes a rather complex data file named a "node association file". This file is not of direct use to a modeller. However it can be used by modelling support software which performs rudimentary parameter upscaling between two models – a coarse model and a fine model. The node association file provides information on the spatial relationships between cells of the fine and coarse models.

Certain assumptions regarding the relationships between coarse and fine model cells underpin use of USGDUALMODEL. These assumptions will almost certainly be met where utility software, or a commercial MODFLOW-USG graphical user interface, constructs a fine model from a coarse model through quadtree refinement of the latter, or through construction of a nested grid over one or more subdomains of the overall coarse model domain. These assumptions (many of which are checked by USGDUALMODEL, but some of which are not) are as follows.

- All cells within both models are vertical rectangular prisms. Thus four of the six faces of each cell are vertical. In horizontal cross section the remaining two sides (i.e. the top and bottom faces of each cell) form a rectangle. Verification that these conditions are met can be checked using the USGORTHCHEK utility.)

- The overall domains of both models are the same.

- Some (or all) cells of the coarse model are subdivided into fine cells. While the subdivision pattern can be arbitrary, all fine model cells must be rectangular prisms, as stated above. The subdivision process must be such that each fine model cell lies entirely within a single coarse model cell. (USGDUALMODEL does not check this.)

- The two orthogonal directions in the XY plane defined by the vertical faces of all cells must coincide for the coarse and fine models. Thus one model must not be rotated with respect to the other. This is necessary (though not sufficient) to prevent a fine model cell from overlapping two coarse model cells. (USGDUALMODEL checks this condition.)

- The coarse model must have the same or fewer layers than the fine model. Any one fine model layer must lie completely within the bounds of a coarse model layer. Multiple fine model layers can replace a single coarse model layer.

Using USGDUALMODEL

Prompts and Responses

USGDUALMODEL asks a lot of questions. As for other members of the Groundwater Data Utility suite, if the response to any of these questions is "e" (for "escape"), the user is taken back to the previous question.

USGDUALMODEL commences execution by prompting for the names of unstructured grid specification files for the coarse and fine models. Its prompts are:

```
Enter coarse model unstructured grid spec file:
Enter fine model unstructured grid spec file:
```

After ensuring that the files are indeed present, USGDUALMODEL reminds the reader of some of the assumptions under which it operates. It says:

```
Both of these model grids must satisfy the following criteria:
- each cell is a vertical prism with six faces
- cell sides form rectangles in the XY plane (check with USGORTHCHEK)
- fine model cells form a complete tile coverage of coarse model cells
Are these criteria satisfied? [y/n]:
```

If the answer to this question is "y" USGDUALMODEL proceeds with its work under the assumption that the user knows what he/she is doing. It begins by reading the first part of each of the coarse and fine model grid specification files in order to establish the dimensionality of these models. If it finds that the fine model has more layers than the coarse model, it asks the user for the layering relationships between the two models. Prompts are:

```
The fine model has more layers than the coarse model.
Layer correspondences must be provided.
  How many fine model layers in coarse model layer 1?
  How many fine model layers in coarse model layer 2?
  etc.
```

The above question is repeated for each coarse model layer. If, after all prompts have been issued and responses provided, the number of fine model layers cited in these responses does not add up to the total number of layers in the fine model grid as read from the unstructured grid specification file, USGDUALMODEL repeats the above series of prompts until the user gets it right. Note that if the number of layers in the fine model is the same as that in the coarse model, the above prompts are not issued as there is obviously a one-to-one relationship between fine and coarse model layers. If, on the other hand, the coarse model is found to possess more layers than the fine model, then USGDUALMODEL issues a terse error message and ceases execution.

Next USGDUALMODEL asks for the name of the file that it must write. Its prompt is:

```
Enter name for node association file:
```

Some programs which use this file may want to be informed of coarse and fine model layer thicknesses on a cell-by-cell basis. So USGDUALMODEL asks:

```
Include cell thickness data in this file? [y/n]:
```

If the answer is "y" it asks:

```
Enter node data table file of coarse cell thicknesses:
Enter node data table file of fine cell thicknesses:
```

The specifications of a node data table file are provided in Part A of this manual. This file must have at least two columns of data, the first of which is labelled "NODE" or "NODE_NUMBER". In the present case, another column must be labelled "THICKNESS"; USGDUALMODEL finds this column itself (or issue an error message if it does not). A node data table file containing node thicknesses is easily prepared in the following way:

1. Use the USGPROP2TAB1 or USGPROP2TAB2 programs to read the MODFLOW-USG discretization file, extracting cell tops and bottoms in turn from this file. These are recorded in node data table file format by these programs.

2. Paste these columns into a spreadsheet such as EXCEL.

3. Subtract one from the other to calculate thickness on a cell-by-cell basis.

4. Export the thickness data (with or without the BOTTOM and TOP columns) as a (tab-delimited) node data table file.

USGDUALMODEL will cease execution with an error message if any nodes are omitted from either the coarse or fine model thickness node data table files. It will forgive negative cell thicknesses however, assuming that these pertain to inactive cells (USGDUALMODEL is not aware of the activity status of each cell); it upgrades negative thicknesses to zero.

USGDUALMODEL then does its processing. It finds all fine model cells that are contained within any coarse model cell. This can take a while, so patience may be required. It then writes the node association file. The information contained within this file should allow other programs to proceed with processing of dual model data without the need to establish coarse/fine model cell associations, for this work has been done for it by USGDUALMODEL.

Details of the node association file which USGDUALMODEL writes are now presented.

The Node Association File

The node association file is divided into different sections. Each section is preceded by a text header. The header begins with a "*" character (just like a PEST control file). A seriously edited version of this file is shown below.

```
* coarse grid specifications
     35244          3
     11748     11748     11748
* fine grid specifications
     48972          9
     13156      1584      1584     13156     13156      1584   etc.
* coarse model node data
```

```
         1     1   10000.00  103.23
         2     1   10000.00  98.321
         3     1   10000.00  89.324
         etc.
* fine model node data
         1     1   10000.00   83.432
         2     1   10000.00   82.312
         3     1   10000.00   75.395
         etc.
* number of fine layers in each coarse layer
         3         1         5
* number of fine cells in each coarse cell
         1         1         1         1         1        etc.
         etc.
         1         1         1         1
* coarse-ordered fine cells array
         1         2         3         4         5        etc.
         etc.
* coarse cell in which each fine cell is situated
         1         2         3         4         5        etc.
         etc.
* coarse model nodal thicknesses
   309.6499       244.2447       305.0672       295.5038  etc.
         etc.
* fine model nodal thicknesses
   309.6499       244.2447       305.0672       295.5038  etc.
         etc.
```

Extract from a node association file written by USGDUALMODEL.

The contents of each section are now described.

* coarse grid specifications

The first line following the section header contains two numbers. These are the total number of cells in the coarse model grid and the number of layers in the coarse model grid respectively.

The next line provides the NODELAY array – the number of cells in each model layer. There are as many entries in this array as there are layers in the model. The array wraps over to the next line if the coarse model has many layers.

* fine grid specifications

The specifications for this section of the node association file are the same as those of the preceding section; however entries pertain to the fine model instead of the coarse model.

* coarse model node data

This section has as many lines as there are cells in the coarse model. Each line has either three of four entries, depending on whether cell thickness data are requested. The first entry is the node number; the second is the model layer to which each node belongs; the third is the area of the cell to which the node belongs; the fourth (optional) entry is the vertical thickness of the cell. Cell areas are evaluated from the

locations of cell vertices provided in the unstructured grid specification file for the coarse model. Cell thicknesses are read from a node data table file as described above.

* fine model node data

The specifications for this section of the node association file are identical to those of the preceding section; however data in this section pertains to the fine model rather than to the coarse model.

* number of fine layers in each coarse layer

This section contains a single array of numbers. The array has as many entries as there are layers in the coarse model. If necessary, the array wraps to the next line. As the heading states, each entry is the number of fine layers that are contained within each coarse model layer.

* number of fine cells within each coarse cell

A single array of numbers is provided, these wrapping onto successive lines as needed. The array has as many elements as there are cells in the coarse model. Each entry is the number of fine model cells that are contained within the respective coarse model cell. Note that where there is more than one fine model layer within a coarse model layer, then the cells that are contained within a coarse model cell are stacked horizontally and vertically.

* coarse-ordered fine cells array

This array (which wraps onto as many lines as are required to hold it) contains as many elements as there are cells in the fine model. This is the array which normally takes the longest time to fill. It has as many elements as there are cells in the fine model. Given a coarse model cell, it allows the user to quickly find the fine model cells which are contained within that coarse cell. It is ordered such that the first N elements list the N fine cells that are contained within coarse model cell number 1; the next M elements list the M cells that are contained within coarse model cell number 2; etc. Values for N, M etc. for each coarse model cell constitute the array that resides in the "* number of fine cells in each coarse cell" section of the node association file.

* coarse cell in which each fine cell is situated

This array too has as many elements as there are cells in the fine model. For each cell it simply states the cell number of the coarse model in which the pertinent fine model cell is situated.

Uses of USGDUALMODEL

Through its "observation re-referencing" functionality, PEST allows calibration of a complex model with one or more simple models used for calculation of derivatives. This can result in large savings in overall inversion time, as by far the largest number

of model runs required for calibration of a model are those used for calculation of derivatives.

Conceptually, MODFLOW-USG provides unique opportunities for harvesting the benefits of dual model usage. The "real" model may possess a uniformly fine grid throughout the entire model domain. Alternatively, it may possess a grid that is fine in places of interest and coarse elsewhere; the grid for this model may have been constructed through local nesting or quadtree refinement of a coarse model grid. In either case the more detailed model may take much longer to run than the coarse model, and may even suffer from convergence difficulties.

If pilot points are being used for parameterization of the model domain, these may have been placed at many locations within many model layers. Where the sensitivity of model outputs is being sought to a pilot point parameter which is not situated in a fine part of the model grid, then the fine model can be replaced by a uniformly coarse-gridded model for the pertinent sensitivity-calculation model run. Alternatively, where a model is uniformly fine, a user may design a number of coarse-gridded models which are locally fine in the vicinities of different groups of pilot points while being elsewhere coarse. The partially fine models can then be run when calculating sensitivities to pilot points within the various groups; the running of a partially fine-gridded model may be far less computer intensive than the running of a uniformly fine-gridded model for pilot point sensitivity calculations.

In all of these cases a means must exist for rapidly computing (an approximation to) coarse model hydraulic properties from fine model hydraulic properties so that the coarse model can be run in place of the fine model for derivatives calculation. Some kind of (aerially-weighted and maybe thickness-weighted) averaging process may be appropriate. (More complex upscaling procedures require information on cell connectivity and not just location.) The software that performs upscaling-through-averaging must be aware of the spatial relationships between fine and coarse model cells. USGDUALMODEL provides this awareness.

See Also

See also USGORTHCHEK, USGPROP2TAB1 and USGPROP2TAB2.

USGGRIDLAY

Function of USGGRIDLAY

USGGRIDLAY reads an unstructured grid specification file. For a user-specified layer in this file, USGGRIDLAY writes any or all of the following files:

- a MIF/MID file pair featuring cell vertices;
- a SURFER BLN file of cell vertices;
- an XYZ file of grid node locations and elevations.

The MIF/MID file can be imported into any GIS platform (such as the MAPWINDOWS and QGIS public domain GIS packages). Within a GIS, a layer of the grid can be displayed over other map data. Grid integer or nodal data can then be assigned and edited prior to being exported from the GIS platform and used by the model.

Use of USGGRIDLAY is predicated on the assumption that all MODFLOW-USG model cells are vertical prisms (USGGRIDLAY verifies that this is the case). The upper vertices of each cell are represented in the MIF and BLN files written by USGGRIDLAY; because of the vertically prismatic nature of each cell, a set of lower vertices lie vertically below these, and hence are not represented in these output files. Node locations (the coordinates of which are supplied in the unstructured grid specification file) are represented in the XYZ file.

Using USGGRIDLAY

As for any other program of the Groundwater Data Utilities suite, the user can backtrack to the previous prompt by responding with “e” to the current prompt.

On commencement of execution, USGGRIDLAY prompts for the name of the unstructured grid specification file which it must read. The user is then asked to identify the grid layer for which information is to be recorded in USGGRIDLAY output files. The prompts are:

```
Enter name of unstructured grid specification file:

Enter layer number of interest:
```

Next, USGGRIDLAY asks the user for the file types that he/she would like USGGRIDLAY to write:

```
Enter filename base for MIF/MID files (<Enter> if none):
Enter filename base for BLN file (<Enter> if none):
Enter name for nodal xyz file (<Enter> if none):
```

At least one of these must be requested; all of them can be requested if desired.

The user has the option of having all cells within the requested layer represented in the output files written by USGGRIDLAY; alternatively he/she may request that some of these cells be "filtered out". This may be warranted if, for example, some of them are inactive. Optionally a node data table file can be read in which information is provided for each cell of a model grid. (See part A of this manual for a description of this file type). The user selects the data column containing numbers which will be used as a basis for omitting cells. He/she also selects a number. Every cell to which this number is assigned in the identified column of the node data table file will be omitted from all USGGRIDLAY output files. Prompts and some typical responses follow. These responses assume that the node data table file is named *grid.ndt* and that one of the columns in this file has a header "IBOUND". Cells with an IBOUND value of zero are omitted from representation in USGGRIDLAY output files.

```
Filter output using column of node data table file? [y/n]: y
Enter name of node data table file: grid.ndt
Enter column header: ibound
Does column contain integer or real data? [i/r]: i
Enter value for node omission: 0
```

Note the following points pertaining to use of a node table data file for filtering out model cells.

- The column header is case-insensitive. Thus the header in the node data table file in the above example may be "IBOUND" or "ibound".

- As is specified in Part A of this manual, the first column of a node data table file must have the header "NODE" or "NODE_NUMBER".

- It is not essential however that the node data table file which USGGRIDLAY reads cites all cells within an unstructured model grid. Only those cells which are identified for omission must be represented in this file.

Note also that re-importation of edited tables of node data into a MODFLOW-USG model may be more difficult than it otherwise would be if this table has missing elements.

Note the following points pertaining to USGGRIDLAY's MIF/MID file functionality.

- No coordinate system or projection data is recorded in the MIF file. The GIS which reads this file will therefore ask the user for the coordinate reference system to which the data pertains.

- Three data columns are represented in the MIF/MID file pair. The first is the node number (labelled "node_number"). The second and third columns are labelled "int_val" and "real_val". All data values for all nodes are set to zero in these columns. The user can fill these columns with other data as he/she sees fit, and re-import this altered data later into the model.

Uses of USGGRIDLAY

Files written by USGGRIDLAY allow a grid to be viewed on a map – both the cells within the grid (each defined by its respective corner vertices), and the nodes of the grid.

The MIF/MID files can be imported into any GIS wherein they can be converted immediately into ESRI shapefile format if desired. (In many GIS platforms the latter can be edited whereas the former cannot.) The integer and real data columns supplied in the MIF and MID files can then be filled with data such as cell activity, boundary condition locations, and/or a hydraulic properties – all against a map background. Other data columns can be added if desired. Node-specific integer and real data thus acquired can then be saved to a table and combined with data of the same type pertaining to other layers. The data can then be assimilated into a MODFLOW-USG model.

See Also

See also USG2VTK and USGNDTF2MIF.

USGMOD2ARRAY

Function of USGMOD2ARRAY

USGMOD2ARRAY performs a similar function to MOD2ARRAY; however it works with a MODFLOW-USG input dataset rather than with a MODFLOW input dataset. It reads a sequence of arrays of layer-specific node data from a MODFLOW-USG input file; it then re-writes all of this data to a single file which can be read using MODFLOW-USG's "EXTERNAL" array-reading protocol. It alters the MODFLOW-USG input data file so that MODFLOW-USG is directed to read the new external file. It also alters the MODFLOW-USG name file so that MODFLOW-USG is instructed to open the external file at the beginning of its execution.

The re-written data is recorded as a single column in the external file. This can be easily turned into a "node data table file" (see part A of this manual for a description of this file type) if desired; all that is required is a leading NODE_NUMBER column and appropriate column headers (these can be added by cutting and pasting from other files). Also, data in the external file can be easily read and written by the PLPROC parameter list processor.

Using USGMOD2ARRAY

USGMOD2ARRAY commences execution by prompting for the name of the name file pertaining to the MODFLOW-USG model of interest:

```
Enter name of MODFLOW-USG name file:
```

USGMOD2ARRAY reads all data recorded in this file and stores it in its memory. It then opens the DISU file that is cited in the name file (note that an unstructured grid is thereby assumed). From the DISU file it obtains the number of nodes and layers in the current model, as well as the distribution of nodes between model layers (from the NODELAY array).

Next USGMOD2ARRAY prompts for the name of a MODFLOW-USG input file that contains data for all layers of the model. This can be a DISU, BAS, BCF or LPF package input file that is cited in the name file. The prompt is:

```
Enter name of MFUSG BCF/LPF/BAS/DIS package input file:
```

USGMOD2ARRAY reads array data in a MODFLOW-USG input file in the same way that the USGPROP2TAB2 utility does. It assumes that the standard MODFLOW-USG layer-specific array header line is followed by text which includes the array type (for example "kx"), and the layer to which the current subarray pertains (for example "layer 2"). Note that:

- The same array type text identifier must be associated with instances of the array pertaining to all model layers ("kx" in the above example);

- The string "layer" followed by the layer number must also be present in the array identification text string for all model layers;

- Data for all model layers (and not just a subset of these layers) must be provided in the MODFLOW-USG input file.

Array identification text such as the above is added by most commercial MODFLOW-USG graphical user interfaces to the MODFLOW-USG data input files which they write. USGMOD2ARRAY's prompt is:

```
Enter text string for array recognition in this file:
```

In the above example, only the string "kx" would be provided in response to this prompt; the existence of the "layer" string is presumed. If the string by which the array is recognized contains a space, it must be placed between quotes when responding to this prompt.

MODFLOW-USG next asks whether the array data that it is required to read and re-formulate is integer or real. The prompt is:

```
Are these arrays integer or real? [i/r]:
```

It next asks for the names of the files which it must write:

```
Enter name for external array data file:
Enter name for new package input file:
Enter name for new MFUSG name file:
```

The new name file cites the new MODFLOW-USG package input file and the external array file containing the re-formulated array data. The new MODFLOW-USG package input file no longer holds the user-specified array data. However, in accordance with MODFLOW-USG protocol, a data header is retained in this file at all locations where layer-specific node subarrays previously existed within it. This header directs the MODFLOW-USG U1DREL or U1DINT subroutines to read the missing data from the external data file.

Uses of USGMOD2ARRAY

If hydraulic property or other data lies in its own file, it can be processed much more easily than if it resides in a file which it shares with other data. Thus this data can be processed as part of a composite model which is run by PEST. Because the data header that remains in the MODFLOW-USG package input file instructs MODFLOW-USG to read this data using free-field format, a pre-processor is given a high degree of latitude in how it writes this file. It can write it as a list confined to a single column (as does USGMOD2ARRAY), or as a series of layer-specific arrays. If it takes the latter option, data for each new MODFLOW-USG layer must begin on a new line.

See Also

See also USGPROP2TAB1 and USGPROP2TAB2.

USGMOD2OBS

Function of USGMOD2OBS

USGMOD2OBS does for MODFLOW-USG what MOD2OBS does for MODFLOW. It writes a bore sample file based on data recorded in a binary head/drawdown output file produced by MODFLOW-USG. It is important to note, however, that heads/drawdowns in this binary file must pertain to the nodes of a MODFLOW-USG unstructured (rather than structured) grid. (Use MOD2OBS if the grid is structured.)

Despite their similarities there is a significant difference between USGMOD2OBS and MOD2OBS. Factors used by USGMOD2OBS to interpolate from the unstructured grid to measurement points must be supplied by the user. In contrast, MOD2OBS calculates interpolation factors itself. (Calculation of interpolation factors is, of course, a far easier task for a structured grid than for an unstructured grid.)

As is discussed in Part A of this manual, a bore sample file records measured or calculated values at one or a number of sites, at one or many times for each site. USGMOD2OBS requires that the user supply such a file; this user-supplied bore sample file presumably records field-measured values at sites and times of interest. USGMOD2OBS generates a new bore sample file in which model-calculated heads/drawdowns are recorded at the same output times as in the user-supplied bore sample file (as long as these times lie within the model simulation time). Interpolation from times at which data is recorded in the MODFLOW-USG binary head/drawdown file to the times featured in the user-supplied bore sample file is linear in time. Data recorded in the USGMOD2OBS-produced bore sample file is thus the model-generated counterpart of the user-supplied bore sample file.

Using USGMOD2OBS

As for many other members of the Groundwater Data Utilities suite, a settings file named *settings.fig* must be present in the directory from which USGMOD2OBS is run. See Part A of this manual for a description of this file type. This informs USGMOD2OBS of the protocol which it must employ for reading and writing dates.

USGMOD2OBS commences execution with the prompt:

```
Enter name of node-to-bore interpolation file:
```

The name of an appropriate file should be supplied in response to this prompt. If a default filename for the node-to-bore interpolation file has been read from a filename file (named *files.fig*) resident in the current directory, that filename will appear with the above prompt. (The pertinent line of this file will say "node_to_bore_interpolation_file = *file*", where file is the name of the file; see Part A of this manual.) It can be accepted through pressing the <Enter> key or rejected by supplying the correct filename. Note that, as is the case for other members of the

Groundwater Data Utilities suite, responding to any prompt with "e" (for "escape") takes you back to the previous prompt. In this way mistakes can be quickly corrected.

The node-to-bore interpolation file contains, for each location (i.e. bore) to which interpolation must take place, the node number(s) and corresponding factors through which interpolation to the bore from nodes of the unstructured grid is implemented. Nodal interpolation factors for each bore location must sum to unity; USGMOD2OBS will generate an error message if this is not the case. For any given bore, node associations can extend across MODFLOW-USG layers. Either groundwater flow or CLN nodes may be cited, but not a mixture of both. Specification details for a node-to-bore interpolation file are provided in Part A of this manual. An example node-to-bore interpolation file follows.

```
419001A  1 28435 1.000
419003B  3 31985 0.425 11986 0.250 22286 0.325
A381     4 123 0.340 133 0.260 99 0.300 1005 0.100
```

Part of a node-to-bore interpolation file.

On any given run, it is not necessary that USGMOD2OBS undertake interpolation to all bores cited in a node-to-bore interpolation factor file. The user can select which bores will be involved in the interpolation process by providing the name of a bore listing file in response to the prompt:

```
Enter name of bore listing file:
```

Each bore cited in the bore listing file must also be cited in the node-to-bore interpolation file. If desired, the bore listing file can also be the node-to-bore interpolation file. If this is the case, then all bores cited in the node-to-bore interpolation file will also be cited in the bore sample file generated by USGMOD2OBS.

USGMOD2OBS next asks whether the interpolation factors provided in the node-to-bore interpolation file refer to groundwater flow (GWF) or connected linear network (CLN) nodes used in the MODFLOW-USG model:

```
Are the interpolation factors associated with GWF or CLN nodes [g/c]?
```

The response to this prompt should be "g" if nodes cited in the node-to-bore interpolation file are groundwater flow (GWF) nodes, or "c" if they are connected linear network (CLN) nodes. Note that the CLN process has its own separate numbering scheme in MODFLOW-USG input and output files (even though internally to MODFLOW-USG CLN node numbers are appended to the end of the GWF node number list). The unique number for each CLN node is given by the IFNO variable (which starts from 1) on the CLN package input file. (These may be connected to GWF nodes designated through the IGWNOD variable. Refer MODFLOW-USG documentation for further details.)

If a node goes dry in a MODFLOW-USG simulation it is assigned an easily-identified value (i.e. HDRY) which denotes this condition. This should be a very high value (see prompt below). If a dry node is used for interpolation to one or more bores, this

situation must be properly accommodated. USGMOD2OBS provides two options. One is denote any bore whose interpolated value is influenced by this node as itself dry. In this case USGMOD2OBS provides a value of "`dry_or_inactive`" for the bore's interpolated value. The other option is to re-calculate interpolation factors such that the dry node is omitted and the remaining interpolation factors sum to 1.0. The user chooses between these two options through his/her response to the following prompt:

```
Reapportion interpolation factors if any dry/inactive nodes [y/n]?
```

A response of "n" invokes the first of the above options whereas a response of "y" invokes the second. If all nodes from which interpolation takes place to a particular bore are dry, a value of "`dry_or_inactive`" is assigned to the bore regardless of which of these options is chosen.

Next USGMOD2OBS requests the name of an existing bore sample file, this presumably containing observed heads or drawdowns at observation wells. (Recall that it is the task of USGMOD2OBS to calculate the modelled counterparts to these.)

```
Enter name of bore sample file:
```

If the name a bore sample file appears in a filename file (named *files.fig*) which is stored in the current working directory, that name will appear as a default in the above prompt; it can be accepted by simply pressing the <Enter> key.

While the bore sample file whose name is supplied in response to the above prompt can contain data pertaining to more bores than those listed in the bore listing file, and record sample values over a time interval exceeding the model simulation time, it is a good idea to reduce the amount of redundant information present in that file to a minimum, as this will lower USGMOD2OBS memory usage and processing time.

As already stated, USGMOD2OBS undertakes spatial interpolation of model results to the locations of bores cited within the user-supplied bore sample file (and bore listing file), as well as temporal interpolation to the times at which samples were taken for each such bore. In doing this, USGMOD2OBS computes a model-generated "sample" corresponding to each measured "sample" in the user-supplied bore sample file. The set of such samples is recorded in bore sample file format.

Next, USGMOD2OBS prompts for the name of the binary MODFLOW-USG head/drawdown file which it must read:

```
Enter name of unformatted MFUSG-generated file:
```

It is the user's responsibility to ensure that MODFLOW-USG stores enough data in this file to allow accurate temporal interpolation to sample times listed in the user-supplied bore sample file. For times which do not correspond to model output times, USGMOD2OBS performs linear interpolation between times pertaining to arrays present in the MODFLOW-USG binary head/drawdown file. Through the setting of appropriate MODFLOW-USG Output Control variables, the user should ensure that

binary output is available at close enough time intervals for linear interpolation to be reasonably accurate.

Should the user inadvertently specify a binary MODFLOW-USG output file that pertains to a structured rather than an unstructured grid, USGMOD2OBS will cease execution with an appropriate error message. (The MOD2OBS utility should be used in this case.) An error message will also be issued if node numbers are cited in the node-to-bore interpolation file which do not appear in the binary head/drawdown file. It is the user's responsibility to ensure compatibility between grid specifications and node interpolation factors in other, less detectable, ways. (To promote ease of use, USGMOD2OBS does not read the DISU file for the current model, so other incompatibilities may go unnoticed if the user supplies an incorrect node-to-bore interpolation file, or if the binary head/drawdown file pertains to a different model.)

USGMOD2OBS needs to know the "threshold value" above which a cell is considered to be inactive or dry. If the absolute value of any nodal head or drawdown provided in the binary head/drawdown file exceeds this value, USGMOD2OBS presumes that the cell is dry or inactive. The treatment of dry and/or inactive cells is discussed above.

```
Enter inactive threshold value for numbers in this file:
```

USGMOD2OBS then asks a series of questions, the answers to which will allow it to calculate the date and time corresponding to each model output time. (Model output times are recorded in array headers in the binary head/drawdown file.)

```
Enter time units used by model (yr/day/hr/min/sec) [y/d/h/m/s]:
Enter simulation starting date [dd/mm/yyyy]:
Enter simulation starting time [hh:mm:ss]:
```

(Note that the date format used by USGMOD2OBS in these prompts depends on the contents of file *settings.fig* situated within the current directory. If this file is not present, USGMOD2OBS will not run.)

USGMOD2OBS then prompts:-

```
If a sample time does not lie between model output times, or if there
is only one model output time, value at the sample time can equal
that at nearest model output time:-

Enter extrapolation limit in days (fractional if necessary):
```

If a sample from a particular bore lies either before the first time of model output, or after the last time of model output, USGMOD2OBS cannot perform linear interpolation from model output times to the bore sample time. Hence it will calculate an "extrapolated" value at that time equal to the first or last model-calculated value respectively for that bore. However, there is a limit to the time over which such extrapolation can take place; this is set by the user's response to the above prompt.

Note that the earliest time at which MODFLOW-USG output is available is the end of the first time step. Hence if you require model output at a time as close as possible to the beginning of the simulation, make this first time step as short as possible.

Next USGMOD2OBS requests the name of the bore sample file which it must write:-

```
Enter name for bore sample output file:
```

USGMOD2OBS next carries out temporal and spatial interpolation to the sites and times cited in the user-provided bore sample file. It then produces a bore sample file of its own, with samples at exactly the same dates and times as those occurring within the user-provided bore sample file, but with model-generated numbers substituted for measured ones. It is important to note, however, that there may not be a sample in the USGMOD2OBS-generated bore sample file corresponding to every sample in the user-provided bore sample file. USGMOD2OBS does not generate a sample for a particular bore and time under the following conditions:-

- if a user-supplied sample precedes the earliest time of model output by an amount exceeding the extrapolation limit;
- if a user-supplied sample postdates the latest time of model output by an amount exceeding the extrapolation limit;
- if a bore appears in the bore sample file but not in the bore listing file.

As has already been discussed, interpolation to the site of a bore cannot take place if the bore is associated with one or more nodes that are deemed to be dry or inactive on the basis of the inactive/dry threshold specified by the user, and either (a) reapportionment of node-to-bore interpolation factors is not undertaken or (b) all nodes required for interpolation to the site of the bore are dry or inactive. Under such circumstances the value "`dry_or_inactive`" is assigned to the head/drawdown associated with that bore. Hence if a bore becomes dry during a PEST run in which USGMOD2OBS forms part of a composite model, the same instruction set can read the USGMOD2OBS-generated bore sample file as there will be no difference in the number of lines appearing in this file. However if the instruction set includes an instruction to read this dry/inactive head value an error condition will arise. PEST may then cease execution with an error message. If this is the case the resulting PEST error message will be such as to direct the user to the source of the problem. Alternatively, if the PEST LAMFORGIVE and/or DERFORGIVE variables are set, PEST will take the remedial actions specified in PEST documentation.

Uses of USGMOD2OBS

USGMOD2OBS was written to expedite use of PEST in calibration of a MODFLOW-USG model. A comparison between observed borehole data and its model-generated counterparts is easily achieved by running MODFLOW-USG followed by USGMOD2OBS as a composite model. Because USGMOD2OBS performs both spatial and temporal interpolation to the sites and times of measured data, and presents the results of its calculations in the same format as the measured data (i.e. as a bore sample file), comparison between the two datasets can be made with ease. When used in conjunction with PEST, USGMOD2OBS forms a vital component of the model calibration process. The PESTPREP, PESTPREP1 and PESTPREP2 utilities documented herein can be used to automate generation of an instruction file for

reading of a USGMOD2OBS-generated bore sample file, and for writing of the "observation data" segment of a PEST control file.

USGMOD2OBS is just as useful in steady-state MODFLOW-USG calibration as it is in transient calibration. However in this case there will normally be only one "time" at which model output is available, this being equal to the notional "elapsed time" since the beginning of the steady state simulation (this being defined in the MODFLOW-USG input dataset). In this case USGMOD2OBS cannot interpolate between neighbouring MODFLOW-USG output times to the time of a borehole head or drawdown measurement. Instead it can conduct only spatial interpolation to sample sites, assuming temporal coincidence of borehole sample times with model output times. In this case the user should ensure that the measurement bore sample file contains steady-state samples which all pertain to a date and time which is close to the notional model output time (i.e. within the use-specified temporal extrapolation limit). This is a simple matter if it can be assumed that steady-state conditions prevail on a certain date. Beware, however, of making the user-supplied extrapolation time too large, for then neighbouring samples in the user-supplied bore sample file may be close enough to the notional model output time to warrant inclusion in the USGMOD2OBS-generated bore sample file. If this occurs you must either decrease the extrapolation time, or assign duplicated observations a weight of zero. Alternatively, provide only one "measured" head/drawdown for each bore in the user-supplied bore sample file and set the USGMOD2OBS temporal extrapolation limit to a very high value in order to ensure that all sample times represented in this file are covered. (The latter is the recommended option.)

See Also

See also MOD2OBS, USGMOD2SMP, PESTPREP, PESTPREP1 and PESTPREP2.

USGMOD2OBS1

USGMOD2OBS1 behaves in an almost identical fashion to USGMOD2OBS. The only differences between these programs are the following.

1. Instead of asking for a threshold value through which dry or inactive cells are defined, USGMOD2OBS1 asks specifically for values which denote inactive cells (MODFLOW-USG variable HNOFLO) and those which denote dry cells (MODFLOW-USG variable HDRY).

2. When USGMOD2OBS1 reads the header to the first array contained within the MODFLOW-USG binary output file, it assumes that values which don't make sense indicate that the file was actually written by a double precision version of MODFLOW-USG. It then adjusts its behaviour to read such a file. If the values that it reads still don't make sense, then it terminates execution with an error message.

USGMOD2SMP

Function of USGMOD2SMP

USGMOD2SMP does for MODFLOW-USG what MOD2SMP does for MODFLOW. It reads a binary head/drawdown output file generated by MODFLOW-USG for an unstructured grid and writes a bore sample file containing heads/drawdowns spatially interpolated to the sites of user-supplied bores. Spatial interpolation takes place for all simulation times featured in the MODFLOW-USG binary head/drawdown file.

Despite their similarities, there is an important difference between the interpolation functionality provided by USGMOD2SMP and that provided by MOD2SMP. USGMOD2SMP employs a user-supplied set of node-to-bore interpolation factors whereas MOD2SMP calculates interpolation factors itself. (Calculation of interpolation factors is much easier for a structured grid than for an unstructured grid.)

Many programs documented in this manual are able to manipulate data contained in a bore sample file. A USGMOD2SMP output file is immediately available for processing by these programs.

Using USGMOD2SMP

Program USGMOD2SMP will not run unless a settings file (named *settings.fig*) is present within the directory from which it is invoked. As discussed in Part A of this manual, a settings file determines the manner in which dates are read and written by programs of the Groundwater Data Utilities suite.

USGMOD2SMP commences execution with the prompt:

```
Enter name of node-to-bore interpolation file:
```

The name of the appropriate file should be provided in response to this prompt. If a default filename for the node-to-bore interpolation file has been read from a filename file (name *files.fig*) resident in the current directory, that filename will appear with the above prompt. (The pertinent line of this file will say "node_to_bore_interpolation_file = *file*", where *file* is the name of the file; see Part A of this manual.) It can be accepted through pressing the <Enter> key or rejected by supplying the correct filename. Note that, as is the case for other members of the Groundwater Data Utilities suite, responding to any prompt with "e" (for "escape") takes the user back to the previous prompt. In this way mistakes can be quickly corrected.

As discussed in Part A of this manual, a node-to-bore interpolation file contains the node number(s) and corresponding nodal interpolation factor(s) for each user-supplied bore of interest. Nodal interpolation factors for each bore location must sum to unity. For a given bore location, node associations can extend across multiple MODFLOW-USG layers. Nodes may pertain to either the groundwater flow (GWF) process or

connected linear network (CLN) process of MODFLOW-USG, but not to a mixture of both of these.

An example node-to-bore interpolation file follows. See Part A of this manual for further details. Note that an error message will be issued if illegal node numbers (i.e. zero or negative) are cited in the node-to-bore interpolation file or if the nodal interpolation factors for a bore do not sum to unity.

```
419001A  1 28435 1.000
419003B  3 31985 0.425 11986 0.250 22286 0.325
A381     4 123 0.340 133 0.260 99 0.300 1005 0.100
```

Part of a node-to-bore interpolation file.

The user is able to select which of the bores cited in the node-to-bore interpolation file will appear in the final USGMOD2SMP output file by providing the name of a bore listing file in response to the prompt:

```
Enter name of bore listing file:
```

Each bore cited in the bore listing file should also be cited in the node-to-bore interpolation file. If desired, the bore listing file can also be the node-to-bore interpolation file; thus all bores cited in the node-to-bore interpolation file will take part in the spatial interpolation process.

USGMOD2SMP next asks whether the interpolation factors provided in the node-to-bore interpolation file refer to groundwater flow (GWF) or connected linear network (CLN) nodes in the MODFLOW-USG model:

```
Are the interpolation factors associated with GWF or CLN nodes [g/c]?
```

The response to this prompt should be “g” if nodes cited in the node-to-bore interpolation file are groundwater flow (GWF) nodes, or “c” if they are connected linear network (CLN) nodes. Note that the CLN process employs its own separate numbering scheme on MODFLOW-USG input/output files (even though internally to MODFLOW-USG CLN node numbers are appended to the end of the GWF node number list). The unique number for each CLN node is given by the IFNO variable (which starts at 1) on the CLN package input file. (These may be connected to GWF nodes designated through the IGWNOD variable. Refer to MODFLOW-USG documentation for further details.)

When carrying out spatial interpolation to a specific bore location from MODFLOW-USG nodes using interpolation factors supplied in the node-to-bore interpolation file, two options are available to address the situation whereby dry or inactive conditions are encountered at one or more of these nodes. The first option is to set the interpolated value for the bore to 1.1e35 if any node associated with the spatial interpolation process pertaining to that bore is itself dry or inactive. Alternatively, USGMOD2SMP can automatically reapportion user-supplied interpolation factors for those nodes that remain active and wet after omission of the offending dry/inactive nodes from the interpolation process. The adjusted interpolation factors sum to 1.0

while retaining their original ratios. USGMOD2SMP requires that the user select which of these options to take:

```
Reapportion interpolation factors if any dry/inactive nodes [y/n]?
```

USGMOD2SMP next prompts the user for the name of the MODFLOW-USG binary head/drawdown file which it must read:

```
Enter name of unformatted MFUSG-generated file:
```

Should the user inadvertently specify a head/drawdown file that is associated with a structured rather than an unstructured grid, USGMOD2SMP will cease execution with an appropriate error message. An error message will also be issued if node numbers are cited in the node-to-bore interpolation file which never appear in the binary head/drawdown file. It is the user's responsibility to ensure compatibility between grid specifications and node interpolation factors in other, less detectable, ways. (To promote ease of use, USGMOD2OBS does not read the DISU file for the current model so other incompatibilities may go unnoticed.)

USGMOD2SMP next asks:

```
How many different output times are represented in this file?
```

USGMOD2SMP needs to know the number of different output times so that it can dimension arrays appropriately before reading the MODFLOW-USG-generated binary head/drawdown file. If you are unsure of the contents of an unformatted MODFLOW-USG output file, use program USGARRDET to echo array headers found in this file. Alternatively, in response to the above prompt, supply a number that you are sure is greater than the number of output times represented in the head/drawdown file; if this number is not large enough USGMOD2SMP will soon inform you.

In order that it detect the presence of dry or inactive cells and, if necessary, readjust interpolation factors in the manner described above, USGMOD2SMP next prompts:

```
Enter blanking threshold value for arrays in this file:
```

Provide a positive number that is less than the absolute values of the user-supplied MODFLOW-USG variables HDRY and HNOFLO.

Before it can generate a bore sample file, USGMOD2SMP needs to know how to convert elapsed model simulation times to true dates and times. So it asks:

```
Enter time units used by model (yr/day/hr/min/sec) [y/d/h/m/s]:
Enter simulation starting date [dd/mm/yyyy]:
Enter simulation starting time [hh:mm:ss]:
```

(Note that the date format used in the second of the above prompts depends on the contents of the settings file *settings.fig*.) Then, after prompting:

```
Enter name for bore sample output file:
```

USGMOD2SMP reads the unstructured MODFLOW-USG binary head/drawdown output file and undertakes spatial interpolation to the sites of all bores listed in the

bore listing file in the manner described above. The outcomes of this interpolation process are recorded in bore sample file format to the file whose name is provided in response to the above prompt.

Uses of USGMOD2SMP

Because it performs the dual functions of interpolating model results to bore locations, and writing its results in bore sample file format, USGMOD2SMP makes model-generated data "look like" field data. Members of the Groundwater Data Utilities suite which process bore sample files are thus able to process model outcomes. In particular SMP2HYD can be used to construct data files that can be used by commercial plotting software for the plotting of borehole hydrographs. These hydrographs can be plotted on the same graphs as measured borehole hydrographs, thus enabling a comparison to be made between model-generated data and field data. Program SMP2SMP can be used to interpolate model-generated heads/drawdowns to the times at which measurements were taken of these quantities. Combined use of USGMOD2SMP and SMP2SMP thus emulates use of USGMOD2OBS. Programs PESTPREP, PESTPREP1 and PESTPREP2 can then be used to facilitate construction of a PEST input dataset.

See Also

See also SMP2SMP, USGARRDET, USGBUD2SMP and USGMOD2OBS.

USGNDTF2MIF

Function of USGNDTF2MIF

"NDTF" stands for "node data table file". As is explained in Part A of this manual, this type of file associates one or more items of integer or real data with each node of a finite difference grid. This data is arranged in tabular form.

USGNDTF2MIF reads a node data table file. It also reads an unstructured grid specification file in order to obtain information on grid geometry. It writes a MIF/MID file pair which contains data from both of these sources pertaining to a user-specified model layer. These files can be readily imported into a geographical information system (GIS) for spatial analysis or editing in a map context. Using functionality available through these platforms, this same model data can be re-written using other common GIS protocols, for example using the popular ESRI shape file. Alternatively (or as well), node data that is edited in the GIS can simply be exported in tabular form, whereby it is easily re-formatted using a text editor, ready for use by MODFLOW-USG.

Using USGNDTF2MIF

USGNDTF2MIF commences execution by prompting for the name of the unstructured grid specification file which it must read.

```
Enter name of unstructured grid specification file:
Enter layer number of interest:
```

Next it prompts for the name of a node data table file:

```
Enter name of node data table file:
```

As is explained in Part A of this manual, the first column of a node data table file must be labelled "NODE" or NODE_NUMBER". USGNDTF2MIF checks that this is indeed the case. If it is not, USGNDTF2MIF reports the error and ceases execution. If this is the only column that the node data table file contains then USG2NDTF will ask no more questions about this file; only node numbers will be recorded in the MIF/MID files which it writes. Alternatively, if there are other columns in the file, then USGNDTF2MIF next asks:

```
The following columns have been detected in the node data table file.
Indicate whether you would like pertinent data transferred to MIF/MID file.

   Data in column labelled "IBOUND"?  [y/n]:
   Does column contain integer or real data?  [i/r]:
   Use data from this column to filter cell output? [y/n]:

   Data in column labelled "Kx"?  [y/n]:
   Does column contain integer or real data? [i/r]:
   Use data from this column to filter cell output? [y/n]:

   etc.
```

Note that if the response to the first of any of the above trio of prompts is "n" then the second and third questions will not be asked. Note also that, as for any prompts issued by any of the members of the Groundwater Data Utilities suite, a response of "e" allows backtracking to the previous prompt.

If the answer to the third of the above trio of prompts is "y" then USGNDTF2MIF asks either of the following questions, the first if the column contains integer data and the second if the column contains real data. The questions are:

```
Enter value for no cell output:
```

or:

```
Enter abs(threshold) above which there is no cell output:
```

If the data is integer, and if the data value for a particular cell corresponds to the user-supplied value for no cell output, then that cell will not appear in the MIF/MID files written by USGNDTF2MIF. This provides a useful means of, for example, filtering out cells whose IBOUND value is zero. If the data is real, then cell filtering takes place if the absolute value of data pertaining to a cell is greater than the value of the user-supplied threshold. This can also provide a means of filtering out inactive cells, for the heads assigned to such cells by MODFLOW-USG are often values which are easily recognized by virtue of being excessively high or excessively negative. Often only one data column will be used for filtering. However if multiple columns are so designated, then the data value in any of the columns can cause removal of that cell from the MIF/MID file pair written by USGNDTF2MIF.

USGNDTF2MIF prompts for the filename base of the MIF/MID file pair which it must write:

```
Enter filename base for MIF/MID files:
```

An extension of "*.mid*" is appended to this filename base to formulate the name of the MIF file, while an extension of "*.mid*" is appended to formulate the name of the MID file. USGNDTF2MIF then writes the MIF and MID files and ceases execution.

It should be noted that if any node cited in the unstructured grid specification file for the model layer of interest is not cited in the node data table file, then USGNDTF2MIF detects this omission and ceases execution with an appropriate error message.

USGNDTF2MIF's final prompt may seem a little strange at first. (It is probably the reason that you are actually reading this manual.) It is:

```
Enter a suitable value for epsilon (<Enter> if 1E-3):
```

There are a number of processes within the USGNDTF2MIF algorithm which require it to determine whether two points (for example cell centres or vertices) have the same easting and northing. Two points may indeed have the same easting or northing, but the numbers which represent these eastings and northings in the computer may not be equal; alternatively, cell vertices at exactly the same horizontal location may be given

slightly different east and north coordinates by the program that wrote the unstructured grid specification file if calculation of vertex locations was complex. Hence tolerance must be allowed. Epsilon is that tolerance. Sometimes a broad tolerance should be provided. For example if the smallest grid cell size is 10m × 10m, then a tolerance of 1m will allow rather large errors in cell vertex calculations. In most cases however, a much smaller tolerance (e.g. 10^{-3}m) is fine; this is the default.

Uses of USGNDTF2MIF

As stated above, it is anticipated that the primary use of USGNDTF2MIF will be as a facilitator of node data editing in the GIS context. Most GIS packages (including the popular MAPWINDOWS and QGIS public domain packages) import MIF/MID files. They also allow export of the data held within these files in other commonly used formats. (Note that spatial data editing in some GIS packages can be faster if the data is imported as a shape file.)

Once model data has been edited against a map background, or assigned using GIS spatial functions, it can be exported. If node data is exported as single, layer-specific columns this data is easily readable by MODFLOW-USG. Alternatively it can be processed for importation into MODFLOW-USG (or other packages/models) using other protocols.

Note that cell filtering should not take place if data editing (rather than simply data display) is to take place in a GIS, for this will make re-importation of edited data into a MODFLOW-USG dataset harder because of the missing cells.

See Also

See also USGGRIDLAY, USG2VTK.

USGORTHCHEK

Function of USGORTHCHEK

The "ORTH" in USGORTHCHEK stands for "orthogonal". USGORTHCHEK reads an unstructured grid specification file. It visits all model cells within a user-specified layer of this file. It checks that each cell is a vertical prism and that each cell is a rectangle in plan view. This is established by verifying that each of its corners is a right angle in plan view.

Using USGORTHCHEK

Typical USGORTHCHEK prompts and responses are as follows.

```
Enter name of unstructured grid specification file: qt2.gsf
Enter layer number of interest: 3

Enter name for report file: temp.rec
- file qt2.gsf read ok.
- file temp.rec written ok.

Maximum cell internal direction cosine = 1.21211E-08
Cell at which this occurs              = 2936

Maximum number of cell vertices        = 4
```

Part of a report file is shown below.

```
NODE_NUMBER  NUMBER_OF_VERTICES  DIRN_COSINES_AT_VERTICES--->
    2533            4            7.57493E-10  7.57493E-10  7.57493E-10  etc.
    2534            4            7.57493E-10  7.57493E-10  7.57493E-10  etc.
    2535            4            7.57626E-10  7.57626E-10  7.57626E-10  etc.
    2536            4            7.57493E-10  7.57493E-10  7.57493E-10  etc.
    2537            4            7.57493E-10  7.57493E-10  7.57493E-10  etc.
    2538            4            7.57537E-10  7.57537E-10  7.57537E-10  etc.
    2539            4            7.57670E-10  7.57670E-10  7.57670E-10  etc.
    2540            4            7.57493E-10  7.57493E-10  7.57493E-10  etc.
    etc.
```

Part of a USGORTHCHEK report file.

If orthogonality prevails then all direction cosines should be very low. They may not be exactly zero due to round-off error. Note that the arrangement of vertices on any one line of a report file is the same as their arrangement in files generated by USGGRIDLAY and USGNDTF2MIF.

Uses of USGORTHCHEK

An unstructured grid may have cells of different sizes, but may still be comprised of rectangular cells. This will occur, for example, if it is built from a regular grid through quadtree refinement of the latter. Certain types of model pre- and post-processing can be simplified if a rectangular cell structure prevails.

See Also

See also USGGRIDLAY and USGNDTF2MIF.

USGPROP2TAB1

Function of USGPROP2TAB1

USGPROP2TAB1 stands for "USG-properties-to-table-option-1". It reads a MODFLOW-USG input file which contains hydraulic properties or integer values for cells within the finite-difference grid. In many cases this will be the input file for a package such as LPF or BASIC. It reads all arrays in which cell properties of a certain type are listed (for example arrays of layer-specific horizontal hydraulic conductivity or IBOUND). It writes all such properties in a single table, ordered according to node. This table can then be read by programs such as USG2VTK for three-dimensional visualization of hydraulic property data.

Using USGPROP2TAB1

In a file such as the MODFLOW-USG LPF package input file, hydraulic properties such as specific yield and storativity are normally found in a number of arrays, each pertaining to a different layer of the model grid. Where the grid is unstructured, the number of elements in each of these arrays may be different, as an unstructured grid does not need to possess the same number of elements in each of its layers. For a particular property type, USGPROP2TAB1 reads all of these arrays. It then writes an output file which contains two columns, each of which begins with a header. The first contains node numbers (numbered from 1 to the number of nodes in the grid). The second contains elements read from these arrays.

USGPROP2TAB1 can read integer arrays as well as real arrays. Hence if it reads the BASIC package input file for a MODFLOW-USG model, it is able to write a table of nodal IBOUND values.

Correct operation of USGPROP2TAB1 relies on a number of assumptions. However these assumptions are often met in practice. The first of these assumptions is that the program which wrote the MODFLOW-USG input file has included a text specifier for each array on the same line as the array header. Arrays are generally read by the MODFLOW-USG U2DREL (for real numbers) or U2DINT (for integers) subroutines. Use of these subroutines requires that array headers follow a certain protocol. This protocol does not require that the following array be named. However many MODFLOW graphical user interfaces write their own text descriptor following the standard U2DREL or U2DINT header. This supplementary header may be something like "HK array for layer 3", or "IBOUND array for layer 1". Use of USGPROP2TAB1 depends on the existence of this text descriptor.

USGPROP2TAB1 also assumes that the array begins on the line immediately following the header. Thus if the CONSTANT or OPEN/CLOSE protocol is employed for storage of the array on the MODFLOW-USG input file, USGPROP2TAB1 will not operate correctly. It will report an error message instead.

USGPROP2TAB1 is "dumb" in the sense that it knows nothing about the construction of the model grid. The only knowledge that it possesses is of the total number of cells in the grid; it obtains this information from the user. It does not know the number of cells within each layer of the MODFLOW-USG grid. It obtains this information as it operates.

When USGPROP2TAB1 finds the user-specified array designator text string on any line of a MODFLOW-USG input file, it assumes that a layer-specific array begins on the next line. It reads numbers from that and subsequent lines until one of the following things happens:

1. A line contains a character string that cannot be read as a number.

2. The end of the file is encountered.

If it encounters either of these conditions it assumes that it has tried to read a line too far, and ends its collection of element values for the particular layer at the previous line. It is very important to note that, for this process to work, numbers in arrays must be separated from each other by a space; hence array elements must be readable using free-field formatting.

Once it has reached the end of a layer specific "partial array" in this manner, USGPROP2TAB1 then continues scanning the MODFLOW-USG input file looking for further occurrences of the user-designated string. If it finds another occurrence it repeats the above procedure, assuming that the next array designated by the same text header belongs to the following layer. It appends the nodal property values that it reads from this next layer-specific array to the end of the list of values that it has already read.

Hopefully, when this process is complete, USGPROP2TAB1 will have read a value for every node in the grid. If it finds more values than nodes, it informs the user of this. It also informs the user if it finds fewer array values than nodes; in the latter case the remainder of the array is filled with user-specified default values.

Typical USGPROP2TAB1 prompts and responses are as follows:

```
Enter name of MODFLOW-USG model input file: umodel.lpf

Enter text string for array recognition in this file: hk
Are these arrays integer or real? [i/r]: r

Enter total number of nodes in grid/network: 10092

Enter name for node data table file: hk.dat
Enter value to assign to uncited nodes: 1.1e32
```

The text string used to inform USGPROP2TAB1 that a layer-specific partial property array follows must provide unique identification of these arrays. It must occur nowhere else within the MODFLOW-USG input file except in headers to arrays which are of this type. Optionally, this text string can be surrounded in quotes when responding to the above prompt.

Note that USGPROP2TAB1 can also read a CLN package input file. In that case the number of nodes supplied by the user must pertain to the CLN network rather than to the MODFLOW-USG grid.

Uses of USGPROP2TAB1

Despite some of the restrictive assumptions embodied in its design, use of USGPROP2TAB1 provides a means of quickly re-formatting model property and integer data in a way that is immediately available for the use of programs such as USG2VTK. Unlike programs such as USGPROP2TAB2, its use does not require that the user have knowledge of the number of nodes in each model layer.

See Also

See also USGPROP2TAB2 and USG2VTK.

USGPROP2TAB2

Function of USGPROP2TAB2

USGPROP2TAB2 performs a similar role to USGPROP2TAB1 in that it reads a series of layer-specific integer or real arrays from a MODFLOW-USG input file and records the contents of these arrays as a single collective table. However it uses slightly different protocols for the reading of array data from those used by USGPROP2TAB1. In particular, it reads these arrays in the same way as MODFLOW-USG does, gaining the information to do this from the header to each array. Through keywords such as CONSTANT, INTERNAL and OPEN/CLOSE, together with accompanying format specifiers and multipliers, many different options for the recording of array data are available. USGPROP2TAB2 understands the above keywords, and can read corresponding arrays accordingly.

The table recorded by USGPROP2TAB2 in its output file includes a column of node layer numbers, in addition to array data read from MODFLOW-USG input files.

Using USGPROP2TAB2

USGPROP2TAB2 commences execution with the prompt:

```
Enter name of MODFLOW-USG nodes-in-layer file:
```

A nodes-in-layer file is described in Part A of this manual. It records the number of nodes within each layer of an unstructured MODFLOW-USG grid. USGPROP2TAB2 requires this information so that when the CONSTANT keyword is provided in a layer-specific array header, it knows how many array elements the constant value pertains to.

USGPROP2TAB2's next prompt is:

```
Enter name of MODFLOW-USG model input file:
```

This may be, for example, an LPF input file, a DISU file or a BASIC package input file. It is assumed that the file contains a series of arrays, with each such array providing information for all elements within a single layer of the grid. Properties of more than one type may be represented in different series of arrays, all residing within the one MODFLOW-USG input file.

In a MODFLOW-USG input file, each array is preceded by a header. This instructs MODFLOW-USG how to read the array. USGPROP2TAB2 expects the first item on any such header line to be CONSTANT, INTERNAL or OPEN/CLOSE. See MODFLOW-USG documentation for the meanings of these keywords, and for the data input protocols which they signify.

Despite the fact that it is not standard MODFLOW protocol, USGPROP2TAB2 expects the MODFLOW array header to be supplemented with extra header data.

Many MODFLOW graphical user interfaces write this supplementary header so that the user can know the nature of the data that each array contains. For example, the normal MODFLOW header may be followed by text such as "HK array for layer 2". USGPROP2TAB2 requires this extra header information so that it, too, can identify arrays of interest. It reads only those arrays to which it is directed by the user. These arrays are identified through a character string contained within the supplementary header that is unique to each such augmented header.

USGPROP2TAB2 prompts:

```
Enter text string for array recognition in this file:
```

Suppose the user responds to this prompt with "hk". The response is case-insensitive; quotes are nonessential unless the string contains a space. Following this user-supplied response, USGPROP2TAB2 scans the MODFLOW-USG input file until it finds the string "hk" on a line of this file. It will assume that it has thereby found an array header line and will process it as such, looking for the strings CONSTANT, INTERNAL or OPEN/CLOSE at the start of the line. Before it does this however it looks for the string "layer" on the same line as "hk". It then looks for an integer immediately following the "layer" string and reads it as the layer number. Having acquainted itself with the layer number to which the ensuing array pertains it then knows how many elements it must read from that array, or to how many array elements it must assign a CONSTANT value. If the string "layer" and an ensuing integer are not found on this line, USGPROP2TAB2 terminates execution with an appropriate error message.

Next USGPROP2TAB2 asks:

```
Are these arrays integer or real? [i/r]:
```

It requires this information for obvious reasons. USGPROP2TAB2's final prompts are:

```
Enter name for node data table file:
Enter value to assign to uncited nodes:
```

If there are fewer arrays pertaining to the requested property type in the MODFLOW-USG input file than there are layers in the model grid (as read from the node-in-layer file), USGPROP2TAB2 fills in missing elements in its output table with the number provided in response to the second of the above prompts.

Uses of USGPROP2TAB2

Data in tables written by USGPROP2TAB2 are recorded in node order. These tables can then be read by USG2VTK so that layer property data can be included in a VTK file and then displayed.

USG2VTK can read a file containing many property data columns. USGPROP2TAB2 writes only one column of property data however. Hence multiple USGPROP2TAB2 runs are required to list node data for multiple property types. Columns in different USGPROP2TAB2-generated files can easily be placed into a single multi-column file

using text editors that allow column cutting and pasting. Microsoft EXCEL could also be used for this purpose.

See Also

See also USGPROP2TAB1 and USG2VTK.

USGPTINGRID

Function of USGPTINGRID

USGPTINGRID ("PTINGRID" stands for "point in grid") reads a bore coordinates file. This file assigns a character identifier, (*x*,*y*) coordinates, and a model layer number to each of a series of points. See Part A of this manual for full specifications of this file type. USGPTINGRID also reads an unstructured grid specification file. It then locates each point within the unstructured grid; it does this by calculating the cell number (equivalent to the node number) in which each point lies. Optionally it also reads a node data table file which assigns values to cell nodes; see Part A of this manual for specifications of this type of file. If so, the output file written by USGPTINGRID contains, as well as the node number associated with each point, the value assigned to that node in the node data table file.

Using USGPTINGRID

Like many of the utilities that are written to support use of MODFLOW-USG, USGPTINGRID commences execution by prompting for the name of an unstructured grid specification file. The prompt is:

```
Enter name of unstructured grid specification file:
```

Next it prompts for the name of a bore coordinates file. If the response to this, or any other prompt, is "e", the user is taken back to the preceding prompt. USGPTINGRID asks:

```
Enter name of bore coordinates file:
```

Next it asks for the name of a bore listing file. This contains just a single list of bore identifiers; see Part A of this manual for details. This file performs a similar role to that which it performs for many other Groundwater Data Utilities, in that it acts as a filter for points recorded in the bore coordinates file; only those bores which are featured in the listing file are subject to further processing. If a bore is cited in the listing file but not in the bore coordinates file, an error condition arises; USGPTINGRID then ceases execution with an appropriate error message.

USGPTINGRID's next prompt is:

```
Read column of a node data table file? [y/n]:
```

If the answer is "n" then USGPTINGRID will record in its output file only the cell in which each point cited in the bore listing file lies. However if the response is "y", then USGPTINGRID prompts for the name of a node data table file, and the column of this file from which it must obtain node data; columns are identified though their headers. It also asks whether the data comprising the column is integer or real. The prompts are:

```
Enter name of node data table file:
```

```
Enter column header:
Does column contain integer or real data? [i/r]: r
```

Finally USGPTINGRID prompts for the name of the file which it must write. It asks:

```
Enter name for output file:
```

The USGPTINGRID output file contains up to six columns of data. The first four of these echo data read from the bore coordinates file. They contain, respectively, bore identifiers, eastings, northings and layer numbers. The next column contains the node which occupies the cell in which each bore lies. If a node data table file is read, then a further column is recorded in the USGPTINGRID output file. This contains the real or integer values associated with cells within which bores lie as read from the user-specified column of the node data table file.

Note that if a point in the bore coordinates file does not lie within the grid, and/or if a value for the cell in which a point lies is not supplied in the node data table file, appropriate text indicating this condition is recorded in the appropriate place in the USGPTINGRID output file.

Uses of USGPTINGRID

The most obvious use of USGPTINGRID is to indicate where, in an unstructured grid, a pumping or observation well lies. In the former case, knowledge of this is essential for construction of an appropriate well (or CLN) input data file. In the latter case a model-generated node data table file (written perhaps by USGBIN2TAB_H) can be read to obtain model-generated heads or drawdowns for the cells in which observation wells lie.

See Also

See also USGBIN2TAB_H and USGGRIDLAY.

USGQUADFAC

Function of USGQUADFAC

"QUADFAC" stands for "interpolation factors for a qaudtree-refined grid". USGQUADFAC generates these factors for the use of programs USGMOD2OBS and USGMOD2SMP. In those parts of the model domain where the grid is uniform (or if the grid is entirely uniform), interpolation is bilinear – this being the same as that undertaken by the MOD2OBS and MOD2SMP utilities from nodes of a structured grid. However this interpolation scheme is modified in areas of transition between different grid cell spatial densities. As is discussed below, interpolation may take place from three model nodes to the site of a well rather than from four model nodes at certain places within the model domain in order to maintain lateral continuity of the interpolation process along any transect through the model domain.

USGQUADFAC requires that the following conditions be met by the unstructured model grid.

1. The unstructured grid must be rectilinear. That is, the boundaries of all model cells must point in either of two directions, these directions being at right angles to each other. This condition can be checked using the USGORTHCHEK utility.

2. One model cell must not connect to any more than two cells across a model cell boundary.

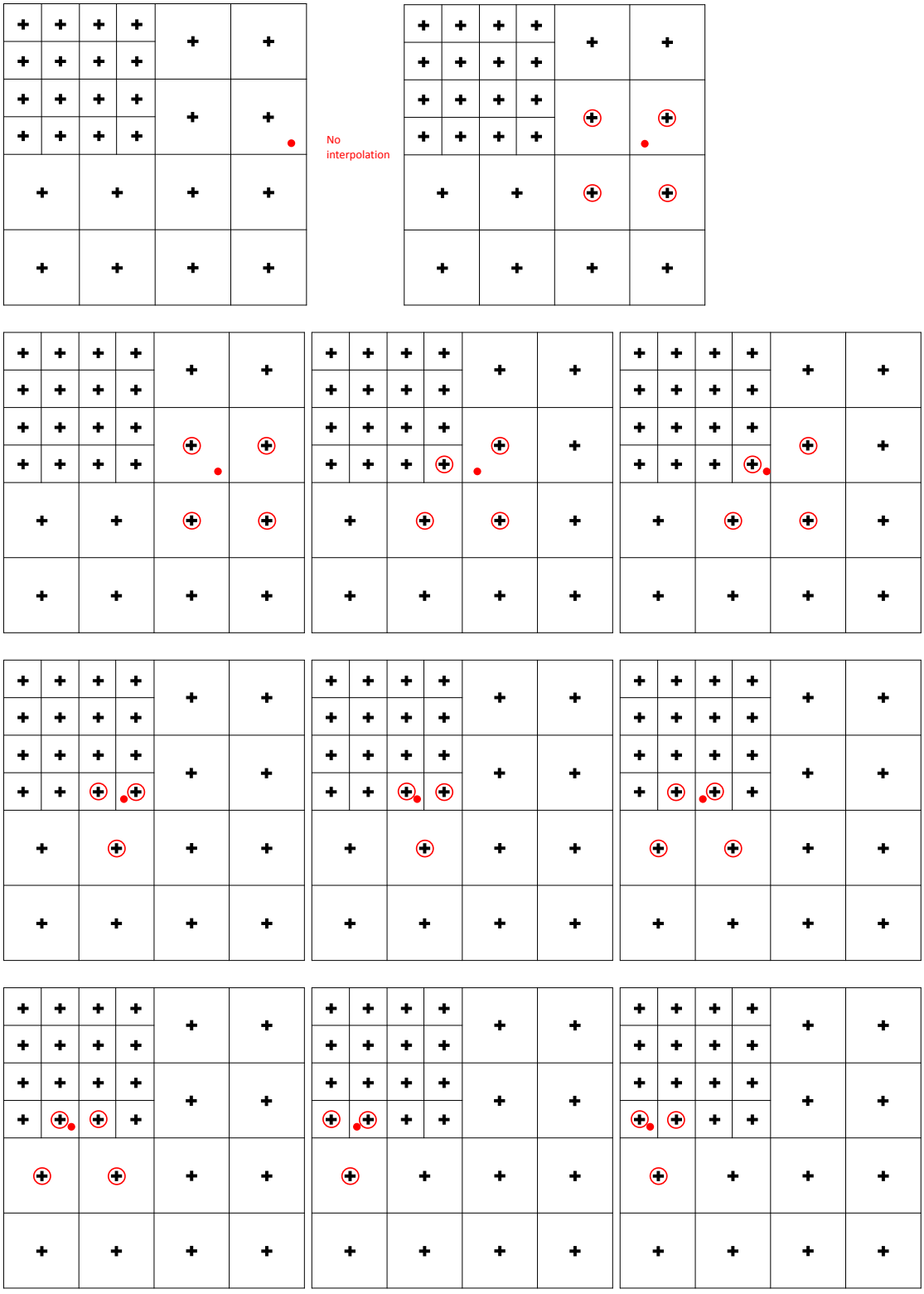
USGQUADFAC also requires that the unstructured grid specification file which it reads specify that each model cell have eight vertices, with four pertaining to the upper boundary of the cell and four pertaining to the lower cell boundary. In plan view each of these groups of four vertices must define a rectangle. In practice, it is not impossible that one of these upper and lower rectangles is represented by more than four vertices. For example a vertex may be introduced to the midpoint of the side of a rectangle to accommodate the fact that the cell abuts two smaller cells along that side. This allows the upper surface of the larger cell to match those of its neighbours along the join between them if the upper/lower surfaces of the smaller cells represent steep topography more faithfully. USGQUADFAC does not accommodate this situation; if this is a problem, please consult the author and the code will be amended to accommodate this situation.

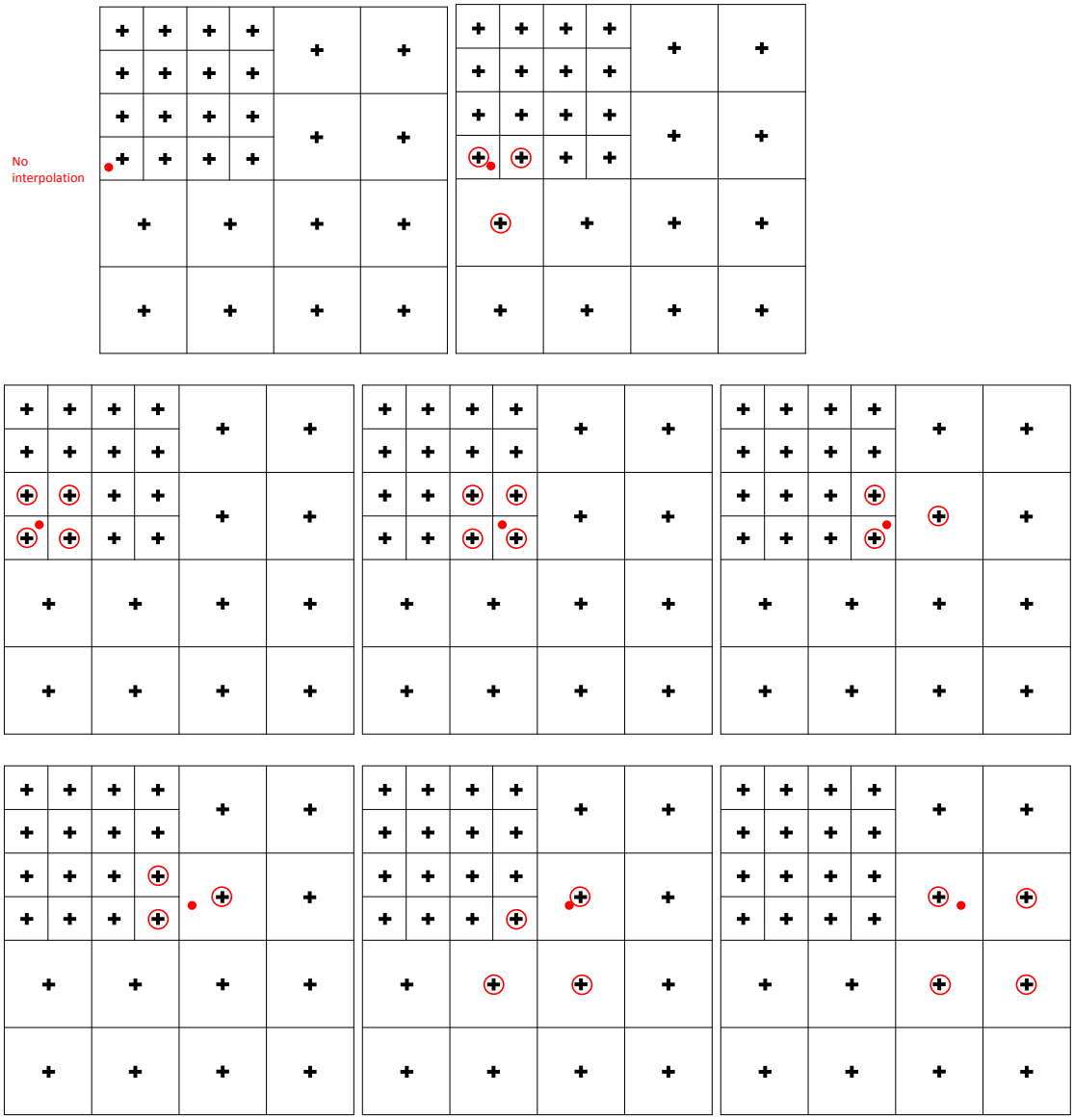
Using USGQUADFAC

General

As stated above, where possible, interpolation factors calculated by USGQUADFAC implement bilinear interpolation from the four cell centres that surround a user-supplied interpolation point (normally an observation well). Where the grid is locally structured, these cell centres define a rectangle. In parts of the grid where cell density

is transitional, these four points form a quadrilateral whose sides are not parallel. In some transitional areas interpolation takes place from only three cell centres in a way that maintains lateral continuity as the interpolation point is displaced. See the figure below.





In each of the above sub-figures cell centres are represented by crosses. The point to which interpolation takes place is represented by a red dot. Red circles denote cell centres from which bilinear interpolation takes place to that point.

Interpolation will not take place to a point that is not "surrounded" by at least three model cell centres. Such points are omitted from the interpolation factor file which is written by USGQUADFAC; a warning of this condition is written to the screen.

Prompts and Responses

As for all other members of the Groundwater Data Utility suite, you can backtrack to a previous prompt by responding to the current prompt with "e" (for "escape").

USGQUADFAC commences execution with the prompt:

```
Enter name of unstructured grid specification file:
```

USGQUADFAC obtains the coordinates of all model cell centres and vertices from this file. As stated above, it will cease execution with an appropriate error message if other than eight vertices are ascribed to any model cell.

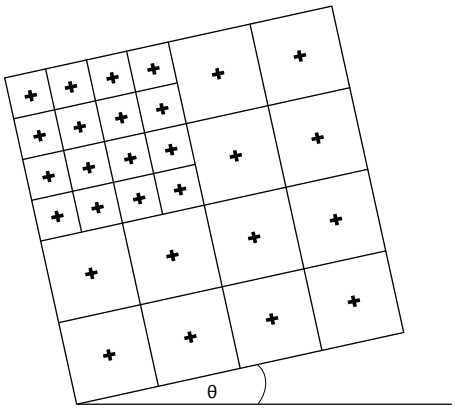
It then asks:

```
Are grid cell boundaries N-S and E-W? [y/n]:
```

If the boundaries of all model cells are aligned in either the E-W or N-S directions, then respond to this prompt with "y". However if you respond with "n" USGQUADFAC asks for the angle of rotation of the grid. Its prompt is:

```
Enter rotation (anticlockwise) of grid:
```

Provide a number between -90 degrees and 90 degrees. The rotation angle is illustrated in the figure below.



θ is the grid rotation angle.

Conceptually, USGQUADFAC could determine the grid rotation angle itself. But the programming is easier if the user supplies the rotation angle. In subsequent processing USGQUADFAC checks for cell rectilinearity. It also rotates the grid (and borehole coordinates) so that it can work in a coordinate system in which the model grid is oriented in an E-W / N-S direction. If violations of this condition are detected in the rotated grid it reports this condition to the user and ceases execution.

USGQUADFAC next asks:

```
Enter name of unstructured MODFLOW-USG discretization file:
```

The MODFLOW-USG discretization file is normally written by the graphical user interface which was used to build the model. USGQUADFAC reads the IAC and JA arrays from this file so that it is completely informed of cell connectivity.

Next USGQUADFAC asks:

```
Enter name of bore coordinates file:

Enter name of bore listing file:
```

As for other members of the Groundwater Data Utility suite, the bore listing file is used to select a subset of bores from the bore coordinates file for processing. If desired the bore listing file can be the same as the bore coordinates file. Interpolation factors are calculated for all bores listed in the bore listing file.

Next USGQUADFAC prompts for the name of the file that it must write. This file contains bilinear interpolation factors. It is readable by the USGMOD2OBS and USGMOD2SMP utilities. The prompt is:

```
Enter name for node-to-bore interpolation file:
```

USGQUADFAC's final prompt may seem a little strange at first. (It is probably the reason that you are actually reading this manual.) It is:

```
Enter a suitable value for epsilon:
```

There are a number of processes within the USGQUADFAC algorithm which require it to determine whether two points (for example cell centres or vertices) have the same easting or northing (possibly after rotation of the grid to introduce N-W / E-W grid cell edge alignment). Two points may indeed have the same easting or northing, but the numbers which represent these eastings and northings in the computer may not be equal, especially if those numbers are the outcomes of re-orientation calculations undertaken by USGQUADFAC or the program that wrote the unstructured grid specification file in the first place. Hence tolerance must be allowed. Epsilon is that tolerance. A broad tolerance is suggested. For example if the smallest grid cell size is 10m × 10m, then a tolerance of 1m is suggested, through a much smaller tolerance (e.g. 10^{-3}m) would probably be fine.

USGQUADFAC then performs its calculations, recording its progress to the screen. If the MODFLOW-USG grid is large and many wells are featured in the bore coordinates file, then these calculations may take a while. So a little patience may be required.

Uses of USGQUADFAC

The primary purpose of USGQUADFAC is to provide the means by which USGMOD2SMP and USGMOD2OBS can calculate heads and drawdowns at well locations though interpolation of these quantities from a MODFLOW-USG grid. The MODFLOW-USG grid can be regular, or quadtree refined.

Note that interpolation factors calculated by USGQUADFAC take no account of the active/inactive status of cells from which interpolation may take place to an observation well. Flexibility of grid design in MODFLOW-USG allows for elimination of inactive cells; if an observation well lies in such a cell (which is therefore outside the grid), no interpolation factors are calculated for that cell. However where an inactive cell is represented in a MODFLOW-USG grid, then interpolation factors may include that cell as USGQUADFAC is not aware of its active/inactive status. However, as is documented herein, USGMOD2SMP and USGMOD2OBS can recognize the inactive status of that cell through the values of

heads or drawdowns that are associated with it. They can adjust USGQUADFAC-calculated interpolation factors accordingly.

See Also

See also USGORTHCHEK, USGMOD2SMP and USGMOD2OBS.