User guide for *mfLab*

T.N. Olsthoorn
tolsthoorn+mfLab@gmail.com
+31-6-20440256

Delft University of Technology
Civil engineering and Geosciences
Stevinweg 1, 2626 CN Delft, The Netherlands
www.tudelft.nl

Waternet
Vogelenzangseweg 21, 2114 BA Vogelenzang, The Netherlands
www.waternet.nl

13 May 2013

Contents

Abstract

mfLab is an open and flexible software environment that combines Matlab (http://www.mathworks.com/products/matlab/) with the also open source MODFLOW suite of groundwater simulators (http://water.usgs.gov/software/lists/groundwater/). A working copy can be obtained from http://code.google.com/p/mfLab by a checkout with Subversion (http://sourceforge.net/apps/trac/sourceforge/wiki/Subversion).

mfLab stands for "MODFLOW Laboratory" as it it generates input files for the MODFLOW family of programs, i.e. MODFLOW, MT3DMS, SEAWAT and others, as well as related packages to perform any kind of groundwater simulation. The input files for these codes are generated using the Matlab routines that are the foundation of *mfLab*. The input files are generated after the model has been defined in the Matlab environment. Other *mfLab* routines read in the binary output produced MODFLOW etc. and will be post-process and visualize them in Matlab. *mfLab* can also read legal input files of the MODFLOW family made by external programs such as GUI's. Therefore, existing models made by others can be readily read in, visualized and analyzed.

mfLab will not be limited to the Matlab environment, although it currently does for practical reasons. Although Matlab is free for students at the TU-Delft where I teach, this is not the case outside communities that benefit from site licence. However the wealth of Matlab and its user inteffface are believed to be substantially stronger than its free competitors, which are Peyton, Octave and Scilab.

I invite anyone interested to join further development and extension of *mfLab* with the ambition to provide full and efficient access to the whole set of MODFLOW-related programs and their packages, so as to remove any limitations with respect to exploitation of this wealth of groundwater simulators.

I also invite anyone to join porting *mfLab* to and from other open-source scientific programming and visualization environments like *Octave*, *Scilab* and *Python* (see URLs at the end of this abstract).

mfLab is available as freeware under the GNU free software licence http://code.google.com/p/mfLab, where you can download your working copy by a so-called checkout in *svn* (Subversion version control program). Subversion comes installed on the Mac; Windows users are advised to install *Tortoise svn* to that end, which is a beautiful user interface to *svn* used by Google to service its free-software site http://code.google.com.

Useful URLs: http://code.google.com/p/mflab/, http://water.usgs.gov/software/lists/groundwater/, http://www.octave.org, http://www.scilab.org/, http://www.python.org/, http://www.mathworks.com/, http://subversion.tigris.org/, http://www.tortoise.sourceforge.net,

Introduction

The **MODFLOW** family of groundwater simulation programs, abbreviated in this document to *mf++*, and their related packages comprise the different versions of MODFLOW (known as *mf96*, *mf2k*, *mf2005*, modflowNWT and modflowUSG), and related programs like MT3DMS, SEAWAT etc., and all the packages developed for them to add processes and different types of boundary conditions. These programs provide an unrivaled groundwater simulation suite capable of modeling virtually every groundwater situation. Executables as well as their source code, are freely available, which made them the most widely used groundwater models worldwide. Their openness and versatility has mobilized a large active user community in practice and research. Many have contributed extensions, adding to their versatility. The biannual MODFLOW conferences held at the Colorado School of Mines bring people together exchanging experiences, views, additions and developments. *mf++* keeps well up with developing practical needs, such as calibration, uncertainty predictions, density driven flow, heat flow and transport, including chemical processes, groundwater management, karst groundwater, groundwater-river interaction just to name a few. Hence, it is a good source for teaching students the internals and possibilities of groundwater simulation.

At the same time, using the software in practice may be overwhelming, especially at first sign. Construction of input files often seems a challenge. The large number of oftentimes cryptic parameters and switch flags that need to be understood seems embarrassing. Without some tool to handle the complex input requirements one inevitably feels lost. To make groundwater *mf++* modeling accessible, many commercial graphical user interfaces, so-called GUIs, have been developed. While these fulfill an excellent job, they have limitations, which we intend to overcome with *mfLab*. To name a few:

- Each GUI has its personal complexity. There is no general standard. Learning one does not mean knowing the other.

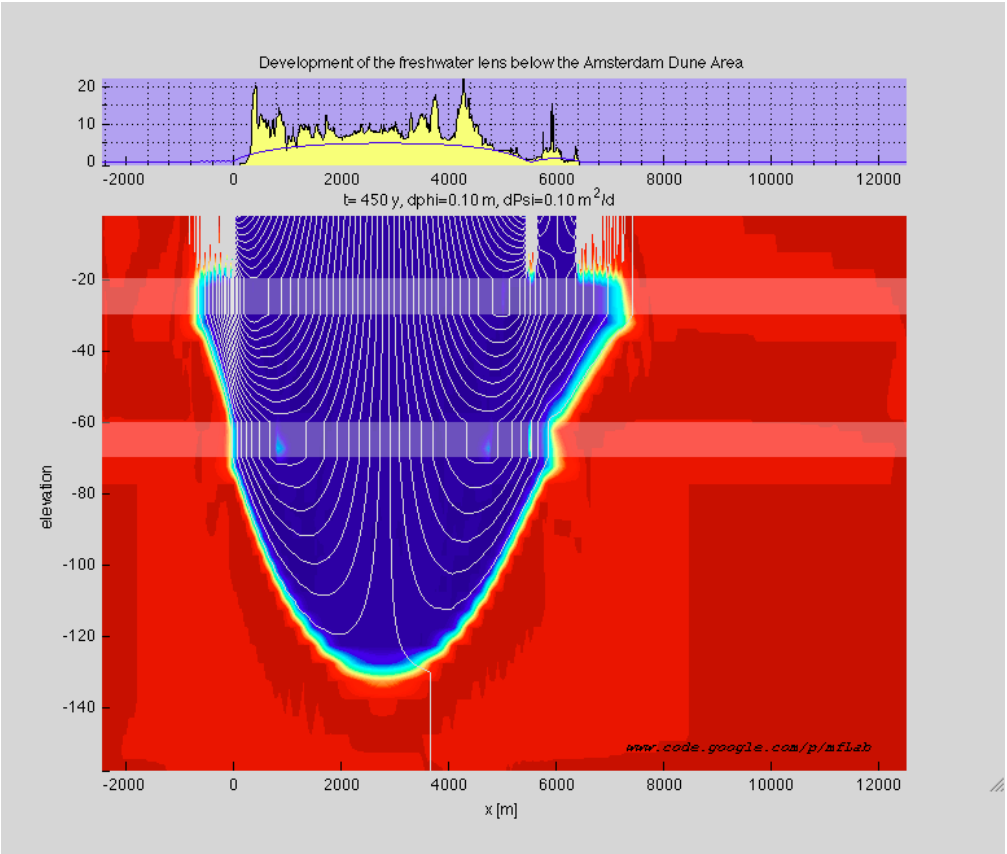


Figure 1: Example of cross section comptuded with mflab using Seawat. See mfLab>examples>sqt_v4>freshwater_lens. Concentrations are shown overlayed by streamlines.

- Some GUIs are rather expensive, making them inaccessible for students or for use in developing countries.
- Free GUIs like PMWIN are very useful, but somewhat outdated as current support is limited to only their recent commercial version.
- GUIs tend to lock-up users and limit them to only their implemented options.
- GUIs generally offer limited access to the the underlying simulators. They may hinder specific use and visualization. GUIs are generally unsuitable for research, but may be excellent for projects.
- GUIs tend to lack model-version control and, therefore, quality assurance is sometimes impossible or insufficient.
- GUI's tend to store all grid files, which takes up a lots of hard disk space and represent, in fact, tons of redundant data. *mfLab* only needs to store the 2 mfiles *mf_ adapt.m* and *mf_ analyze.m* and the Excel workbook *<<basename>>.xls* with the parameters, which usually is at most several hundred kilobytes small, regardless of the size of the model that these files generate.

Quality control requires the work flow to be completely documented and tractable. A GUI that does not register all changes the user made to the model is insufficient for quality control.

mfLab aims to provide efficient and unlimited access to *mf++* codes and packages while providing full reproducibility. This is considered indispensable for research as well as in everyday modeling practice. To achieve this, I chose the Matlab environment for its implementation. This way, Matlab's open high-level programming and visualization environment is exploited.

Matlab's only disadvantage is its fairly high price. However, there are freeware alternatives, namely, *Scilab*, *Octave*, *Python* and possibly others. I strongly stimulate porting *mfLab* to these environments. Porting to *Octave* and *Scilab* should be simple as they are almost line by line compatible. However, my choice for Matlab has been a practical one, as the TUDelft (www.tudelft.nl/en/), where I teach, has a Matlab site license. So all its students and many of its staff have experience with it. For short, *mfLab* is fully available under the GNU free software v3 licence from http://code.google.com/p/mflab. Use *svn* (see subversion.tigris.org) to get your working copy from the code.google.com server.

How does *mfLab* work?

1 Basic philosophy

To construct an *mf++* model, we have to define its grid, conductivity arrays, its boundary conditions and so on, and then generate the necessary input files for the particular simulator code. Matlab has a workspace in which the arrays that comprise the model can be readily constructed, tested visualized and documented. At the same time this model construction is stored in a script (a so-called m-file), so that a simulation can be reproduced at any moment in the future by rerunning this script.

This workflow process allows reproducible improvements along with the progress of a project, which is an essential feature of this environment. The expression strength of Matlab allows building a model in just a few lines, which keeps model construction conceptually simple, on which model size has no effect. It is just as straightforward to build a 1000x1000x100 cells model as one that has only 10x10x2 cells. Matlab helps transparent construction of arbitrary complex models.

Once the model is constructed, *mfLab* generates the input files needed to run the *mf++* programs and launches the target executables.

After the program(s) have finished, *mfLab* will read their (often binary) output and makes the results available in the Matlab environment, where it is post-processed and visualized, utilizing once again the strength and versatility of Matlab.

Users of *mf++* programs are often overwhelmed by the large set of cryptic (hard or impossible to remember) parameters and flags that are necessary to fine-tune these models. Dealing with them in a consistent and manageable way is a prerequisite for every modeling environment. *mfLab* solves this by using an Excel workbook as a general multi-page container for these parameters. (Excel is proprietary, but almost ubiquitously present on PCs and Macs. I'm not aware of viable alternatives, although Open Office may be an

option. Using an Excel workbook for this has many advantages: It is multi-page and, therefore, suitable to store sets of parameters of different type and structure in an ordered and accessible way. Excel is present on virtually every desktop worldwide and almost every world citizen knows how it works. Excel is also directly accessible from Matlab. Moreover, it is convenient because Excel by itself is a computational environment in which one can computate, copy and document the parameters it stores.

Further, the *mf++* parameter labels stored in the Excel workbook carry their explanation in the form of regular Excel comments. These pop up when hovering over them with the mouse. This way, this parameter workbook also serves as a manual by making sure the user has the relevant parameter information always at hand.

mfLab reads out this workbook, but only fetches the parameters necessary to generate the input for the particular *mf++* target models. Because most parameters will not change between projects, one seldom needs to worry about parameters settings between runs.

A model in *mfLab* normally consists of 3 local files. Two of them are Matlab *m-files* that contain Matlab code, and the third one is the mentioned Excel workbook. The two local m-files have fixed names: *mf_ adapt.m (or, alternatively, mf_ build.m)* and *mf_ analyze.m*, while the Excel workbook has a local name of your choice indicated further as *<<basename>>*. Hence it's called *<<basename>>.xls*.

The *<<basename>>* is set as one of the the first lines in *mf_ adapt.nl*.

All *mf++* input files generated by *mfLab* are given the same *<<basename>>* but get a different extension, which indicates their type (for example .dis, .bas, .wel, .drn, .riv, .chd, .lpf, .hds, .bgt etc). These extensions can be freely chosen, but it is wise to use some standard as indicated between the previous parentheses.

All *mf++* input and output files may be removed from the directory, only keeping the mentioned three to reproduce the simulation entirely at any point in the future. This not only is transparent, it also saves tremendous redundancy and gigabytes of disk space, and therefore, facilitates quality control tremendously.

The model itself is constructed within the Matlab script *mf_ adapt.m*. You make this script yourself as it is specific to the model. However, different models will likely have similar scripts. Therefore, it is always a good idea to copy an example from which to start.

The script *mf_ adapt.m* may be just a few lines for a simple model. It may also be complex, accessing different external databases and GISes, and may carry out a lot of pre-processing etc. It can run external programs like statistical codes to prepare the required input. However, Matlab scripts like *mf_ adapt.m* are built the Matlab way, i.e. they are developed, tested and documented interactively line by line, expression by expression, so that at no point the process becomes overwhelming.

Matlab's internal debugger is an important additonal tool when stumbling over errors and unexpected results. It allows setting breakpoints, verifying and testing the status of the workspace at any point during the execution.

The *mfLab*/Matlab environment facilitates another debugging method, which can hardly be overestimated, especially when dealing with real-world models of great complexity, which tend to be inherently difficult to understand and verify; in *mfLab*/Matlab it is straightforward to simplify a model constructed in the *mf_ adapt.m* script to the extent that it becomes amenable for verification, for instance by comparing its results with analytical solutions. This can be done by adding a few lines of code to the end to *mf_ adapt.m* scipt, without touching the just constructed model. Once the simplified model is verified, these lines can be removed one by one, thus retracting step by step to the original complex model, without ever touching it. Such lines may for instance replace a complex conductivity or recharge distribution by a uniform one, or it may switch off all wells but one etc, anything necessary to allow verifying the model.

Also, if constructed correctly, the model can be run with a different computational grid. This facilitates rapid development and may drastically reduce computation times, postponing runs of the full grid to the production phase, when the model has been completely verified. *mfLab* facilitates building your model directly from databases with no presumptions with respect to the computational grid.

While *mf_ adapt.m* is used to construct the model, it is analyzed using the script *mf_ analyze.m*. *mf_ analyze.m* is also a local Matlab script specific to the model. It reads out the results (head, draw-down, concentrations etc), extracts specific portions of it as required, interprets it (for instance by computing a zone-budget) and visualizes the results. *mfLab* has the functions to read out the unformatted files produced by *mf++* models. It can even extract portions of it precisely, so that it is not necessary to load a complete 2 GB output file and, as a consequence, exhaust your computer's memory. Precise extraction is also much faster than loading a complete file. The powerful visualization functions of Matlab can be readily used in *mf_ analyze.m* or are

embedded in provided *mfLab* functions to reduce complexity for the groundwater modeler and better target his/her requirements. As is the case with *mf_ adapt.m*, the best way is to start with the *mf_ analyze.m* script of a similar model, which will be 90% or so reusable. The examples that come with *mfLab* are a good start.

The Excel workbook containing the simulation parameters has a number of worksheets dedicated to specific types of parameters. There is a worksheet called NAM, MFLOW, MT3D, SEAWAT, PER, LAY and some more. These worksheets are described elsewhere in this manual. This Excel workbook has the name *<<basename>>.xls* and is local. The contents of <<basename>>.xls tends to change hardly between models. Therefore, starting with an existing one is the best way to start a new model. The workbook 'parameters.xls' under mflab/examples is an example worksheet that is up to date and can always be used as a start.

The Excel workbook will generally contain many more parameters than is required for a particular model. It may for instance contain the parameters for MT3D and SEAWAT, while one only wants to run MODFLOW. This does not matter for *mfLab* as it only extracts the parameters it needs for the target model. The advantage to have all possible parameters for all possible models and their packages in the workbook is, that these packages and models can be readily switched "on" or "off" without changing the workbook. This keeps the workbook general. A user can add whatever is desired to the workbook: *mfLab* only extracts what it needs. This design facilitates documentation and experimentation. One way to experiment is to, for instance, copy an entire worksheet like MFLOW to for instance a worksheet named MFLOW(2) and change parameters in the MFLOW worksheet as desired. MFLOW(2) keeps the old MODFLOW parameters, while in the new run *mfLab* only extracts the parameters from the old worksheet MFLOW. You can hide worksheets, and their columns and lines to prevent distracting clutter from sight without deleting anything from the workbook. Hence, using a workbook as a multi-page parameter container is a flexible and helpful approach.

Then, finally how does *mfLab* work and how to make it work?

You can use all Matlab's and *mfLab*'s functions (see *mfLab/mfiles*) to further rework, show or analyze the model results. By adding your lines of Matlab code to the *mf_ analyze.m* makes sure your analysis is automatically executed in every future run.

Illustrative example

2 Introduction

In this chapter we start right away with an example to get the feeling of how mfLab works. Details and references can be found in following chapters. For this example we assume the user is at least to some exent familar with Matlab, Excel and Modeflow. More background info on Matlab works can be obtained from the gui that is available on the internet:
http://www.mathworks.nl/help/pdf_doc/matlab/getstart.pdf
which is the Matlab primer for version 1012.
Information on usage of Modflow (2000) can be obtained from
http://water.usgs.gov/nrp/gwsoftware/modflow2000/ofr00-92.pdf
We will use modflow 2000 for now. From a user's point of view there are virtually no differnces between versions 2000 and 2005.

The example is described in the Modlfow 2000 manual for which the reference was just given. It referst to the figure in the cover page of that manual and was carried along as the first example from previous version.

The actual example is is shown in figure 3. It concists of 3 model layers (aquifers) separted by 2 confining beds (aquitards). The bottom two ones are semi-confined and the top layer has a free water table (phreatic). Therefore, the bottom two layers are charactarized by their transmissvity and elastic storage coefficient, while the top layer is characterized by hits hydraulic conductivity, the depth below the water table and its specific yield. The two confining beds are characterized by their vertical conductance, i.e. their vertical conductivity over their thickness. Storage within the confining beds is deemed negligible. As boundary conditions we have recharge from precipitation a presribed head in the west end of the two uppermost layers, while all other boundaries are closed, which is automatically the case when nothing about them is specified. There is a drain reaching from the fixed-head boundary in the west to about 2/3 of the width of the model in east direction

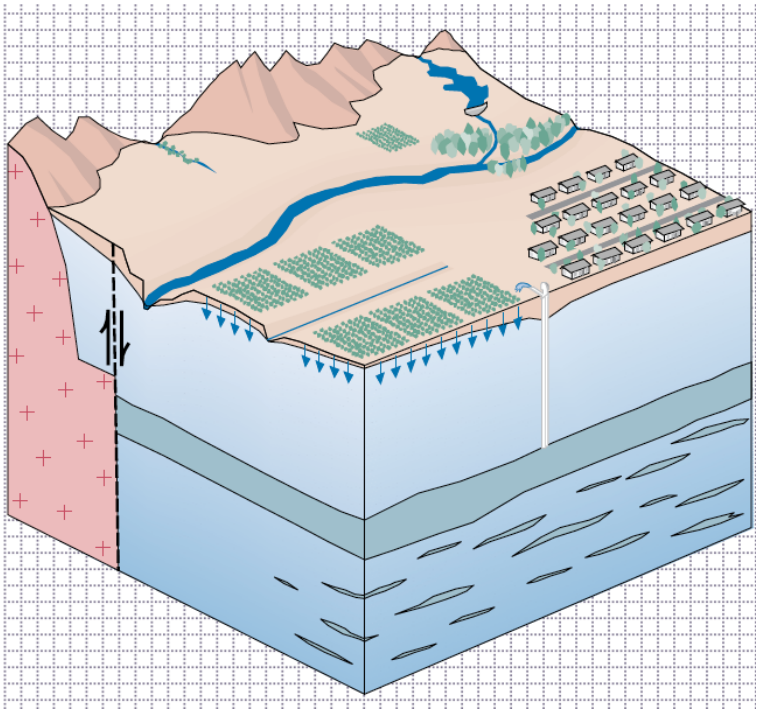


Figure 2: Model on cover of Modflow 2000 manual

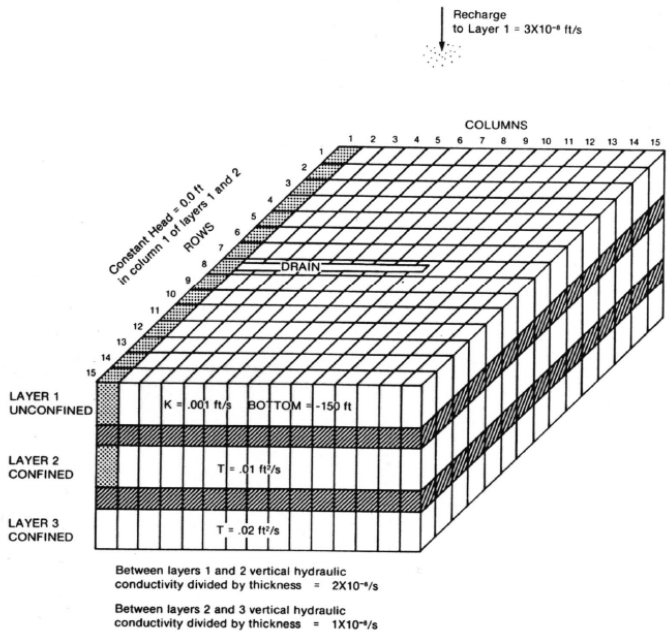


Figure 3: Example ex1 from Modlfow 2000 manual

as shown in the figure. Next to this, there are a number of wells that extract from the top and or form the bottom aquifers. Most of the actual values are given in the figure.

In MODFLOW such a model can be defined in two ways. First as quai-3D, by using the transmissivities of the semi-confined aquifers without reference to the actual elevations of the layers, and by horizontal and vertical conductivities, with honoring of the elevation of the bottom and top of each layer. The latter is the more modern way, as for particle tracking and mass/heat transport we need these elevations anyway. Hence we skip the approach which uses transmissivities and focus on the one that is based on conductivities and actual layer elevations. The first appoach uses MODFLOW's BCF package, the latter uses its LPF package.

What data are necessary for this model?

The model needs a number of 3D arrays holding parameters for the model cells and some arrays that hold the information on the boundary conditions. Further, initial heads are always required. They are needed to start the iterative solver, they are the essential initial condition in the case of transient flow, and provide the fixed value for the model cellls whose head needs to be kept fixed.

mfLab uses the exact parameter names that MODFLOW uses. Therefore, one is referred to the MODLFOW manual to check which parameters are required and what their meaning exactly is.

For this example we need the following 3D arrays (with one value per cell in the layers and those in the confining beds:

HK Horizontal conductivity for the phreatic and the two semi-confined layers

VKA Vertical conductiviy or vertical anisotropy of all layers

VKCB Vertical conductivity of the confining beds

SY Specific yield of the top layer

SS Specific elastic stroage of all layers.

STRTHD Initial heads of all layer

IBOUND Array telling in which cells the head will be kept fixed and in which the head will be computed. IBOUND also defines which parts are infactive (if any)

When mfLab is launched, it expects these variables to be present in Matlab's workspace. They must also be of the right size as required by the grid of the finite difference network, to be shown duely. It is up to the user to provide thise arrays. The user does so by runing a Matlab script which always has the name mf_adapt.m (alternative name mf_build.m). This file contains Matlab instructions that will generate the required arrays. Lauching mfLab is actually done by running the script mf_setup.m. This is the backbone of mfLab. It executes mf_adapt.m, so that the arrays will be generated and then proceeds writing the intput files for MODFLOW. When finished MODFLOW will be run automatically.

So the user has to make sure that the arrays are in the workspace when mf_adapt has finished. There are thousands of ways a user may wish to generate the required arrays. One could invoke a GIS, or any other model to do part of the work, or read data from databases and rework them into proper arrays for MODFLOW or one could generate the arrays with same simple Matlab commands. An enormous wealth of possibilitis exist, small, complex and intermediate. However, in this simple example, simple Matlab commands will suffice. This is the way we will approach this problem.

It is clear that all arrays should match the size of the finite differnce grid to be used. Matlab allows to easily deal with any grid, but at the end of the day some grid has to be chosen. Because one can readily rerun mf_adapt or mf_setup for that matter, it is easy to change the grid later on if, for instance, more detail is required.

A grid requires a ste of coordinates for the horizontal, and vertical gridlines. We will use xGr, yGr and zGr to define the coordinates of te grid lines. Onese these have been set, those of the cell centers as any other grid-depdendent variable can be computed and needs not to be stored. The zGr would expres constant layer elevations in all cells. This is not required in MODFLOW. Therefore, Z may be used instead of zGr, where Z has an elavation value for the top and the bottom of all layers.

The easiest and most general way to deal with a grid is using the grid object constructor:

```
gr = gridObj(xGr,yGr,zGr,LAYCBD,MINDZ);
```

MINDZ is optional. It defines the minimum layer thickness. It is normally set to 1 mm or 1 cm.

LAYCBD defines the confining bed positions. It can be left out if there area no confining beds. But in our case there are. LAYCBD is a vector with one value per aquifer layer, where zero means that this aquifer has no confining bed connnected to its bottom, and where a value different form zero means that it has. In this case

```
LAYCBD = [1 1 0]; % or [1 1]
```

The coordinate values like xGr can be given in any order. The grid object will take care that they weill be sorted and that duplicates are removed (to 1 mm accuracy). The same is true or the yGr and zGr. While xGr will alway run from left to right (upward), yGr and zGr always run downward. Hence the first row in the yGr array or vector has also the highers coordinate. The same is true for zGr; the z axis is always increasing upwards. The top layer has the highest z-coordinate and the bottom layer the lowest. In the case the z-columns of the finite difference grid differ, one provides the full 3D array of zGr coordinates to the grid object. This array has a z-value for the bottom and the op of all cells in the finite difference mesh. When the size of the mesh is Nx columsn, Ny rows and Nz layers, the the size of xGr is $Nx + 1$, that of yGr is $Ny + 1$ and that of zGr $Nlay + Ncbd + 1$ and that of Z is at least $[Ny + 1, Nx + 1, Nlay + Ncbd + 1]$. At least refers to the possible presence of confining beds. Each confinng bed adds an extra layer. $Nlay$ is the number of model layers (aquitards) and $Ncbd$ the number of confining beds. Note that the total number of layers $Nz = Nlay + Ncbd$. All these details are implied by the $LAYCBD$ vector that has been supplied to the grid object constructor, so the user only has to supply the $LAYCBD$ information and to make sur the correct coordinates are given. The user should not worry about the inernal details of how to deal with a complex set of layers and confining beds.

To generate the grid for this example model in *mfLlab*, inlcude these lines:

```
xGr    = [0:5000:75000]
yGr    = [0:5000:7500];
zGr    = [200 150 100 0];
zGr=[200 -150 -200 -300 -350 -450]; % Plane elevation vectorin feet
kh =[1e-3; 1e-4; 2e-4];
vkcb=[1e-6; 5e-7];
LAYCBD = [1 1];
gr     = gridObj(xGr,yGr,zGr,LAYCBD);
```

The grid can now be inspected by typing its name:

```
gr
```

followed by a return.

The grid has numerous methods to provide services based on it. These can be viewed by pressing on the blue workd Methods that shows after the gr was typed as a command.

One of the sevices that it can provide is generate arrays of the right size with constant values or with a constnat value for ech layer. This is the easiest way to geneate the requreted array, when their parameter values are constant within the layers.

```
HK     = gr.const([kh1; kh2; kh3]);
VK     = gr.const(kv);  % you may also use VKA
VKCB   = gr.const([vkcb1; vkcb2]);
SY     = gr.const(sy);
SS     = gr.const([ss1; ss2; ss3]);
STRTHD = gr.const([200; 100; 50]);
IBOUND = gr.const(1);

IBOUND(:,1,[1 2])=-1; % top 2 aquifers at left of model have constant head
```

Recharge must be provided for each and every stress period. In case one value per stress period suffice, i.e. when the recharge is the same everywhere in the model, this is easiest done in the accompanying workbook ex1.xls. The worksheet PER therein, has a a row for every stress period, for as far values between stress

periods differ. If all values are the same for successive stress periods, these intermediate stress periods need not be specified. Hence if the simulation has 1000 stress periods with the same values in everyone of them, only one line suffices in the worksheet PER. The number 1000 in the column iper the denots the number of stress periods in the simulation.

The workbook has a sheet called NAM (names). For the recharge to be invoked, the RCH package must be switched on. This is done by inserting the value 1 to the right of the cell with the nam RCH.

The PER worksheets collect all variables and parameters that are time-dependent. As values are constant during one stress period, putting them togther guaranties that all are synchronized. This worksheet cotains, among others, collumns INRECH and RECH. RECH holds the recharge value for the stress periiods, a single value for the entire model in this stress period. And INRECH>0 tells mfLab whether that the recharge is unifrom and to be obtained form column RECH. If INRECH==0 then mfLab expects the variable RECH to be given in Matlab's workspace as a 3D array, in which the 3rd dimension is the number of stress period. Hence one sheet of size (Ny,Nx) values for every stress period.

The following bounadary condition is the drain. For drains to work, the DRN package must be switched on in the NAM worksheet. Type a one next to the cell with the name DRN.

The drain has to be specified by means of its cell coordinates at LRC (layer, row, column) followed by the elevation of the drain in these cells and by its hydraulic conductance$C\left[\mathrm{L}^2/\mathrm{T}\right]$, to that the flow is

$Q_{LRC} = C_{LRC}\left(\phi_{LRC} - h_{dr,LRC}\right)$

This can be done by hand in mfLab, by specifying the LRC, $h_{dr,\,LRC}$ and C_{LRC}, fo rall cells in the workspace

```
DRN = [
];
```

Or it can be one automatically using the grid and its mothod gridObj/bcnLine.

```
DRN = gr.bcnLine(basename,'DRN',IBOUND,zoneVals)
```

basename is always the name of the problem at hand and matches the basename of the xls file. This allow finding the PER sheet in the XLSfile where transient data are stored. The label 'DRN' refers to the package used. It makes sure that subtile differences between packages are correctly dealt with. IBOUND is used as a ZONE array. To make this work, we must put an arbitrary but known zone number in the cells that correspond with the drain. In this case we might say

```
iDrn = 12 % the drain get zone number 12 (arbitrary but unique)
```

and than store this zone number in the IBOUND array:

```
IBOUND(...) = iDrn;
```

The last parameter is zoneVals. It defines what values will actually be stored on the drain cells. There is one line per zone, starting with the zone number. Subsequent values correspond with the values that the boundary in question requires. In the case of the drain this is a drain elevation and a drain conductance pertaining to each cell. These value may differ between cells as well as betwen stress periods, i.e. fluctuate in time. The values can be scalars, vectors or strings. In the case of scalars, all cells of the zone get the same value and als get this value in all stress periods. In the case of a vector, which is a long as there are cells in the zone in question, all values may be different, but these values are constant in time. In the case of strings, the strings are interpreted as header in the PERworksheet, from where a value for each stress period can be grabbed. In this case all cells of the zone in question get the same value in space but a different value in time. The possibility of all cells of the zone getting different values in both space and time, has not been implemented. With this, we can specify our drain

```
zoneVals= {iDrn,  ,,}
DRN = gr.bcnLine( )
```

Checking by running mf_adapt

When ok, run mf_setup.

with ϕ the head in the aquifer cell and h_{dr} the elevation of the drain in that cell.

. in which one or more vlau

How

to get *mfLab* to work?

3 How to get your *mfLab* copy?

A complete working copy of *mfLab* can be obtained using by a checkout at http://code.google.com/p/mflab by means of Subversion (svn) version control software (see http://subversion.tigris.org) used by Google to service its open software development and distribution site http://code.google.com. To checkout, you must have *svn* installed on your computer . *Svn* comes pre-installed on Macs and most UNIX systems, but Windows users must download it from the tigris.org site just given.

Subversion is a great tool for general version control of anything you work on, like projects, software and reports, dissertations and so on. Therefore, there is little reason to hesitate installing it if you do not yet have it. The fact that Google selected it for its code.google.com server to provide users and developers easy version and release control says something. Windows users may want to download the beautiful Tortoise Subversion user interface from the same tigris.org site. From that site, you may also want to download a copy of the free subversion book, which can also be obtained in printed form O'Reilly publishers. The book will be most useful for Mac and Unix users who generally type subversion commands in their terminal application. Tortoise users on Windows will enjoy its great user interface, which can be more intuitive. For an introduction, refer to the tutorial in the mentioned book. It's worthwhile to get familiar with svn.

To see how to check out (svn jargon for downloading your working copy) look under the source tab on http://code.google.com/p/mflab for the checkout command. From the *terminal* application on the Mac or Unix or from the *dos prompt* in Windows make sure you first navigate to the directory where you want your working copy to land, and then type the checkout command. In Tortoise look at the instructions. This should be straightforward and probably more convenient for most users than using terminal or the dos command window.

Except for this user guide, there are no download packages on the site. This user guide in only to get you going. Checking out through *svn* to get a *mfLab* working copy is really much better than downloading a copy in a zip for tar file as *svn* frees you of having to bother about future updates. Just typing *svn update* from any directory in your working copy next time tells *svn* to download only what has changed since your last checkout to update your working copy. With Tortoise these updates will be even easier and perhaps more transparent.

4 The *mfLab* directory structure

The svn checkout will download a working copy of *mfLab* onto your computer. The directory structure will be as follows:

mfLab

- ***doc***
- ***bin***
- ***mfiles***
  - analytical
  - *read*
  - *write*
  - *gridcoords*
  - *etc*
  - *fdm*
  - *visualization*
- ***examples***
  - *NHI*
  - *mf2k*
  - *mf2005*
  - *mt3dms*

swt_ v4
swi
- extra directories may be added in the future

doc contains this user guide. It also contains additional information, such as how to compile the source files of *mf++* models on the Mac.

bin contains copies of the executables. It is a convenient location to store them. If you dislike copying executables to this location, you could replace the with links (aliases on the Mac or shortcuts on Windows) to them.

mfiles contains several directories to categorize more or less the actual Matlab functions that comprise *mfLab*.

mfiles/read contains the *mfLab* functions (Matlab mfiles) to read *mf++* input files back into in the Matlab workspace. This allows reading any original MODFLOW, MT3DMS, MOCDENSE or SEAWAT model obtained from whatever source or colleague. These files will be extended in the future when useful.

mfiles/write contains the *mfLab* functions to write (generate) the mfiles for *mf++* programs. It also contains the *mf_ run.m* script, which is the backbone of *mfLab*. These files will be extended with any new package added to *mfLab*.

mfiles/grid contains *mfLab* functions that relate to the computational grid and to handling coordinates.

mfiles/etc contains *mfLab* functions that do not logically fit under the other directories

mfiles/fdm con stains some finite difference models written entirely in Matlab. These are used in lectures in groundwater modeling at the TUDelft. They can also be useful to check MODFLOW output.

examples/mf2k contains examples pertaining to MODFLOW 2000 (mf2k)

examples/mf2005 contains examples pertaining to MODFLOW 2005

examples/mt3dms contains examples pertaining to MT3DMS

examples/swt_ v4 examples pertaining to SEAWAT (version 4)

examples/swi examples pertaining to the SWI (salt water intrusion)

Other directories may be added over time. Each of these example directories contains subdirectorie with concrete examples, or contains a further division of the examples below which reside subdirectories with the concrete examples.

Each example directory contains at least three files: mf_adapt, mf_analyze, <<basename>>.xls, where <<basename>> is the name of the example that is used to name all input and model generated output files. These files differ only in their extension. These three files suffice to generate the required input files, to run the model and to analyze the results. Running is done by typing *mf_ setup* in the command window, and viewing and analyzing by typing *mf_ analyze* in the command window, after the model run invoed by *mf_ setup* has finished. Note that *mf_ setup* runs *mf_ adapt* along the way to generate the required input data for the models in question. *mf_ adapt* can be run by the user to verify that all required arrays are being generated and are available in Malab's workspace.

5 Make *mfLab* known to Matlab

Once you have *mfLab* on your computer, you have to make it known to Matlab. The simplest way is to navigate in Matlab to the directory mflab/mfiles. Then type *mfsetpath* to run the script which set the paths to the mfile directories of *mfLab* and also replaces the path to the executables in the script *mflab/mfiles/write/setExcutables.m* to the *mflab/bin* directory on your current system.

If this runs without complaints, this is all you have to do.

If it doesn't work you have to add the *mfLab*'s mfile directories to Matlab's search path yourself. This can be done using the addpath(<<path>>) function in Matlab as shown below (you may also look in the file *mfsetpath.m*).:

mfsetpath also produces instruction and command to create a shortcut in Matlab's shortcut toolbar on the left of the Matlab window just below the icons. Follow the instructions provide by mfsetpath when it is run or look into the file.

```
% Shortcut summary: Setting the mfLab paths in Matlab
pathstr = '~/GRWMODELS/mflab/mfiles/'; % change this to your path to mfLab's mfiles directo
addpath([pathstr 'write']);
addpath([pathstr 'read']);
addpath([pathstr 'etc']);
addpath([pathstr 'analytic']);
addpath([pathstr 'visualization']);
addpath([pathstr 'gridcoords']);
addpath([pathstr 'fdm']);
addpath([pathstr 'NHI']);
fprintf('... mfLab paths set.\n');
```

Instead of typing in these lines every time you invoke Matlab, it's far more convenient to store them in a so-called *shortcut* in the *shortcut toolbar* at the top of the Matlab screen (see figure). Next time you want to use *mfLab* in Matlab just push this shortcut button. The figure shows Matlab's shortcut toolbar and my shortcut *mfpaths*. To run the shortcut, just press it with the mouse. The figure also shows the opened shortcut *mfpaths*. It contains an *addpath* command (addpath call) for every mfiles directory that Matlab needs to know about. The P used in these calls is just a string holding the name of the path to the mfiles directory, like: P='Z:/GRWMODELS/mflab/'. This string is outside the visible part of the shortcut window shown in figure 5.

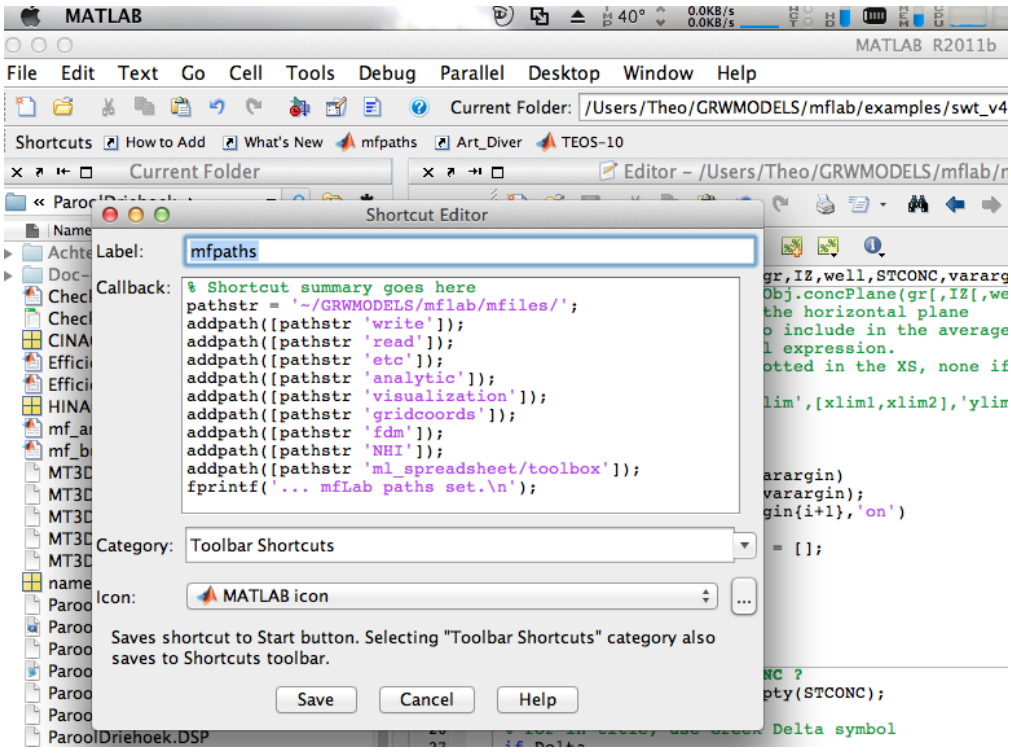


Figure 4: Make a shortcut on the shortcut toolbar to set the paths to the mfLab directories

To check whether *mfLab* finds its functions and will work type *which mf_setup* in Matlab's command

window. Matlab will then show if and if so which *mf_ setup* it finds and will execute when called.

5.1 Make the location of the executables known to *mf _ setup*

If running *mfsetpath* worked, then you should be all set. mfsetpah has, in fact, replaced the string "*path-tosetexecutables*" in the mfile setexecutables.m by the actual path *<<your path/mflab/bin>>*. Therefore the mfile setexecutables.m should from now on know where to find the exectables *mf2k.exe swt_ v4.ext* etc. You may take a look into the script *mflab/mfiles/write/setexecutables.m* to see if the path to the executables has been replaced by the path on your comptuer. If not, you may do so by hand.

If running *mfsetpath* went smoothly, the path to the executables in *mflab/bin* has already been replaced by the current one, so you should be all set and don't have to do anything at all. In case you run on both a mac and PC, you may have to run *mfsetpath* both from the mac operating system and from Windows to replace the path to the executables for both operating systems in the file *setExecutables.m*. This is because the same directory has a different path from the perspective of the two operating systems. But you don't have to bother if you run on only a single OS.

Once you checked out your working copy of *mfLab* from *http://code.google.com/p/mflab* by *svn*, exclude the file *setExecutables.m* from future downloads. Look at the "source" page for the code it says something line this:

This ensures that new versions don't override your directory settings, so that future releases are guaranteed to work immediately without change having to redo the path setting.

If you dislike copying executables to the *mflab/bin* directory, replace the executables with a link (alias, shortcut) to the ones you want to use instead.

So this would be the SVN command to get you working copy of the source of mfLab in its entirety

```
svn checkout https://mflab.googlecode.com/svn/trunk/ mflab
```

This will yield a directoy mflab in the directory where you lauched the command with all subdirectories below it.

In TortoiseSVN you fill in the URL

```
https://mflab.googlecode.com/svn/trunk/
```

and the name of the local directory, probably *mfLab*, in the other box.

This should start the downloading.

There is a manual for installing mfLab on WIndows PC's using Tortoise for on the Download page called

```
mf_windows_installation_guide
```

6 How to run *mfLab*?

Once the paths have been set, navigate ("*cd*") to your project directory and start working with *mfLab* by building your model in *mf_ adapt.m* invoking *mf_ run* or generate the input files for the target groundwater models and subsequently run the *mf_ analyze.m* script to analyze and visualize the results. But first you have to make the location of the *mf++* executables known to *mfLab*.

Having your *mf_ adapt.m*, *mf_ analyze.m and <<basename>>.xls* in a local directory, you invoke *mf_ run* from within the Matlab command window. Of course, Matlab has to be able to find this script as well as all other *mfLab* functions necessary to generate the input files for the target *mf++* programs. This is explained above under "how to make *mfLab* work?". But assuming for now that the required paths are set correctly, *mf_ run.m* will run. It is the backbone of *mfLab*. Unless you are developing new features to it or encounter a bug, you should not have to change it in any way.

mf_ run executes *mf_ adapt.m*, from which it gets the *<<basename>>* of the current model. It then builds the model arrays following the instructions in the scrip *in mf_ adap*t. *mf_ run* reads out the *<<basename>>.xls*

```
% TO 091218

% Path to the executables. I put all of them into mfLab/MODELS/bin
% for convenience, because I have several differntly compiled versions.
% It's your choice however. Anyway, set the parameters MODFLOW MT3D etc
% down below to their actual lcoations on your hard drive.
%
% If you don't like to copy your executables to another location, then copy
% a link to them into the mflab/bin directory.

% NOTICE the " " in the paths below to manage spaces in file names
% in system command used to launch the external executable later on


fprintf('Defining paths to your excecutables\n');

if ismac
    MODELS='~/GRWMODELS/mflab/bin/';  % location of my executables
    MODFLOW=[MODELS,'mf2k.mac'     ];  % location of MODFLOW executable
    MT3D   =[MODELS,'mt3dms5s.mac'];  % MT3DMS executable
    SEAWAT =[MODELS,'swt_v4.mac'   ];  % SEAWAT Executable
    SWI    =[MODELS,'mf2kswi.mac' ];              % mf2k which knows SWI
elseif ispc
    % dos('net use W: "\\vwmware-host\Shared Folders\tolsthoorn On My Mac"');

    % use the mapped drive. This shortens the file names a lot in my case
    MODELS='Z:\tolsthoorn On My Mac\GRWMODELS\mflab\bin\'; % location of my executables
    MODFLOW=[MODELS,'mf2k.exe'     ];  % location of MODFLOW executable
    MT3D   =[MODELS,'mt3dms5b.exe'];  % MT3DMS executable (use binary version on windows
          % so that it is comopatible with the stanard windows mf2k exacutable)
    SEAWAT =[MODELS,'swt_v4.exe'   ];  % SEAWAT Executable
    SWI    =[MODELS,'mf2kswi.exe' ];  % mf2k which knows SWI
else
    help computer
    error(['You''re not running on either a Mac or PC, as yet only mac and pc are supported\n',...
           'Nevertheless, unix is expected to run on mac without change.\n',...
           'Try changin ismac to isunix in setup, rarray and m files readding unformatted\n',...
           'model output (i.e. readDat, readBud, readMT3D).\n',...
           'Type help computer for more information.']);
end
```

Figure 5: Contents of the script *setExecutables.m* called from *mf_run.m* line +/- 142

Figure 6: Look at the Source to see the URL and SVN instruction to get your chekcout

Use this command to anonymously check out the latest project source code:

```
# Non-members may check out a read-only working copy anonymously over HTTP.
svn checkout http://mflab.googlecode.com/svn/trunk/ mflab-read-only
```

Figure 7: SVN instruction found on the Source page of the site

file to retrieve the required parameters, including stress periods and layer characteristics. It then starts generating the input files for the target models. It knows the target models and packages from the NAM worksheet in the *<<basename>>.xls* workbook.

If everything runs fine, you end up with the local directory full of input files for the target models. All these files have the same basename. Their different extensions indicate their function as defined in the NAM worksheet.

Next to that, *mfLab* generates so-called name files, that is, files the a *".nam"* extension. These name files list the packages and files to be used by the target executables. The file *mf2k.nam* will always be among them. It is the name file required by mf2k *and mf*2005. If, on the other hand, MT3DMS is to be run, then the name file *mf3dms5.nam* is also generated. In case SEAWAT is to be run, you will find the name file *swt_v4.nam* as well.

The reason for generating several name files is, that the input files required by SEAWAT also suffice to run MT3DMS and MODFLOW. So, having the set of name files available facilitates debugging. For instance, if SEAWAT stumbles, you can still try to run MT3DMS with its own name file. If that doesn't work, then something may be wrong with the MODFLOW files, so try to run MODFLOW using its own name file. No additional files are required because all the necessary files and the name files have already been generated by *mf_run*.

mf_run also generates three batch files (files with a "*.bat*" extension): *mf2k.bat*, *mt3dms5b.bat* and *swt_v4.bat* which will invoke the executables by simply double clicking on them (under Windows).

To see how to run individual *mf++* target models, refer to the last lines of the *mf_run* script. The function *system*(...) at the end of this script executes a command in the underlying operating system directly from Matlab. So select the desired one and press shift F7 on the Mac or press F9 on Windows to execute it directly from the Matlab editor.

mf_run tries to figure out which target models are to be run. It does this by looking at the packages that are "on" in the NAM worksheet. If the SEAWAT-specific VDF package is "on", *mfLab* assumes you want to run SEAWAT. If this is not the case, but the MT3DMS-required package BTN is "on" in the NAM worksheet, *mfLab* assumes you want to run MT3DMS. In case the SWI (salt water intrusion) package is "on" in the NAM sheet, *mfLab* presumes you desire to run MODFLOW with the SWI package "on". If non of these are "on", *mfLab* will just launch MOFLOW2000 (mf2k).

As said before, you can always run a specific *mf++* model by running the concerned system(...) expression at the end of *mf_run.m*.

Finally, once the specific *mf++* model has come to *normal termination*, launch *mf_analyze* to extract, analyze and visualize the results.

7 Debugging

A model is best set up iteratively while use is made of error messages to figure out which data are still necessary. Generally, only *mf_adapt* needs to be changed. *mf_run* should not be touched unless you need to resolve an error in the code or you are adding new features to *mfLab*.

To optimize visualization you may have to adapt the script *mf_analyze.m*. This too is best done iteratively.

When errors are encountered (and even if only warnings are thrown by Matlab), it helps to switch on the debugger from within the toolbar of the Matlab editor. In the menu set "always stop if errors". You may also set "always stop if warnings". If Matlab stumbles over an error it will immediately stop and show you where the problem arose. You can then solve it or, better, first investigate the problem by inspecting the parameters in the environment where the error occurred. You can walk down the stack along which the error happened to trace it source. In Matlab you may explore any expression to find out what the problem is before resolving it and starting anew. As long as you see the K>> prompt you are in debugging mode, which may imply you are somewhere deep down in the calling stack inside the environment and scope of a local function. Type *dbquit* to get out and start again.

As said before, if everything runs fine including the visualization, but you don't understand the outcomes, you have a conceptual problem to resolve and need to put your model on the workbench. An effective way may be to add a few lines at the end of *mf_adapt* to simplify your model so that it becomes comprehensible and amenable to for instance comparison with analytical solutions. For instance, you could set all conductivities to a fixed value k wiht the following Matlab instruction:

K(:,:,:)=k

thus simplifying your complex model to a simple rectangular box. This can be done with other parameters and boundary conditions alike:

STRTHD(:,:,:)=0;

It is always advised to inspect the global and list files *<<basename>>.glo* and *<<basename>>.lst* to see how the model performed and whether or not it converged and the water balance was closed sufficiently. These files are crucial in case the executable crashes somewhere during the run. This then happens completely outside Matlab and the mentioned files may be your only resource to start your search for the cause. Finding the cause may be hard at times. A good approach is also to carefully inspect the input files produced by *mfLab* to make sure they are correct.

Some errors that have occurred (but may by now be tackled by *mfLab* issuing a comprehensible error message regarding the cause):

Same unit number for different files. (is now checked by mfLab and an error is issued before running MODFLOW)

Running MT3DMS with the specified output time vector TIMPRS starting at zero makes MT3DMS hang without warning. This is now tackled by *mfLab* by removing this zero from the vector if it occurs.

Replacing an input file with one from the examples downloaded form the USGS but having a different unit number (make sure that unit numbers match).

Running a model of tiny dimensions so that some of the floating point numbers became zero due to the applied FORTRAN format like F12.2 (tackled but this might happen at other locations).

Assigning a full 3D matrix to the DELC or DELR vectors in *mf_ adapt*, so that MODFLOW tried to construct a really huge model that made the computer hang (tackled).

Matlab's g (general) number format sometimes does not fit in the 10 space wide fields required by the fixed format input files of WEL, DRN, RIV, GHB and CHD. The g format is great as it always guarantees the requested accuracy in the most efficient way. However, the Matlab's implementation may just take 12 spaces if it needs them, even if it was ordered to use only 10. This may happen at arbitrary locations in the generated input files. I have tried to capture this in a somewhat sophisticated way in the writing routine, but this is actually something that Matlab should have solved a long time ago. I consider it a nasty Matlab bug. I'm not certain at this point that it can never ever happen again, but I did my best to prevent it.

8 Reading out the Excel workbook on different platforms

The Excel workbook *<<basename>>.xls* containing the model parameters is read out by *mfLab* using Matlab's function *xlsread*. *xlsread* will only run trouble-free on Windows systems that have Excel installed (actuall have the Excel com server). On systems without Excel and on non-Windows computer systems, i.e. systems without the Excel com server, *xlsread* has to be run in so-called "basic" mode, which provides less capabilities to read out workbooks. However, this is not a big deal. First, *mfLab* only needs to read entire worksheets of a workbook, which is possible in basic mode. Second, *mfLab* automatically runs *xlsread* in *"basic"* mode on non-Windows systems, so that users do not have to worry about it. However, original Excel workbooks read by *xlsread* in *"basic"* mode may still cause trouble, as the error message shown in the figure below shows.

```
Preparing name file struct.
Basename current model is 'ex1'
Getting nam file data from ex1.xls
Skipping 16 bytes of extended strings.
Skipping 16 bytes of extended strings.
Skipping 16 bytes of extended strings.
Skipping 16 bytes of extended strings.
Skipping 16 bytes of extended strings.
Skipping 16 bytes of extended strings.
??? Error using ==> xlsread at 207
Specified worksheet was not found.

Error in ==> mf_setup at 121
    [Nnum,Ntxt]=xlsread(XLSF,'NAM','','basic');
```

Figure 8: Error reading xls workbook on non windows system due to incompatible Excel file

This is due to the fact that the *xlsread* "basic" mode has never been updated by the Mathworks in Matlab since Excel version 05/95. In "basic" mode, *xlsread* cannot read more than 3 worksheets from workbooks generated by recent Excel versions. The solution (or workaround) on non-Windows systems and on Windows systems without the Excel com server installed, the Excel workbook has to be saved as an Excel 05/95 file. I have done this with the Excel workbooks of all examples provided with *mfLab* to ensure that they will work also on non-Windows computers without any changes. Saving Excel files in this format has no consequences, except that Excel asks if you are sure you want to save in that old format. Just press "yes" if you intend to run on non-Windows systems or on systems without the Excel com server. Else, say yes or just save in any of the more recent format.

One of the things that do not work with the Excel 05/95 is conditional formatting. This was used in the NAM worksheet to to automatically let lines turn green for those packages that are "on". As an alternative I have now replaced this feature with a left arrow like "<====" popping on to the right of each line with its package "on". A little less nice but it works almost as well. Alternatively, you could use filter, to make only the "on" lines visible.

Since *xlsread* is probably one of the most used functions of Matlab, the Mathworks should have updated it a long time ago. Hopefully they will finally do so with the next release, the importance of the function and the price of Matlab is high enough to warrant this.

8.1 Using xlsread

In Matlab, *xlsread* is called to read Excel worksheets. The behavior on Windows and non-Windows computers is a bit different. Windows computers with MS-office installed also have the Excel Com Server available. On these computers, requests for data in Excel workbooks are interpreted by the Excel Com Server. This is an advanced beast that allows very precise transfer of Data ro and from Excel workbooks of almost any previous and recent version. However, on computers without this severs, i.e. Windows computers without MS-office and Macs, data has to be read out directly from the binary Excel file. This is done by using the "basic" argument and it only works if the worksheet is saved in the excel5.0/win95 format to be found in the saveas window. MfLab generally uses the basic form, but this is only obiligatory on computers without the Com Server installed. So on Mac's make sure the workbooks are saved in *excel 5.0/win95* format.

There is a number of differences in the result of reading an excel workbook under Windows and on the Mac, but mfLab has hitherto been able to cope with them. To make sure that the behavior is the same on different computer platforms, line up the Excel worksheet data so that the range containing non-empty cells matches with Excel worksheet cell "A1". That is prevent empty rows and columns above and to te left of your data range. With this in mind xlsread is an extremly powerful function. Hopefully the Mathworks will one day see how useful it is and make sure that its behavior is exactly the same on all platforms.

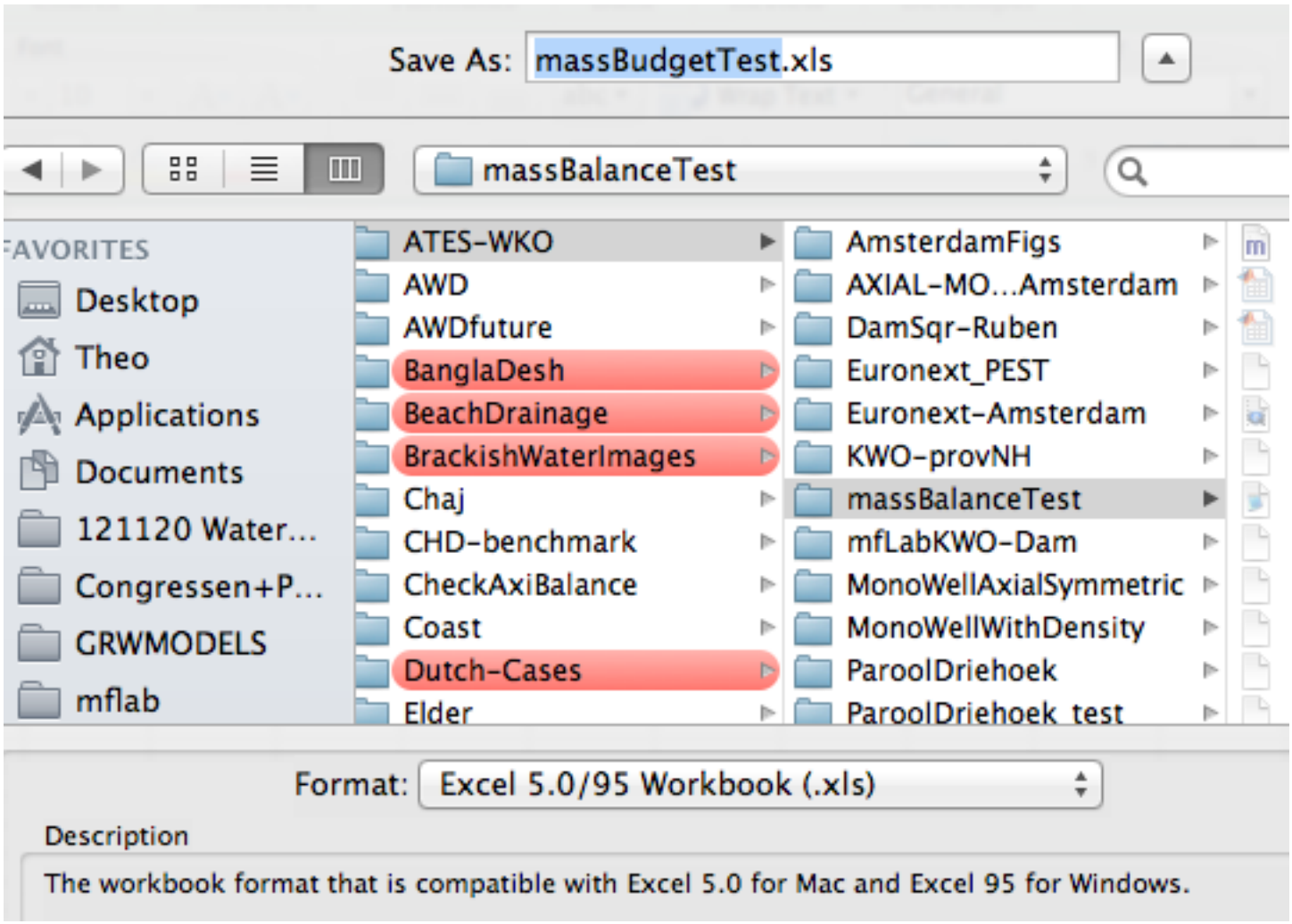
The xlsread function is fully described in the Matlab help system. Type doc xslread to get this help.

8.2 [NUMhdr,NUM,TXThdr,TXT] = getExcelData(basename,sheetname,direction)

The *mfLab* function getExcelData reads data from the excelworkbook *<<basename.xls>>* or *<<basename.xlsx>>* where direction is either 'Horizontal' or 'Vertical', denoting the direction in which of the labels of the columns and rows respectively are ordered. The data below horizontally arranged colmun labels start at row 3, the labels are in row 2 and row 1 is free to be used by the user but must contain labels. Generally this row is used to name the package that uses this variable named by the label in the second row. Likewise, with argument 3 equal to 'Vert', the data are arranged in rows and start in the 3rd column. The second column contains the labels and the first is free to use by the user, but is generally used to denote the package that uses the variable in the second column. For instance the spreadsheets MFLOW, MT3D and SEAWAT have labels ordered vertically, while the spreadsheets PER and LAY have their labels ordered horizontally. You can add any data and read it out by getExcelData as long as it obeys the rules just outlined.

The results are split in two groups: 1) numerical data, 2) text data. NUMhdr is a cell array with the labels the columns or rows containing the numerical data, and NUM is the array containing the actual data. If the 3rd argument was 'Hor', then these data are in columns. Likewise, TXThdr is the cell array with the labels of the text data, while TXT is a cell array with the actual data (the strings). If the 3rd argument was 'Hor' these text data are in columns, otherwise they are in rows.

Figure 9: Excel file save menu ont Mac, saving as Excel 5.0/95 Wokbook (.xls), see format at lower end of figure.



Reading out any values is then straightforward using mf_lab function strmatchi, which looks for a string (hence a label) in a cell array of strings, i.e. the cell array with the column or row labels. Reading out column PERLEN, would be

```
PERLEN = NUM(:,strmatchi('PERLEN,NUMhdr));
```

Reading out column 'NAME' from text data would look like this:

```
NAM = TXT(:,strmatchi('NAME',TXThdr));
```

This way, any data can be retrieved from an Excel table that obeys the described format. The function getExcelData is often used by *mf_lab* and in user files *mf_adapt* and *mf_analyze.m* as well.

9 Reading unformatted output from MODFLOW, MT3DMS and SEAWAT on non-Windows computers

MODFLOW, MT3DMS and SEAWAT all produce binary output files that need to be read for itnerpretation in *mf_analyze.m*. mfLab has the functions *readDat*, *readMT3D* and *readBud* for that purpose. *readDat* is used to read in the binary head and drawdown files. *readMT3D* is used to read in the files with dissolved and sorbed concentration produced by MT3DMS and SEAWAT and *readBud* is used to read in the binary budget file produced by MODFLOW. It can also be used to read in the interface elevation file, the '.ZTA' file produced by the SWI package (Salt Water Intrusion). Although different FORTRAN compilers produce different binaray files, *mfLab* does its most to deal with that internally so that users do not have to worry about it.

All three functions, readDat, readMT3D and readBud produce a struct array with one element per output time. Each element is a struct that has fields showing additional information pertaining to the element, such as the stress period and time step number, and the number of rows, columns and layers. readDat has field totim with the total time since the start of the simulation, while readMT3D has time instead. Both have the field 'values containing a 3D array with the actual data, one value per model cell for that particular time. So to obtain the heads

```
H = readData([basename  '.HDS']);
```

The number of time elements in heads is

```
NT = numel(H);
```

The time of simulation for the fifth element is

```
time = H(5).totim;
```

And the actual 3D data array is

```
hds = H(5).values;
```

And the head in layer two for that time step

```
hd=H(5).values(:,:,2);
```

which may be contoured.

The files mf_analyze.m in most examples demonstrate the use.

The function readMT3D is similar to readDat, but is tuned to the perticularities of the concentration files produced by MT3DMS and SEAWAT (which are the same). Dissolved concentration are strored in files named 'MT3D???.UCN' and the sorbed concentration in fles name 'MT3???S.UCN', where ??? is the species number (001, 002, 003, ...) in the order as used by the program. Hence reading the sorbed concentration of species 3 would be

```
CS3 = readMT3D('MT3D003S.UCN');
```

The rest is similar to the heads file, except for *totim* being called *time* here.

The function readBud also gives a struct areay with one element per time for which data was saved. Each time elementn is a struct containing the cell-by-cell flow data computed by MODFLOW. There are many cell-by-cell flow types at any time, that are indicated by labels:

'FLOWRIGHTFACE', 'FLOWFRONTFACE', 'FLOWLOWERFACE', 'WELLS', 'CONSTANTHEAD', 'DRAINS', 'RECHARGE' etc.. In principle, each package ats its contribution. Hence each time element of the budget file and the read-in budget struct, has as many full size 3D arrays as there are flow contributions to each of the MODLFOW cells. Therefore, the budget file can be quite huge. To handle this, readBud can read in specific parts of the budget file. See help readBud for instructions. This way one could, for example, only read in the cell-by-cell flows due to the drains. Each time element struct has a field label holding the labels shown above for those cell-by-cell flows thet are present. The actual data are in field term. This is a cell array with as many full size 3D array of cell by cell flow data as there are labels. The order of the term arrays correspond to that of the labels. Hence to pick out the cell-by-cell flows by the wells for time element 10, we can proceed as follows:

```
B = readBud([basename '.BGT']);
QWells = B(10).term{strmatchi('WELLS',B(10).label)};
```

So that QWells is the full 3D array is well extractions (negative) and infiltrations (positive) during the particular time step B(10)....

To see all labels for that time element (note that the labels may vary from time to time, for instance of wells are not present during every stress period),

```
labels = B(10).label
```

etc.

Notice that the budget file does not containe any time information, only the stress period and the time step within the stress period. If one makes sure that the heads file and the budget file are synchronous (by using the same frequency in the labels HDDFL and LCBDFL in the PER worksheet), then that time can be obtained from

```
t = H(10).totim;
```

in this case.

The best way to see how these functions work is to look in the *mf_analyze.m* files of the examples and to work with them by interactively trying them out and inspecting the result.

A tip to combine data like the time when spread over many of the same structs like the ones obtained from the functions *readDat*, *readMT3D* and *readBud*, is joining them into a list. For instance

```
time = [H.totim]
```

Makes such a list and thus yields the times of all elements of struct array H in one array, that can then be used to plot data as function of time or to find retrieve/select the time elements of the heads H between some specific times $t1$ and $t2$:

```
Hsub = H(find(time>=t1 && time<=t2));
```

Setting up a model in Matlab (see mflab/examples/mf2k/ex1)

10 Some exercises in using the grid object (gridObj)

Setting up grids and manipulating them as well as placing real-world wells in the grid is best done using the grid object. An extended tutorial on using the gridObj can be found in the directory metioned hereafter. Handlichg gridobjects is a must for every user of mfLab. The HTML file in that directory was generated by Matlab while doing the exercises. It is therefore correct. It was not (yet) feasible to transfer it ot the mfLabUserGuid for technical reasons only.

```
/Users/Theo/GRWMODELS/mflab/examples/Tutorial/gridObj/HTML
```

11 Adding information to the grid

This part of the manual will be added to the HTML file for the grid

The following should be added to the grid

```
[CHD, PNTSRC] = gr.bcnZone(basename,'CHD',ZoneArray,zoneVals,zoneConc);
```

This particular call generates input for changing head boundaries. Therefore, its ouput is named CHD as required by mfLab and the type of the boundary condition is passed to the method throug the second argument, in this case 'CHD'. These can be replaced by RIV, GHB, DRN etc. Several calls in sequence than thus generate the required data to generate the appropriate input for the models to be run.

ZoneArray is an array of size (Ny,Nx,Nz) of integers where each value is a zone number. This allows defining any area in the model as belonging to a given zone. Negative zone numbers are ok, as mfLab looks only to the absolute value of the zone numbers. This allows for instance to use the IBOUND array, which MODFLOW requires anyhow, to be used as the ZoneArray. MODFLOW interpretes negative values in IBOUND as cells with fixed head and MT3DMS/SEAWAT inteprete negative values of the similar ICBUND array as cells with fixed concentration.

zoneVals has as many lines as there are zones to generate information for. The first column is always the zone number to look for in the zoneArray. The following columns will all simply be added to the output. So, a line in zoneVals like this

```
zoneVal = [5 value1 value2 value 3 ....; 7 value1 value2 value3 ...];% zoneVals array for z
```

will return in the BCN array (i.e. the WEL, RIV, GHB, CHD arrays) as follows

```
[SP L R C value1 value2 value 3 ....]
```

here SP is the stress period number and [L R C] the location of the cell belonging to zone 5 (in this case) and value1 value2 value3 etc exactly as given in the zoneVals array. Note that there will be data for each stress period. In this case these data will be the same for every stress period. We'll deal with time-varying data shortly.

Zone values may also specified as a cell array to allow more flexibility in the input. So, using { }, the zoneArray becomes

```
zoneVal = {5 value1 value2 value 3 ....; 7 value1 value2 value3 ...};% zoneVals array for z
```

In this case, value1 value2 etc may be of diffrent type. The can no also be given as vectors with the length corresponding to the cells in the given zone in zoneArray. An error occurs is the size does not match. It is however, straightforward to check:

```
N = length(zoneArray == iz)%     N is the number of cells belonging to zone iz.
```

To deal with time-varying input in the same call we also have to pack the zone information into a cell array. In this case we can replace any of the numerical data in value1, value2 value 3 ... by a string like

```
zoneVal = {5 value1 'hdr1 value 3 ....; 7 value1 'hdr2' 'hdr2' ...};% zoneVals array for zo
```

Like with wells, the strings are now interpreted as headers of columns in the PER worksheet holding time varying data pertaining to the zone in question. The ouput in will now differ for each stress period but during any stress period will be constant for the particular zones.

Next to the specific boundary condition ouput stored in the obliged boundary names WEL, DRN, CHD, RIV GHB ... for recognition by mfLab, we need to specify the concentration for each zone and each species in the model. This is done through the zoneCond array, the 5th argument of the call as shown above. The zoneConc array follow the same structure as the zoneVals array, i.e. with one line per zone but without a zone number in the first column, because the zoneVals array already had this number. Any value on the same line will be added to the PNTSRC output. So a zoneConc array like

```
zoneConc = [5 conc ITYPE conc1 conc2 conc3; 7 conc ITYPE conc1 conc2 conc3]
```

will yield PNTSCR output like

```
[SP L R C conc1 ITYPE conc1 conc2 con3 ....]
```

with SP the stress period [L R C] the location of this zone cell in the grid, followed by the user-specified data, exactly as specified. Like before, if the zoneConc array is a cell array, using { } instead of [ ], the same flexibility is obtained as described for the zoneVals array, allowing both vectors for with different values for each cell within a zone to be specified or, when strings are used, to specify time-variying concentrations in the corresponding columns in the PER worksheet of the *basename.xls* workbook file. Lastly if zoneConc is just a single value of a single horizontal vector (a simple set of values, while zoneVals has multiple lines) these value or values will be used for all zones.

Finally, how to glue together the different PNTSRC outputs of all the calls without redundant copying?

First get the number of stress periods, so that we can count the ouput of each call to gr.bcZone,

```
[~,~,~,NPER]=getPeriod(basename);

% Then start with the unigue call to gr.well or with an empty cell array
% for PNTSRC and add all subsequent contributions to it. For example

PNTSRC={}
[well, WEL, PNTSRC(end+(1:length(well)))] = gr.well(basename,'wells',HK);
[CHD, PNTSRC(end+(1:NPER))] = gr.bcnZone(basename,'CHD',IBOUND,zoneValsCHD, zoneConcCHD);
[GHB, PNTSRC(end+(1:NPER))] = gr.bncZone(basename,'GHB',IBOUND,zoneValsGHB, zoneConcCHD);
[RIV, PNTSRC(end+(1:NPER))] = gr.bcnZone(basename,'RIV',IBOUND,zoneValsRIV, zoneConcRIV);
```

etc. you may repeat each call as often a you like with as many different zoneArrays as you need.

Of course you must first make sure that the IBOUND array has the correct zone number at the correct locations. You have to specify the different zoneVals and concVals arrays (which are generally small and simple) and you have to make sure the columns corresponding to any specified headers are present and filled with the correct data in the PER worksheet.

This pattern if setting up models is followed in virtually all the examples.

12 The well object

Another useful object is, obviously, a well object. A will object will carry its own name, coordinates, screen elevation, well radius and anything you might want to add to it, such as drilling method, year of construction etc. The extraction may also be added to it, together with its injectin concentration, after the simulation, the output concentration may be computed from the concentration file and added to each well, then each well object really cary all its pertinent and dynmamic data, which can be used to provide numerical and graphical output.

On top of that we can put the well in the grid, so that it then not only contains the model coordinates, but also the grid indices ix, iy and iz. while, at least for vertical wells, ix and iy are scalars, iz will generally be a vector, because often the well screen spans several model layers. Next to that we give it the list of [*Layer Row Column*] so that we can use that to generate input to MODFLOW etc, in the Layer Row Column form in which it requires boundary condition information. Also, we give it the global index idx, so that we can directly put items pertaining to the well into the grid or request data from it, for instance flows from the budget file.

Well objects thus allow easily plotting time curves for individual wells and even for observation points as well as for well series (wells that share certain characteristics like their number or *id*. wihtfor arbitrary locations in the grid, among which for wells. We also add to the well the fraction of the total flow from the screen comes from the cells penetrated by the screen, such that the sum of these fractions is 1. These fractions equal the horizontal conductivity of the well-cells multiplied by there vertical thickness and divided by the screen length. Although the extraction distribution computed this way is not exact for wells that are partially penetrating, it is a practial way to make the most reasonable division. If it is essential for your application, you should use the so-called multi-node well (MWN). This well type makes sure that the heads inside the wel cells are the same, while the total extraction is equal to the one assigned to this well.

The use of the wellObj is explained in the HTML file in the directory

```
mfLab/examples/Tutorial/wellObj
```

13 The problem

The best way to show the working of *mfLab* is to look at a simple example. We use the example ex1 in the *mfLab/examples/mf2k* directory. This is actually the first example worked out in the MODFLOW-2000 (*mf2k*) manual (Open-File Report 00-92, see figure 10). It is described in the manual of MODFLOW2000 (*mf2k*) shown below on page 89ff. The example is historic; it was also used in the original manual of MODFLOW-88 [4].

http://water.usgs.gov/nrp/gwsoftware/modflow2000/modflow2000.html]

The situation to be modeled is shown on the cover page of the mentioned manual. The model network and its data are shown in Fig.11. It consists of 3 aquifers that are separated by 2 aquitards. The model has 15 columns and 15 rows. The cells are all 5000x5000 ft size. The heads at the left boundary are fixed in the first two aquifers. Recharge on the phreatic aquifer is $3e10^{-5}$ ft/s. There is an east-west orientated drain in the first layer in row 8 spanning cells 1 to 10. The first layer has a free water table. Therefore, it has no transmissivity but a conductivity (i.e. K=0.001 ft/s), while the transmissivity is dynamically computed by MODFLOW by multiplying it with the actual wetted thickness of the layer, which is not known beforehand. The transmissivities of the confined second and third aquifers are $T = 0.01$ and $T = 0.02 ft^2 d$ respectively. The flow is in steady state.

The objective is to compute the elevation of the water table, the heads in the aquifers and the discharge through the fixed-head boundaries and the drain.

14 Building the example model, *mf_ adapt*

Navigate to the directory in which you want your model. Make a new directory if necessary. While you may have copied a directory of an earlier model to start with, we assume for now that we start from scratch (which we never do in practice). In that case open a new mfile and save it under the name *mf_ adapt.m*.

14.1 Specify the grid lines

Every model is a 3D block consisting of Ny(number of rows), Nx (number of columns), Nz (number of layers) cells that also have a block shape. One way or another we have to specify the grid lines of the model. They may conveniently be called xGr, yGr, zGr and are always aligned along the three axes in this order: columns, rows and layers. These grids may have little to do with world coordinates, they are measured along the ribs of the Modflow mesh. They do not have to start with zero, because for Modflow only the width of the cells, that is the differene between successive model coordinates matter. Further, it does not matter in what units the coordinates are expressed, as long as all units used by Modflow are consistent. But one should not the ITMUNI and the LENMUNI parameters in the MFLOW sheet of the workbook, which determines which units Modflow uses to print data. It should be noted furthr, that all Modflow-suite models are threedimensional, so that always the grids in the three directions are necessary even if the model has one layer or consists of only one column or row or even a single cell. The positive direction of the gridcoordinates is always from left to right for xGr, from front to back for yGr and from bottom to top for zGr. MFlab does not make any attempt to use real-world coordinates of any specific coordinate system. If necessary model coordinates can be translated into other measurement systems. MFlab has functions to rotate the model and to translate coordaintes from the Dutch national metric system to Google's WGS84 and vice versa.

xGr and yGr may be given first using some other method such as reading a GIS file for the zGr, or the three directions may be specified at once. MfLab has several functions to generate smooth grids, see function *sinespace* in mfiles/gridcoords and makegrid, while use may also be made of other functions like Matlab's one *linspace* and *logspace*. Because the column width is the same for all rows, xGr is generally a vector as is yGr. Using Meshgrid, it is straightforward to generate full XGR and YGR matrcies with the coordinates for each cell corner. These are necessary if one wants to rotate the gird, perhaps for visualisation on a map with a differenc coordinate system. The use of capitals is adopted, but not obligatory to denote full coordinate matrices and full 3D cell value arrays as well.

In this example, we may specify the coordinates of the grid lines xGr and yGr as follows:

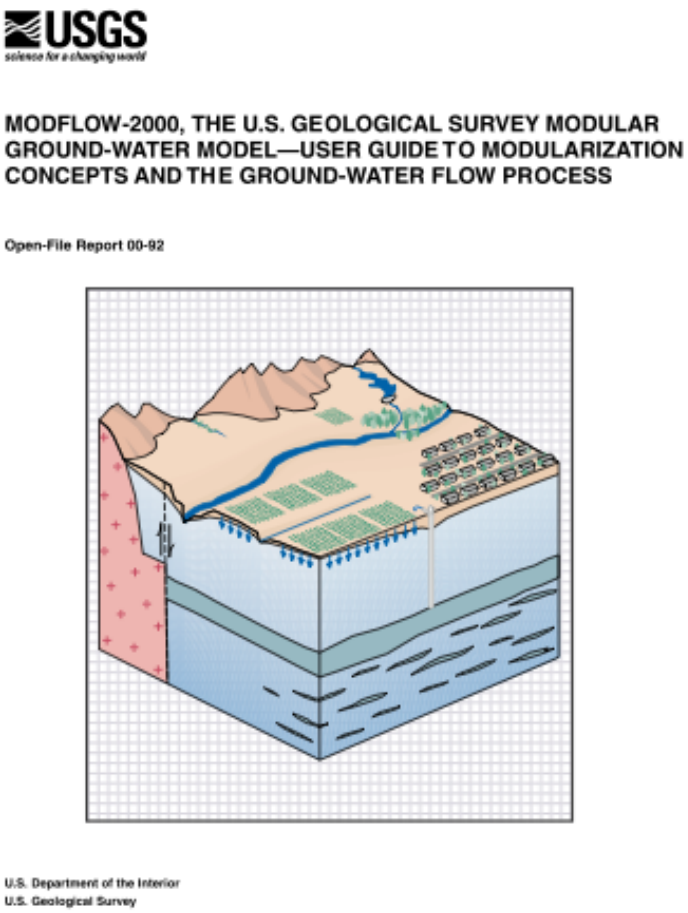


Figure 10: USGS: Open file report - MODFLOW2000 User Guide

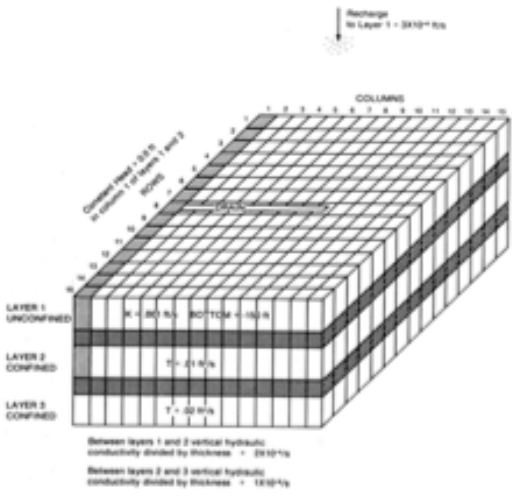


Figure 11: Model of example 1

```
xGr=(0:15)*5000;
yGr=(0:15)*5000;
```

Then we may compute the cell center coordinates, cell sizes and the number of cells using in each direction. This is conveniently done using *mfLab* function *modelsize*

```
[xGr,yGr,xm,ym,Dx,Dy,Nx,Ny]=modelsize(xGr,yGr);
```

One may inspect *modelsize* (or its 3D counterpart *modelsize3*) to see what it does. $Modelsize()$ makes sure that the xGr, xm and Dx are row vectors and the yGr, ym and Dy are column vectors. Also that xGr is increasing while yGr is decreasing. This ensures that the top line of a printout or of scrolled of a model layer corresponds with the highest yGr coordinate, as you would intuitively expect.

Furthermore, the elements in xGr and yGr after having past *modelsize*() are sorted and duplicates are removed. This means that you may specify grid coordinates in any order and include duplicates, the resuls are ordered and without duplicates. This is something is convenient if a mixture of grids and finer sub-grids around individual objects have to be specified and subsequently united into a single grid. So *modelsize()* is a convenient grid housekeeping function which also prevents cluttering of the *mf_ adapt* script with unnecessary detail.

modelsize() also yields the coordinate vectors of the cell centers, xm and ym, those of the size of the cells, Dx and Dy and the number of cells of the model layers, Nx,Ny

Some of the convenience of using of the grid housekeeping function *moddelsize*() may be seen from using *modelsize()* to also shift the coordinates such that 0.0 becomes the center of the model. This may be one using the *mean* function:

```
[xGr,yGr,xm,ym,Dx,Dy,Nx,Ny]=modelsize(xGr-mean(xGr),yGr-mean(xGr));
```

modelsize3() *modelsize3()* is the 3D variant, which works as follows*:*

```
[xGr,yGr,zGr,xm,ym,zm,Dx,Dy,Dz,Nx,Ny,Nz]=modelsize3(xGr,yGr,zGr);
```

The 3D variant also make sure that the zGr is ordered such that the highest values come first.

Because the zGr values can be different in each vertical column, zGr input may be a full 3D array. In that case the output zGr will also be a 3D array. The number of columns and rows equal that of the cells in a layer and the number of zvalues in each vertical column corresponds to the top and the bottom of each cell. In hat case the array name Z is generally used for the output zGr. MfLab requires the 3D coordinate array Z for each model to be defined.

Having the finite difference network coordinates, we may now define the IBOUND array to tell MODFLOW which cells are active and which cells will be treated as having a fixed head. *(mfLab* uses the IBOUND array to determine the size of the model).

14.2 3D model arrays

Orientation of Matlab arrays compared to MODFLOW arrays Note that all 3D arrays in *mfLab* are [*Row*, *Column*, *Layer*] so the orientation of the grids in calls to functions dealing with 3D arrays is generally in this order $[yM, xM, zM]$. This is the natural orientation in Matlab which is most convienent to use in the Matlab environment, the [y,x] orientation also corresponds to what is used in mathematics. For Matlab the Rows (y) are the first dimension, the Columns (x) are the second dimension and the Layers (z) are the third dimension, always !!

This implies that Ny (the rows) always comes first in the array specification, followed by Nx (the column), and finally the Nz (vertical direction). In MODFLOW, on the other hand, the orientation is *Layer*, *Row Column*. This seems confusing at first, but sticking in Matlab with its natural orientation (*Row, Column, Layer*) facilitates anything you do and soon becomes natural.

Writing the arrays in their correct Modflow orienation is dealt with by *mfLab* using the functions in *mfLab/mfiles/write directory*. But a user should never have to deal with them.

IBOUND-array This array defines the cells that are constant head (values<0), inactive (values=zero) and the ones that will be computed (values>0). A simple definition may be like

```
IBOUND=ones(NY,NX,NZ);
```

After which all rows in the first column in layers 1 and 2 can be defiend as fixed head by setting their IBOUND value equal to -1:

```
IBOUND(:,1,[1 2])=-1; % first column in first 2 layers have fixed head
```

Note that IBOUND can be utilized in multiple ways, for instance as zone array, as long as values equal to zero are meant to be inactive and values less than are meant to be fixed heads and values gerater than zero meant to be cells whose heads are to be computed.

Other arrays necessary for the model We may now proceed specifying the other necessary arrays to define our first model. To see how this is done refer to the example in the directory *mflab/examples/mf2k/ex1*; the *mf_adapt.m* script is completely and extensively documented using interlaced comments (text starting with a %).

What variables and therefore, arrays must be specified? Of course, those arrays that are required by MODFLOW and the other programs you intend to run. *mfLab* sticks as closely to the original manuals as possible regarding the naming of the arrays. For instance, if the BCF packages is used, you may have to specify transmissivities, called TRAN in the manual and so *mfLab* expects to find an (NRow,NCol,NLay) sized array called TRAN in the workspace. If instead, the LPF package is to be used, as specified in the NAM worksheet of the accompanying workbook, then, according to the original manual, the user has to specify HK which are the horizontal conductivities. Hence, *mfLab* expects to find a 3D array HK in the workspace when it wants to write out the input file for this LPF packages, and so on. If the required array are missing, an error is issued and the program will stop.

Look in the *mfLab* script mf_adapt in the folder of this example to see which other arrays are defined therein. The user should also inspect the Modflow manual to checkout the input requirements and see the one to one correspondence between *mfLab* and the original Modflow manual.

The names *mfLab* looks for can be found in the top of the *mf_run.m* script, directly after the call to *mf_adapt*. *mf_run* is the script to launch *mfLab*. In general, *mf_adapt* defines the grid and all 3D matrices constituting the model (see figure 12). Fine-tuning parameters and the specification of stress periods and layer properties (such as layer type and wettability) are in the accompanying workbook *<<basename>>.xls*, hence, in the current example case in the worksheet *ex1.xls*.

15 The accompanying Excel workbook *ex1.xls*

Every *mfLab* model consists of at least three files, *1) mf_adapt* to construct the model, *2) mf_analyze* to extract, interpret and visualize its results and 3) the workbook *<<basename>>.xls*. The latter holds model parameters as well as the definition of the stress periods and the layers parameters (such as layer type, wettability and so on). It also has a worksheet NAM which defines which packages are to be included in the run and which files are associated with each package. Packages can be switched on and off on this worksheet, by changing the switch on each line from 0 to 1 and vice versa.

Note that basename is defined near the top of *mf_adapt.m*. This basename defines the basename of all input files, including the accompanying workbook. However the local files *mf_adapt.m* and *mf_analyze.m* are always named the same for every model. Therefore, there can be only one model in a single directory.

The workbook contains many worksheets with names that *mfLab* looks for when seeking the required parameters. These names must not be changed. Neither must the names be changed of the parameters on the sheets. Anything else can be changed and you may add as many sheets and information as you like, because *mfLab* will only try to extract exactly those parameters and values it needs for a particular model.

Instead of describing exactly the contents of the workbook at this point, refer to the the worksheet description elsewhere in this user guide.

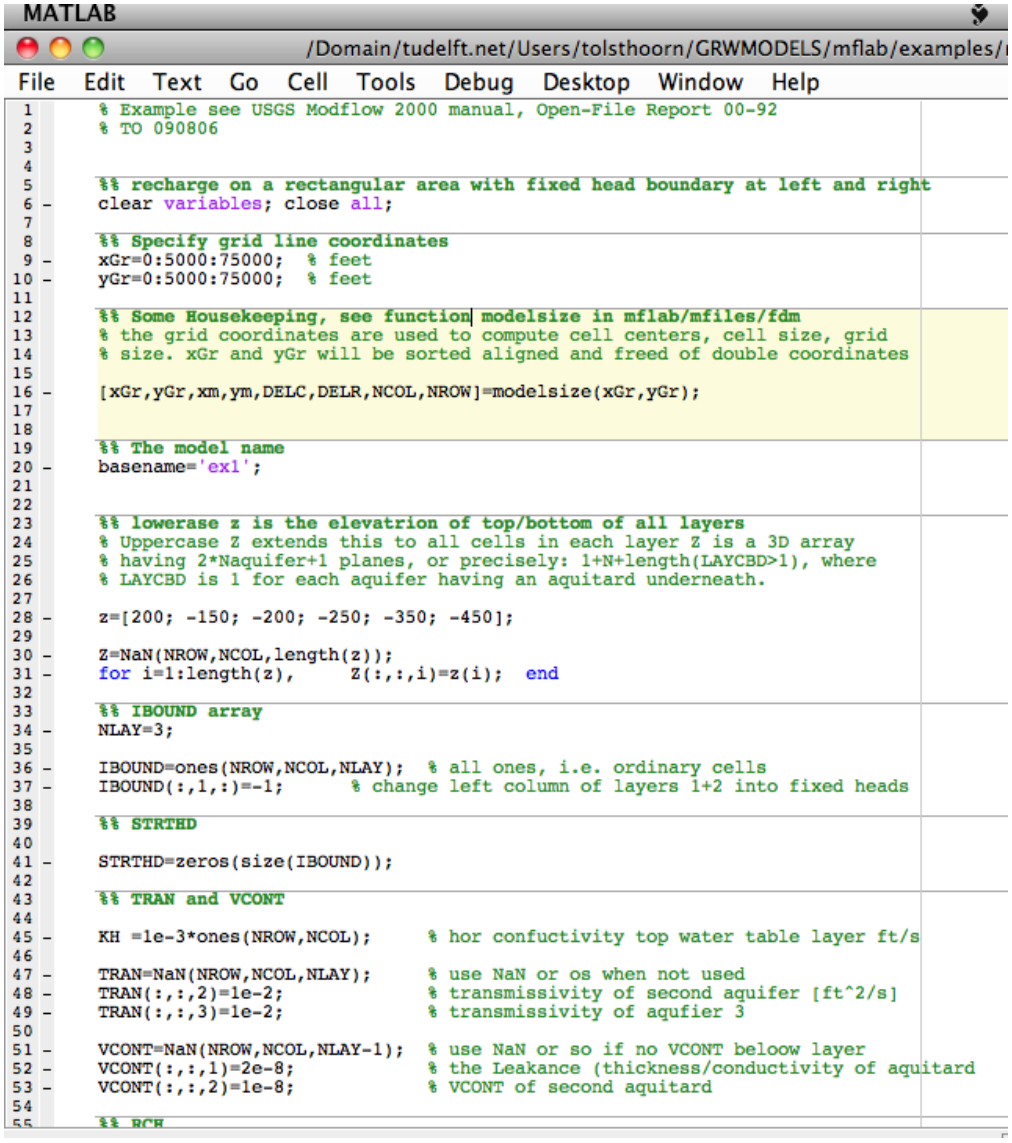


Figure 12: First part of script mf_adapt defining the model

15.1 Inspecting the accompanying Excel workbook

The workbook contains parameters relating to the model. These parameters are arranged in worksheets within the workbook. Figure 18 shows the different sheets as they appear at the bottom of the Excel screen.

Not all the worksheets NAM, MFLOW (MODFLOW), MT3D, PER (stress periods), LAY (layers) etc. are necessary for this simulation.

The example problem is specified in feet and second. MODFLOW doesn't care as long as the dimensions are used consistently across all inputs (i.e. recharge must thus also be specified in feet/s rather than in/day or in/year). However for printed output it may be convenient to see the correct dimensions. These can be specified by changing the top two parameters, ITMUNI and LENUNI in the MFLOW worksheet shown is the figure. The comment that pops up when clicking the cells explain the meaning of these values. Other parameters can be set in the same way.

The worksheet MT3D is not needed here. it has the same structure as the MFLOW sheet, but it was held separate because of the large number of specific transport parameters required by MT3D.

The worksheet PER (see fig.23) contains the information regarding the stress periods.

Instead of the MODFLOW text values 'SS' or 'TR' to indicate that the computation in a given stress period is steady-state or transient, *mfLab* uses the column ISTRAN (=is transient') with an easier to handle numeric value 1 to indicate transient and 0 to indicate of steady-state flow in each stress period.

The PER worksheet also contains output control information (see section describing the PER worksheet).

The worksheet LAY (see figure 24) contains the parameter information for the layers of the model. The structure of the worksheet LAY is similar to that of the worksheet PER.

The sheets for WEL, DRN, RIV, GHB and CHD are all structured similarly as a list (see figures **??** and **??**). The first row of these worksheets contains the names of the columns. The first column is the stress period number. It is followed by the layer, row and column number and, finally, the necessary values.

mfLab requires the stress period number in the first column. This is to unambiguously recognize each specified line and to allow counting lines per stress period by simple selection based on the stress period number.

16 Post-processing and visualizing with *mf_ analyze*

If everything runs fine and the executable terminates normally, model output has been written to output files as specified in the NAM worksheet and the file unit numbers near the top of the MFLOW worksheet. We then use a script called *mf_ analyze* to read, interpret and visualize these outputs. Of course, this script can be given an arbitrary name, but throughout *mfLab* the name *mf_ analyze* is used (figure 13).

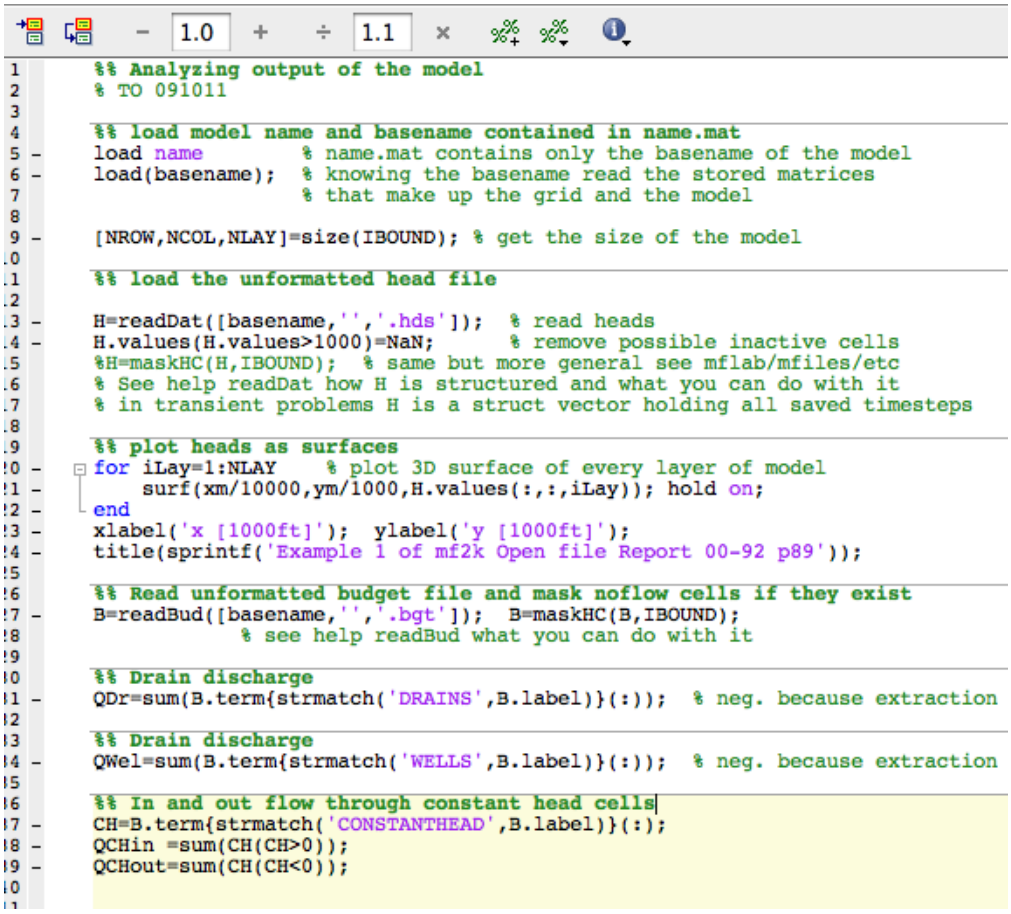
It is possible to invoke *mf_ analyze* automatically after running the models. While this may be convenient for production runs, it is generally not a good idea when the model is still under development. It is then preferred to let the model finish first, and check to see that it has terminated normally before invoking *mf_ analyze*. If not, you may end up inadvertently using the output of some earlier run, because the most recent run has failed and, therefore, has produced no or wrong output. This can be confusing at times. So, at least in the development phase, it is advised to first invoke *mf_ run*, then check for normal termination of the *mf++* executable and then use *mf_ analyze* to interpret and visualize the results.

The script *mf_ analyze* is always tailor made, i.e. specific to the model in question. Anything necessary to optimally visualize and interpret the results of the model are put in. The best way to make your own *mf_ analyze* is to copy an existing one to your project directory and edit it. There is an almost unlimited versatility possible using all of Matlab visualization and animation functions. The possibilities may seem overwhelming at first. But starting from one of the examples is always a good idea. The scripts will not vary too much between the various examples. Some may visualize using contours in the plane of aquifers, others may focus more on cross sections and or show transient dynamics of heads and concentrations at specific observation points or animation. Matlab functions often used for this visualization are *contour*(), *contourf*() *surf*(), *surface*(), *slice*(), *movie*(), *quiver*() and so on. See the examples.

However, you always have to read in the output produced by the *mf++* models. *mfLab* provides a number of functions to read the unformatted files produced by MODFLOW, MT3DMS and SEAWAT. You will likely always need the corresponding functions *readDat*, *readBud* and *readMT3D* to read the contents of these unformatted files into the Matlab workspace. These functions, however are very flexible; they allow reading

out any portion of these files and do it fast. With them, you don't have to load a 2GB output of MT3DMS in memory entirely and run out of computer memory. See the description of the options elsewhere in this user guide.

Once loaded into the Matlab workspace you can do anything with the data, of which visualization with one or more of the mentioned Matlab functions is just one possibility.



```
%% Analyzing output of the model
% TO 091011

%% load model name and basename contained in name.mat
load name        % name.mat contains only the basename of the model
load(basename);  % knowing the basename read the stored matrices
                 % that make up the grid and the model

[NROW,NCOL,NLAY]=size(IBOUND); % get the size of the model

%% load the unformatted head file

H=readDat([basename,'','.hds']);  % read heads
H.values(H.values>1000)=NaN;      % remove possible inactive cells
%H=maskHC(H,IBOUND);  % same but more general see mflab/mfiles/etc
% See help readDat how H is structured and what you can do with it
% in transient problems H is a struct vector holding all saved timesteps

%% plot heads as surfaces
for iLay=1:NLAY    % plot 3D surface of every layer of model
    surf(xm/10000,ym/1000,H.values(:,:,iLay)); hold on;
end
xlabel('x [1000ft]');  ylabel('y [1000ft]');
title(sprintf('Example 1 of mf2k Open file Report 00-92 p89'));

%% Read unformatted budget file and mask noflow cells if they exist
B=readBud([basename,'','.bgt']);  B=maskHC(B,IBOUND);
            % see help readBud what you can do with it

%% Drain discharge
QDr=sum(B.term{strmatch('DRAINS',B.label)}(:));  % neg. because extraction

%% Drain discharge
QWel=sum(B.term{strmatch('WELLS',B.label)}(:));  % neg. because extraction

%% In and out flow through constant head cells
CH=B.term{strmatch('CONSTANTHEAD',B.label)}(:);
QCHin =sum(CH(CH>0));
QCHout=sum(CH(CH<0));
```

Figure 13: Top of script *mf_ analyze.m*

Finally one may use the results of the budget file read by *readBud* in many ways. Of course, it is possible to run the MODFLOW companion USGS program ZONEBUDGET, which also requires the budget file or use the budget file data directly in Matlab:

Reading the budget file results in a structure array in Matlab's workspace, which contains the read data. The length of this array is equal to the number of snapshots saved into this file. This Matlab structure array, or "struct" for short, also holds exactly which time, stress period and time step number each snapshot refers to. It further holds the labels that the file contained, which specify the type of data read. It finally contains the data itself. As the user can specify exactly which rows, columns and layers in which arbitrary order to be read, the struct also shows the numbers of these rows, columns and layers. Hence, the struct generated by *readBud* in the Matlab workspace contains all possible information regarding the budget data produced by MODFLOW.

The same is true for the struct arrays generated by *readDat* and *readMT3D*. All three reading functions are similar, but there are small differences to cope with the exact specification of the unformatted output files produced by MODFLOW and MT3DMS. The options of the reading functions can be seen by typing

help readDat

help readBud

help readMT3D
in the Matlab workspace.
This structure of the struct can be inspected in the usual Matlab way.
For example: If the budget file is read in to the struct B by invoking
B=readBud(*'ex1.bgt'*)
Then B(i) is snapshot i.
length(B) is the number of snapshots in the struct array.
B(i).totim is the simulation time until this snapshot
B(i).period is this snapshots stress period number
B(i).tstp is this snapshots time step inside this stress period
B(i).label is the list of labels contained in this snapshot
If label 3 is the item you want, then
B(i).term{3} is the 3D array containing the flow terms indicated by label 3
To get specific rows, columns and layers form this flow terms array:
B(i).term{3}(rows,cols,layers), where rows is a list of layers (or all ":"), cols a list of columns and layers a list of the layers you want to use.
B(i).rows is the list of rows in the struct, which correspond to the ones selected from the budget files or all rows in the file if the rows were not specified to *readBud*.
B(i).cols and B(i).lays similarly contain the columns and rows obtained form the budget file.
Hence, any information can be extracted form this struct array.
To see all times in this struct array, type
[B.totim]
whiteout indices. Anything the standard powerful Matlab way.

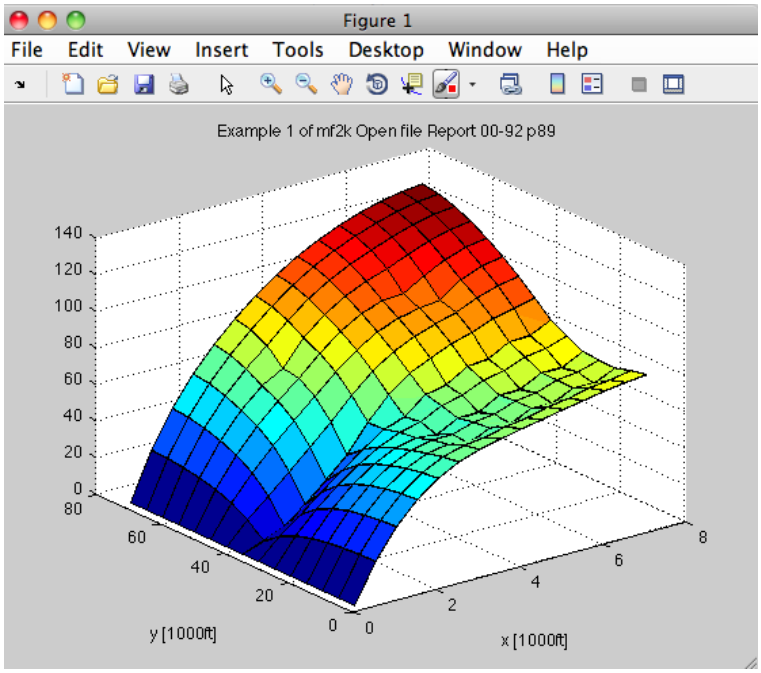


Figure 14: Water table shown as surface

Without wanting to give a course in Matlab here, some possible a little more advanced use of the budget struct will be shown now. For instance, to get the total net outflow during snapshot i through drains cells into a variable QDRN, we could type

QDRN=-sum(B(i).term{strmatch(DRAINS,B(i).labels)}(:); % also notice the - sign here to get discharge as infiltration is positive in MODFLOW.

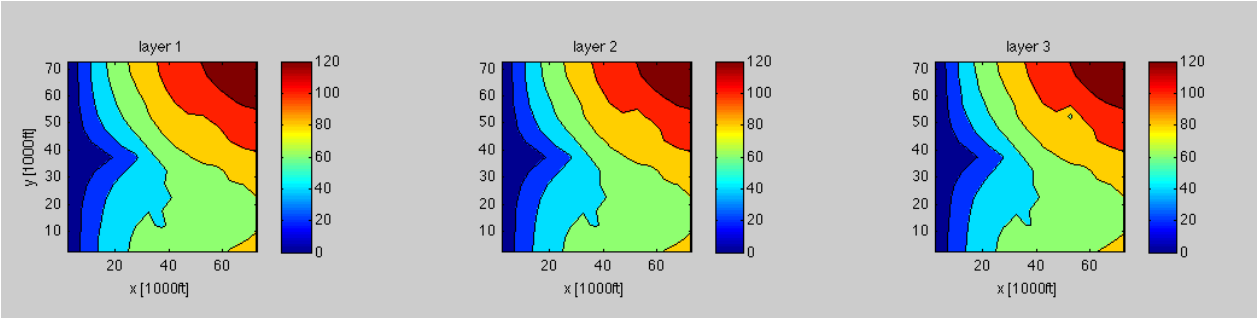


Figure 15: Head contours of the three aquifers

But, of course, since we have only a single time step in this example, i can only be 1. There are several other examples of such water balance computations

Perhaps interesting is the computation of the in and outflow through constant head cells:

Q_CH=B.term{strmatch('CONSTANTHEAD',B.labels};

Q_CHin =sum(Q_CH(:)<0); % total extraction

Q_CHout=sum(Q_CH(:)>0); % total injection

Notice that this manner is a little different from what's in the script *mf_analyze.m*, but works just as well.

You may use the Matlab standard function *spy* to see where the different drains, rivers and well are in the model.

A way to obtain the water balance of a zone is indicated at the end of *mf_analyze*.

These lines are just to show the flexibility and versatility of the Matlab environment to compute useful results from data contained in arrays.

```
45      %% Read unformatted budget file and mask noflow cells if they exist
46 -    B=readBud([basename,'','.bgt']);  B=maskHC(B,IBOUND);
47              % see help readBud what you can do with it
48
49      %% Compute total drain discharge
50 -    QDr= -sum(B.term{strmatch('DRAINS',B.label)}(:));  % neg. because extraction
51
52      %% Computetotal well discharge
53 -    QWel= -sum(B.term{strmatch('WELlS',B.label)}(:));  % neg. because extraction
54
55      %% In and outflow through constant head cells
56 -    CH=B.term{strmatch('CONSTANTHEAD',B.label)};
57 -    QCHin =sum(CH(CH(:)>0));
58 -    QCHout=sum(CH(CH(:)<0));
59
60      %% Demo water balance of a zone
61      % define an arbitrary zone
62 -    zone=zeros(size(IBOUND)); zone(3:11,2:8,:)=3; zone(5:12,5:14,2:3)=5;
63 -    myzone=(zone==5); % select zone number 5
64      % get -1 at downstream and +1 at upstream side of zone
65 -    diffz=zeros(size(IBOUND)); diffz(:,1:end-1,:)=diff(myzone,1,2);
66      % imput inflow by multiplication of diffz with FLOWRIGHTFACE
67 -    qNetInX=B.term{strmatch('FLOWRIGHTFACE',B.label)}.*diffz;
68 -    QNetInX=sum(qNetInX(:)); % total net in due to horizotnal flow
69
70      % Extend this with the y and z directions and add the other terms
71      % inside the zone to obtain teh total zone water budget.
72
```

Figure 16: Bottom of m script mf_analyze.m

Some of the outputs that *mf_analyze* produced from the output files of the example are shown in the figures 13 and 16ibelonging to this section.

Notice that the structs produced by *readDat* (reading heads and draw-downs) and *readMT3D* (reading MT3DMS concentrations) are similar to the one produced by *readBud*, but simpler. The same technologies apply with some small adaptations that are obvious when inspecting the contents of these structs and the way they are used in the *mf_analyze* examples.

17 Example with particle tracking

We will work out example ex2 of the open file report 00-92, i.e. the manual of MODFLOW 2000, p89ff. Parameters as defined in the MODLFOW 2000 manual are not used in *mfLab*, because it is more efficient and less complicated to do any parameterization inside *mfLab/Matlab*.

The problem concerns a groundwater system consisting of three aquifers interlaced with two aquitards. In this model the aquifers will coincide with model layers and the aquitards with confing beds between them. The layer thicknesses are considered uniform as are the porosities and conductivities.

The *xGr*, *yGr* and *zGr* grid lines are defined are defined first. 3 aquifers with two interlaced aquitards requires 6 layer elevations.

```
xGr = 0:5000:75000;  % xGrid coordinates, feet
yGr = 0:5000:75000;  % yGrid coordinates, feet
zGr = [200 -150  -200 -300 -350 -450];  % Plane elevation vectorin feet

kh   = [1e-3; 1e-4; 2e-4]; % horizontal conductivity the three aquifers (model layers)
vkcb = [1e-6; 5e-7];       % vertical conductivity of the two aquitards (confining beds)
por  = 0.35;               % porosity, same for all aquifers  (model layers)
porcb= [0.2  0.3];         % porosity of the aquitards (confining beds)

LAYCBD=[1 1 0]; % defines that first and second model layers have confining bed below them?

% Generate a grid object, that stores the grid and yields useful additional grid informatio
% through its methods. Type gr to see its contents
gr=gridObj(xGr,yGr,zGr,LAYCBD);
```

Next spatial model arrays are defined using the grid object method "const", which yields a 3D array of the size of the grid uniformly filled with tha value(s) specified.

```
% which cells have fixed heads?
IBOUND=gr.const(99);          % default, all >0, meaning cells will be computed
IBOUND(:,1,1:2)=-1;           % overwrite left column for layers 1 & 2 with -1, meaning fix

STRTHD=gr.const(0);           % define all initial heads to be zeros IBOUND

POR   = gr.const(por);        % all layers will have uniform porosity por
PORCB = gr.const(porcb);      % all confining beds will have uniform porosity porcp
HK    = gr.const(kh);         % each layer will have horizontal conductivity equal to the
                              % corresponding values in kh vector;
VKCB  = gr.const(vkcb);       % each confining bed will have uniform vertical conductivity
                              % corresponding values in vkcb vector.
VK=HK*100;                    % set VK high to prevent vertical resistance in aquifers
```

Define the recharge as uniform.

```
RECH={3.0e-8*ones(gr.Ny,gr.Nx,1)};  % recharge in first stress period as cell array
```

Define the requested drain, first as a line and then generate the input for MODFLOW using the grid method "bcnLine" (bcn stands for Boundary Condition or Stress).

```
% The drain are generated here in real world coordinates. Each line contains
% the real-world coordinates of a point of the drain followed by its elevation and leakance
drn=[5000 37500     0 1/5000;...
     50000 37500  100 1/5000];  % [x y Elev Leakance] assumeing layer 1

% These points will be interpolated onto the center of the grid cells intersected by the d
% The leakance is matched with the size of the intersected model cells using
```

```
% the grid object method gr.bcn(data,'type'). This results in a list
% [iPer iz iy ix head Leakance] Default iPer=1
DRN=gr.bcnLine(basename,'DRN',drn);
```

Next, the wells are defined. The basename of the problem at hand provide the basename of the workbook that accompanies this problem. The second argument is the name of the worksheet within this Excel workbook containing the well data. To place the wells with given real-world coordinates into the actual model grid, the gridObj has to be passed and the horizontal conductivity array. This array is used to compute the portion of the well extraction from each model layer intersected by the well screen, based on the transmissivities of all intersected layers. The resulting well is an array of well objects that both hold as their data their real-world and their model grid coordinates.

```
well = wellObj(basename,'wells',gr,HK);
```

At this stage, the flow model is completely defined in the form of its required arrays. Further data, like settings of the layers, stress periods, choice and settings of the solver, choice of the packages to be used, are all defined in the workbook on the corresponding worksheets (NAM, MFLOW, LAY and PER).

Next we define data for particle tracking, required by MODPATH. In this case we let the flowlines start from two ponds defined in worksheet *"waterbodies"*of the accompanying workbook. We will show them in a figure once they are defined, to verify our input.

First show the model grid using the method "*plotGrid*" of the gridObj class. Then the two ponds are defined as objects of the class *waterBodyObj*, and plotted on the grid using the method "*plot*" of this class. This method is called with several plotting properties as arguments, which are basic properties of the Matlab patches drawn by the "*plot*" method.

```
gr.plotGrid;

pond = waterBodyObj(basename,'waterbodies','waterbodies (2)',gr);

pond.plot('edgecolor','g','facecolor','b','facealpha',0.25);
```

The *gridObj* class has several methods pertaining to particles for particle tracking: "*startLoc*", "*relloc2model*", "*plotMPathLocations*". The method *startLoc* requires a zone array of the size of the model and an array zonevals. The latter has one line per zone. Each line has the zone number followed by the root of the number of starting points, and the iFace to place them on. The iFace number refers to one of the six faces of the cell according to the definition given in the Modpath manual. So if this number is 5, 5x5 points will be placed in a regular 5 by 5 grid on that cell face. If iface is zero, the mentioned number of points will be the 3rd root of the number of points to be placed in the cell interior. I.e. 4 would then imply 4x4x4 points placed on a regular grid inside the model cells having the specified zone number.

```
%% Starting points for particle tracking using MODPATH

% Specify the number of points and iFace for these zones
zoneVals = {pond(1).nr, 5 6; ...    % second arg is n, third is iFace
            pond(2).nr, 1, 0};      % same for second zone
```

The method ZONE of the *waterBodyObj* class, as called below as *pond.ZONE(gr)*, generates a zone array in which each cell corresponding to a pond has the zone number equal to the pond number. Notice that pond can be an array of objects of the *waterBodyObj* class. The method "*startLoc*" of the *gridObj* class then uses this array to generate starting locations based on the *zoneVals* just specified.

```
LOC = gr.startLoc(pond.ZONE(gr),zoneVals);
```

These locations are now plot on a new figure in a wireframe of the grid (a mesh). First the figure and the mesh are generated

```
figure; hold on; view(3); xlabel('x [m]'); ylabel('y [m]'); zlabel('z [m]');

gr.plotMesh('faceAlpha',0.15); % show the grid as a 3D transparent mesh
```

Then the starting lcoations LOC, which are in relative coordinates, are converted to real-world coordinates and then plot in the grid, that is, on the current axis with the wireframe mesh already present.

```
xyz  = gr.relloc2model(LOC);

gr.plotMPathLocations(LOC);
```

Of, course, this latter plotting is not necessary for the model as such. It only serves verification if our point definiction is correct.

To check whether the genearation of the arrays necessary for the input of this model are correct, run *mf_ adapt.m* and check the individual arrays.

To run MODFLOW followed by MODPATH, type

```
mf_setup
```

If all goes well, the input for MODFLOW and MODPATH will be generated and both models will be run in sequence. For this to run the settings in the mentioned sheets have to be set correctly. In the example that comes with *mfLab*, this is the case. I.e. in the worksheet NAM, the MF2K and MODPATH are both swichted on, and the other models off. The packages LST, DIS, BAS, LPF, RCH, WEL, DRN, PCG, BGT, HDS are on and others off. In the PER sheet, Transient is off and so on.

18 Particle tracking with MODPATH

Particle tracking with MODPATH has been around since the beginning of MODFLOW. The most recent version is MODPATH6 (2012). For the first time in almost 30 years this version is a major update, based on the very same logic and methods that have been around unchanged since 1994. The new version assumes that interface builders will take care of visualization and that all input is supplied when the program is launched, without any interaction by the user.

The major interface issue is how to tell MODPATH where to put particles and when to launch them. One chooses between three input styles, 3 release option and a placement option. The most straight forward input is style 2, release option 1. This implies that particles are specified through their locations and that they are released at a given punt in time. MODPATH defines particle groups, so the launch time can be chosen for each particle group separately. Further release options allow launching from the same location at multiple times, either regularly or irregulary spaced times.

Short description of the worksheets in the Excel workbook

19 The Excel workbook used by *mfLab*

Groundwater simulation codes like MODFLOW, MT3DMS etc. require many parameter values to be set to guide the simulation. Some parameters are set for en entire simulation run, such as the file names to be used and the maximum number of particles in the MOC-procedure for MT3DMS. Some parameters, although fixed per simulation, may differ between the layers of the model, such as the layer type and the wettability of a layer. Other parameters will differ between stress periods within the simulation, such as the stress-period length and the number of time steps within each stress period. In general, each model (MODFLOW, MT3DMS, SEAWAT etc) and each package within a model (WEL, GHB, SWI etc) requires parameters to be specified. *mfLab* assembles these parameters conveniently in a single Excel workbook, where they are arranged in different worksheets. Figure 17shows different worksheets in an an accompanying *mfLab* workbook.

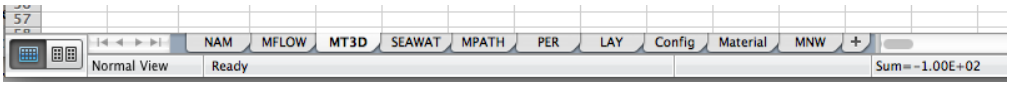


Figure 17: Bottom of Excel screen showing the tabs of the different worksheets in an accompanying *mfLab* Excel workbook

This Excel workbook can thus be regarded as a multiple-page parameter container. The workbook separates the actual construction of the model from the simulation parameters, which are often the same between simulations. As new packages and models are added to *mfLab* their parameters are either added to existing worksheets in the Excel workbook or obtain a separate new worksheet with a clear name, whichever works best and whichever provides the best overview to the user.

More worksheets may be added as new modelling capabilities are added to *mfLab*. Further, the user may add extra worksheets to his/her convenience. This may be practical for parameter handling and for linking such parameters with other project data chosen to be stored in the same workbook. Any Excel function and functionality may be used without affecting *mfLab* as long as it can find the worksheets it needs and the required parameters in them.

Also, there is no need to remove worksheets and parameters that are not required in a particular simulation. In fact, it is discouraged to delete such information, as a project may be extended later and require it then. A good alternative to removing sheets and parameters is hiding them using standard Excel functionality.

There are several main worksheet categories, those that set parameters pertaining to the entire simulation and the entire model, those pertaining to stress periods and those pertaining to layers.

When intending to build a new model, start with copying the mf_adapt, mf_analyze and Excel workbook from an old (similar) model, rename the worksheet accordingly to the basename specified at the top of mf_adapt and change the files into your new model.

The most extended worksheet can always be found as parameters. xls in the mfLab directory.

19.1 Worksheets pertaining to the entire simulation

Worksheets pertaining to the entire simulation and model, generally show a vertical list of package names in the first column, a list of the actual parameter name in the second column and the actual parameter value in the third. The fist column is not used, it merely identifies the package/module that uses the parameter. The second column is used by mfLab to look-up the parameter value next to it. Parameter names are generally unique across MODFLOW packages so that only the second column is necessary for this look-up.

The third column contains the sought parameter value. Some parameters may have a list of values. In those cases adjacent columns in the same rows are used to store the values. An example is the requested output times for MT3DMS in the MT3D worksheet.

Rows and columns not needed by a particular simulation are ignored. The user can, therefore, store any additional information on worksheets, without affecting mfLab as long as the first three columns contain package name, parameter name and parameter value.

Open lines are also ignored as well as any lines with unrecognized parameter names as long as the names that mfLab is looking for are present.

Generally also, each parameter name cell in the worksheets have a comment attached describing the meaning the parameter. This comment is mostly taken verbatim from the manual of the program or package that uses the parameter in question.

19.2 Stress period parameters

All parameters pertaining to stress periods are contained in the worksheet PER. See the description of this worksheet further down. The parameters pertaining to stress periods, regardless of the package that requires them, are stored in this worksheet PER. The first line of this sheet contains the package name and the second the parameter value that is used. Subsequent lines define each stress period. There are various options to define stress periods efficiently as described below under the description of worksheet PER.

19.3 Layer parameters

All parameters pertaining to entire layers are contained in the worksheet LAY regardless of the package that uses them. The fist line in this sheet contains the names of the packages; the second line holds the actual parameter names. Subsequent lines define the layers in the model.

There are various ways to define many layers efficient;y. See the description of worksheet LAY for details. The parameters in question are those that can be set on a per layer basis such as layer type and wettability. Note that cell-by-cell values are always defined in *mf_ adapt*. Sometimes there is a choice to either define

a parameter on a per layer basis in the worksheet or on a cell-by-cell basis in *mf_ adapt*. For instance, the diffusion coefficient for MT3DMS can be defined on a per layer and on a cell-by-cell basis. In such cases the layer definition contains a switch, based on which *mfLab* looks for the values in *mf_ adapt* or in the worksheet.

19.4 Boundary condition parameters

More or less historically *mfLab* has different worksheets to store boundary condition parameters such as WEL, RIV, DRN, GHB, CHD, PNTSRC. Each of them contains a single row of column headings at the top, which are followed by the actual parameter values. Each such row always starts with stress period number followed by a layer, row and column number. To the right of the column number follow the actual parameter values such as the flow for WEL, head and conductance for DRN etc. PNTSRC is needed by the SSM package of MT3DMS (and SEAWAT).

mfLab will look in these sheets only when the package requiring the data is active. Packages are activated in the NAM worksheet, see below. Even if the package, such as WEL, is active, the user has the choice to defined the data in this worksheet or directly in *mf_ adapt* or both. Any mixture is allowed, see description below.

Notice that the stress period number is required as the first column of all mentioned boundary conditions while this is not the case in the *mf++* programs. However, *mfLab* requires the stress period number to identify each line uniquely as belonging to a given stress period; it allows the user to specify the data lines in any order as they will be sorted after reading anyway.

20 Nam worksheet

The NAM worksheet, an example of which is shown in figure 18, is used to generate the name files that all *mf++* programs require. The NAM worksheet has a line for each package that *mfLab* knows about. However, only the packages that are active, that is, the ones that are "switched on", will be used to generate name files.

The lines on this worksheet are in arbitrary order and blank lines are ignored as is any line with an unknown package name. The switch is in column 4. Use 0 to switch a package off and use 1 (or non zero) to switch it on.

In the future switch values >0 may be read as a scenario number with respect to the parameters of this package during the current run. For the time being any value >0 means that the package is on.

The columns in this NAM sheet are as follows:

A — Package name as defined by the program (BAS6, ADV etc). See manual pertaining to the original external-party program. These package names are the exact names defined in the *mf++* manuals.

B — FORTRAN file unit number to be associated with the files. The choice of the unit numbers is free, but they must be unique within the set of active packages. *mfLab* will signal duplicates with an error message.

C — The extension used for the input file generated for this package. Notice that *mfLab* gives all files of the project the same basename as specified in *mf_ adapt.m* followed by the extension given in this column for each package. The choice of the extensions is free, but they must be unique. Of course, it is wise to apply some convention for them (OC for output control, HDS for head, BAS (or BAS6) for the basfile etc).

D — Flag giving the active status of the package. The flag has the following meaning:

- >0 — Package is active.
- 0 — Package is off, not used in the simulation
- <0 — (Only for VDF package, that is SEAWAT's Variable Density Flow package, means that the name file for SEAWAT is generated, but that the VDF package itself will not be used. This allows running SEAWAT as if it were MT3DMS, i.e. with density flow switched off.

- The future the flag may be interpreted as a scenario number such that if for a package a value of say 3 is used, *mfLab* will use the third value to the right of the parameter labels pertaining to this package. Currently it will only use the value immediately next to the parameter number.

E: Description of package (optional, ignored by *mfLab*)

F etc., ignored

To switch one on the package the value in the fourth column from 0 to 1. *mfLab* will try to generate an input file for every package that is "on". The active packages are marked green in figure 18.

The packages with DATA or DATA(BINARY) in the left column designate output files of the model. When BINARY is included, the output file will be unformatted. Currently *mfLab* can only read back in unformatted files using its functions *readDat*, *readBud* and *readMT3D* (see *mf_analyze.m* in any example directory). However, formatted output files are seldom used in practice.

The package name in the left column must obey exactly to the name prescribed by the code, e.g. BAS, BAS6, BCF, BCF6 etc. (see manual of mf2k, MT3DMS, SEAWAT etc).

20.1 MFLOW worksheet

This MFLOW worksheet in the Excel workbook contains all MODFLOW parameters that are constant during an entire simulation (see figure 19). The various packages that require the parameter are mentioned in the left-most column. The actual parameter name is in the second column and the actual parameter value in the third. Some parameters need more values; they will be placed in subsequent columns on the same row.

This worksheet also contains parameters for other packages like SWI (meaning Salt Water Intrusion) because the parameters refer to the entire simulation when run. Parameters are kept together as much as possible to prevent an unwieldy extension of the number of worksheets.

Each cell that holds a parameter name has a comment attached to it explaining what the parameter is. These comments are mostly taken verbatim from the respective *mf++* manual(s)

The parameters for the Salt Water Intrusion package (SWI) are also found on this worksheet.

21 MT3D sheet

The MT3D worksheet contains the parameters that are constant during an entire MT3DMS simulation. Part of it is shown in figure 21. The structure is the same as that of the MFLOW worksheet.

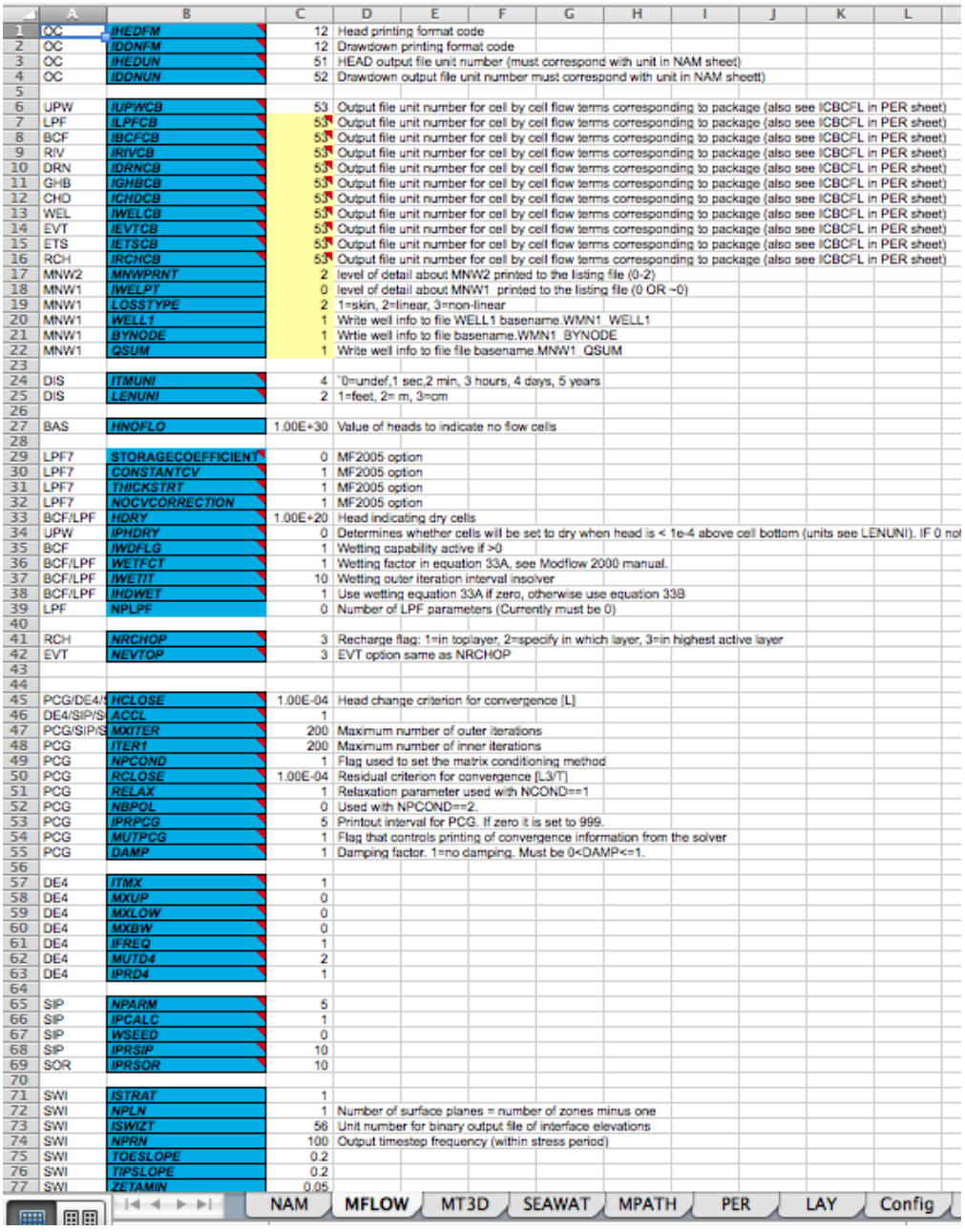
The TIMPRS parameter on the MT3D sheet may contains a series of numbers representing simulation times at which MT3D output is required. This parameter is active if NPRS>0. NPRS is then the number of times to be read for the TIMPRS. *mfLab* will only read as much as TIMPRS time values. A negative value of NPRS is interpreted as the transport-step frequency at which output is desired. In that case values of TIMPRS are ignored.

This frequency pertains to the transport time steps within a stress period, that is, the counter will be reset after every stress period. MT3DMS will always produce output at the end of a stress period, except when TIMPRS is set explicitly.

Synchronizing output of MODFLOW and MT3DMS, even within SEAWAT, which integrates both, can be an issue. To synchronize output of MODFLOW and MT3DMS use a large value for the transport step output frequency NPRS as well as for the output parameters IHDDFLG and ICBCFLG in the PER worksheet. This will cause output for both the MODFLOW and MT3DMS at the end of every stress period. Then set the stress period lengths in the PER worksheet to force output on the desired times.

	A	B	C	D	E	F	G
1	MODEL			ON / OFF		Description	Remark
2	MF2000			1	<====	Use mf2000	MFLOW
3	MF2005			0		Use mf2005 (double precision) instead of mf2k	MF2005
4	MF2007			0		Use mf2007 (includes CPF package) instead of mf2k (110513, windows only)	MF2007CFP
5	MF2005NWT			0		Use modflow-nwt (newton iteration, solves dry cells)	MODFLOW-NWT
6	MF2KASP			0		Use Doherty's MF2KASP instead of MF2k, only available on Windows not on Mac	MFASP
7	MT3DMS			0		Use MT3DMS	MT3DMS
8	SEAWAT			0		Use Seawat (version 4)	SEAWAT-V4
9	MODPATH			0		Use MODPATH (may be combined with other models)	MPATH
10	PEST			0		PEST, must be combined with other models	MPATH
11	UCODE			0		UCODE, must be combined with other models	NOT IMPLEMENTED YET
12	Package	UNIT	Extension	ON / OFF	Indictor	Description	Remark
13	LIST	11	LST	1	<====	Output list	MFLOW
14	BAS6	12	BAS	1	<====	Basic package	MFLOW
15	DIS	13	DIS	1	<====	Discretization package	MFLOW
16	UPW	14	UPW	0		Upstream weighting (must be used with NWT)	mf2005nwt
17	LPF	15	LPF	1	<====	Layer parameter package	MFLOW
18	BCF6	16	BCF	0		Block-centered flow package	MFLOW
19	HFB6	17	HFB	0		HFB package	MFLOW
20	RCH	18	RCH	1	<====	Recharge package	MFLOW
21	EVT	19	EVT	0		Evapotranspiration package	MFLOW
22	ETS	20	ETS	0		Evapotranspiration with multiple segments	MFLOW
23	WEL	21	WEL	1	<====	Well package	MFLOW
24	RIV	22	RIV	0		River package	MFLOW
25	GHB	23	GHB	0		General head boundary package	MFLOW
26	DRN	24	DRN	1	<====	Drain package	MFLOW
27	MNW1	25	MNW1	0		MNW1 package	MF2K
28	MNW2	26	MNW2	0		MNW2 package	MF05
29	CHD	27	CHD	0		CHD (time variant constant head package)	MFLOW
30	ASP	28	ASP	0		Doherty's ASP package, works only on Windows	MFASP
31	OC	29	OC	1	<====	Output control package	MFLOW
32	SIP	30	SIP	0		Strongly Implicit Procedure	MFLOW
33	SOR	31	SOR	0		Sliced Successive Over Relaxation	MFLOW
34	PCG	32	PCG	1	<====	Preconditioned conjugate gradient package	MFLOW
35	DE4	33	DE4	0		Direct solver	MFLOW
36	NWT	34	NWT	0		Newton solver	mf2005nwt
37	DATA(BINARY)	51	HDS	1	<====	unit to write heads	MFLOW
38	DATA(BINARY)	52	DDN	0		unit to write drawdown	MFLOW
39	DATA(BINARY)	53	BGT	1	<====	unit to write budgets	MFLOW
40	DATA(BINARY)	56	ZTA	0		unit to write ZTA planes for SWI	SWI
41	DATA	54	SENS	0		unit to write sensitivity data	MFLOW
42	CBF	55	CBF	0		composit budget file (MODPATH)	MPATH
43	FTL	40	FTL	0		FTL file geneted by MF2K if LMT6 is on for MT3DMS	MT3DMS
44	LMT6	41	LMT	0		Link MT3DMS package	MT3DMS
45	UMT	42	UMT	0		Unformatted flow-transport link file	MT3DMS
46	BTN	43	BTN	1	<====	Basic transport package	MT3DMS
47	ADV	44	ADV	1	<====	Advection package	MT3DMS
48	DSP	45	DSP	1	<====	Dispersion package	MT3DMS
49	SSM	46	SSM	1	<====	Source-sink mixing package	MT3DMS
50	RCT	47	RCT	0		Chemical reaction package, for sorption to be active make sure ISOTHM insheet MT3D>0	MT3DMS
51	GCG	48	GCG	1	<====	Process genearlize CGP solver package	MT3DMS
52	VDF	49	VDF	0		Variable density flow package	SEAWAT
53	VSC	50	VSC	0		Viscosity in SEAWAT	SEAWAT
54	SWI	51	SWI	0		Saltwater intrusion package	SWI
55	PES	52	PES	0		Parameter estimation package	PEST
56	SEN	53	SEN	0		Sensitivity package	PEST
57	OBS	54	OBS	0		Observation package	MFLOW
58	CFP	55	CFP	0		Conduit flow package (110513)	CFP
59	COC	56	COC	0		Conduit flow output package (110513)	CFP
60	CRCH	57	CRC	0		Conduit flow recharge package (110513)	CFP

Figure 18: NAM worksheet



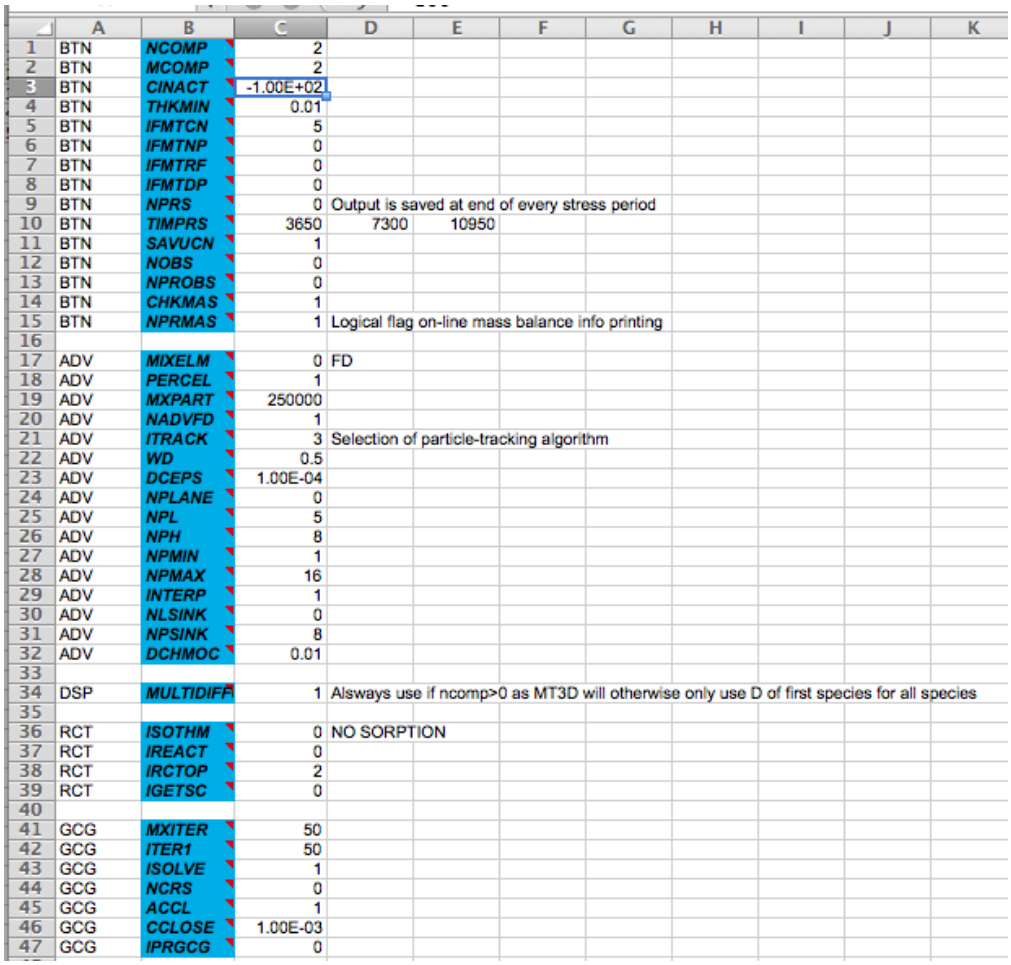
	A	B	C	D
1	OC	IHEDFM	12	Head printing format code
2	OC	IDDNFM	12	Drawdown printing format code
3	OC	IHEDUN	51	HEAD output file unit number (must correspond with unit in NAM sheet)
4	OC	IDDNUN	52	Drawdown output file unit number must correspond with unit in NAM sheett)
5				
6	UPW	IUPWCB	53	Output file unit number for cell by cell flow terms corresponding to package (also see ICBCFL in PER sheet)
7	LPF	ILPFCB	53	Output file unit number for cell by cell flow terms corresponding to package (also see ICBCFL in PER sheet)
8	BCF	IBCFCB	53	Output file unit number for cell by cell flow terms corresponding to package (also see ICBCFL in PER sheet)
9	RIV	IRIVCB	53	Output file unit number for cell by cell flow terms corresponding to package (also see ICBCFL in PER sheet)
10	DRN	IDRNCB	53	Output file unit number for cell by cell flow terms corresponding to package (also see ICBCFL in PER sheet)
11	GHB	IGHBCB	53	Output file unit number for cell by cell flow terms corresponding to package (also see ICBCFL in PER sheet)
12	CHD	ICHDCB	53	Output file unit number for cell by cell flow terms corresponding to package (also see ICBCFL in PER sheet)
13	WEL	IWELCB	53	Output file unit number for cell by cell flow terms corresponding to package (also see ICBCFL in PER sheet)
14	EVT	IEVTCB	53	Output file unit number for cell by cell flow terms corresponding to package (also see ICBCFL in PER sheet)
15	ETS	IETSCB	53	Output file unit number for cell by cell flow terms corresponding to package (also see ICBCFL in PER sheet)
16	RCH	IRCHCB	53	Output file unit number for cell by cell flow terms corresponding to package (also see ICBCFL in PER sheet)
17	MNW2	MNWPRNT	2	level of detail about MNW2 printed to the listing file (0-2)
18	MNW1	IWELPT	0	level of detail about MNW1 printed to the listing file (0 OR ~0)
19	MNW1	LOSSTYPE	2	1=skin, 2=linear, 3=non-linear
20	MNW1	WELL1	1	Write well info to file WELL1 basename.WMN1_WELL1
21	MNW1	BYNODE	1	Wrtie well info to file basename.WMN1_BYNODE
22	MNW1	QSUM	1	Write well info to file file basename.MNW1_QSUM
23				
24	DIS	ITMUNI	4	`0=undef,1 sec,2 min, 3 hours, 4 days, 5 years
25	DIS	LENUNI	2	1=feet, 2= m, 3=cm
26				
27	BAS	HNOFLO	1.00E+30	Value of heads to indicate no flow cells
28				
29	LPF7	STORAGECOEFFICIENT	0	MF2005 option
30	LPF7	CONSTANTCV	1	MF2005 option
31	LPF7	THICKSTRT	1	MF2005 option
32	LPF7	NOCVCORRECTION	1	MF2005 option
33	BCF/LPF	HDRY	1.00E+20	Head indicating dry cells
34	UPW	IPHDRY	0	Determines whether cells will be set to dry when head is < 1e-4 above cell bottom (units see LENUNI). IF 0 no
35	BCF	IWDFLG	1	Wetting capability active if >0
36	BCF/LPF	WETFCT	1	Wetting factor in equation 33A, see Modflow 2000 manual.
37	BCF/LPF	IWETIT	10	Wetting outer iteration interval insolver
38	BCF/LPF	IHDWET	1	Use wetting equation 33A if zero, otherwise use equation 33B
39	LPF	NPLPF	0	Number of LPF parameters (Currently must be 0)
40				
41	RCH	NRCHOP	3	Recharge flag: 1=in toplayer, 2=specify in which layer, 3=in highest active layer
42	EVT	NEVTOP	3	EVT option same as NRCHOP
43				
44				
45	PCG/DE4/I	HCLOSE	1.00E-04	Head change criterion for convergence [L]
46	DE4/SIP/S	ACCL	1	
47	PCG/SIP/S	MXITER	200	Maximum number of outer iterations
48	PCG	ITER1	200	Maximum number of inner iterations
49	PCG	NPCOND	1	Flag used to set the matrix conditioning method
50	PCG	RCLOSE	1.00E-04	Residual criterion for convergence [L3/T]
51	PCG	RELAX	1	Relaxation parameter used with NCOND==1
52	PCG	NBPOL	0	Used with NPCOND==2.
53	PCG	IPRPCG	5	Printout interval for PCG. If zero it is set to 999.
54	PCG	MUTPCG	1	Flag that controls printing of convergence information from the solver
55	PCG	DAMP	1	Damping factor. 1=no damping. Must be 0<DAMP<=1.
56				
57	DE4	ITMX	1	
58	DE4	MXUP	0	
59	DE4	MXLOW	0	
60	DE4	MXBW	0	
61	DE4	IFREQ	1	
62	DE4	MUTD4	2	
63	DE4	IPRD4	1	
64				
65	SIP	NPARM	5	
66	SIP	IPCALC	1	
67	SIP	WSEED	0	
68	SIP	IPRSIP	10	
69	SOR	IPRSOR	10	
70				
71	SWI	ISTRAT	1	
72	SWI	NPLN	1	Number of surface planes = number of zones minus one
73	SWI	ISWIZT	56	Unit number for binary output file of interface elevations
74	SWI	NPRN	100	Output timestep frequency (within stress period)
75	SWI	TOESLOPE	0.2	
76	SWI	TIPSLOPE	0.2	
77	SWI	ZETAMIN	0.05	

NAM | MFLOW | MT3D | SEAWAT | MPATH | PER | LAY | Config

Figure 19: Part of the MFLOW worksheet

	A	B	C	D	E	F	G	H	I	J
54	PCG	MUTPCG	1	Flag that controls printing of convergence information from the solver						
55	PCG	DAMP	1	Damping factor. 1=no damping. Must be 0<DAMP<=1.						
56										
57	DE4	ITMX	1							
58	DE4	MXUP	0							
59	DE4	MXLOW	0							
60	DE4	MXBW	0							
61	DE4	IFREQ	1							
62	DE4	MUTD4	2							
63	DE4	IPRD4	1							
64										
65	SIP	NPARM	5							
66	SIP	IPCALC	1							
67	SIP	WSEED	0							
68	SIP	IPRSIP	10							
69	SOR	IPRSOR	10							
70										
71	SWI	ISTRAT	1							
72	SWI	NPLN	1	Number of surface planes = number of zones minus one						
73	SWI	ISWIZT	56	Unit number for binary output file of interface elevations						
74	SWI	NPRN	100	Output timestep frequency (within stress period)						
75	SWI	TOESLOPE	0.2							
76	SWI	TIPSLOPE	0.2							
77	SWI	ZETAMIN	0.05							
78	SWI	DELZETA	0.05							
79	SWI	NU	0	0.025						
80										
81	NWT	HEADTOL	1.00E-04	units of length)—is the maximum head change between outer iterations for solution of the no						
82	NWT	FLUXTOL	1.00E-01	units of length cubed per time)—is the maximum root-mean-squared flux difference between						
83	NWT	MAXITEROUT	100	is the maximum number of iterations to be allowed for solution of the outer (nonlinear) probl						
84	NWT	THICKFACT	1.00E-06	is the portion of the cell thickness (length) used for smoothly adjusting storage and conductar						
85	NWT	LINMETH	1	is a flag that determines which matrix solver will be used. A value of 1 indicates GMRES wil						
86	NWT	IPRNWT	0	is a flag that indicates whether additional information about solver convergence will be print						
87	NWT	IBOTAV	0	is a flag that indicates whether corrections will be made to groundwater head relative to the c						
88	NWT	OPTIONS	2	are keywords that activate options: SPECIFIED read further options. SIMPLE any linear mo						
89										
90	NWT	DBDTHETA	0.7	ignored						
91	NWT	DBDKAPPA	0.0001	ignored						
92	NWT	DBDGAMMA	0	ignored						
93	NWT	MOMFACT	0.1	ignored						
94	NWT	BACKFLAG	0	ignored						
95	NWT	MAXBACKITER	20	ignored						
96	NWT	BACKTOL	1.5	ignored						
97	NWT	BACKREDUCE	0.75	ignored						
98										
99	NWT	MAXITINNER	100	ignored						
100	NWT	ILUMETHOD	2	ignored						
101	NWT	LEVFILL	1	ignored						
102	NWT	STOPTOL	1.00E-10	ignored						
103	NWT	MSDR	10	ignored						
104										
105	NWT	IACL	2	ignored						
106	NWT	NORDER	1	ignored						
107	NWT	LEVEL	1	ignored						
108	NWT	NORTH	2	ignored						
109	NWT	IREDSYS	1	ignored						
110	NWT	RRCTOLS	0	ignored						
111	NWT	IDROPTOL	1	ignored						
112	NWT	EPSRN	0.001	ignored						
113	NWT	HCLOSEXMD	0.0001	ignored						
114	NWT	MXITERXMD	50	ignored						
115										

Figure 20: MFLOW-worksheet continued



	A	B	C	D	E	F	G	H	I	J	K
1	BTN	NCOMP	2								
2	BTN	MCOMP	2								
3	BTN	CINACT	-1.00E+02								
4	BTN	THKMIN	0.01								
5	BTN	IFMTCN	5								
6	BTN	IFMTNP	0								
7	BTN	IFMTRF	0								
8	BTN	IFMTDP	0								
9	BTN	NPRS	0	Output is saved at end of every stress period							
10	BTN	TIMPRS	3650	7300	10950						
11	BTN	SAVUCN	1								
12	BTN	NOBS	0								
13	BTN	NPROBS	0								
14	BTN	CHKMAS	1								
15	BTN	NPRMAS	1	Logical flag on-line mass balance info printing							
16											
17	ADV	MIXELM	0	FD							
18	ADV	PERCEL	1								
19	ADV	MXPART	250000								
20	ADV	NADVFD	1								
21	ADV	ITRACK	3	Selection of particle-tracking algorithm							
22	ADV	WD	0.5								
23	ADV	DCEPS	1.00E-04								
24	ADV	NPLANE	0								
25	ADV	NPL	5								
26	ADV	NPH	8								
27	ADV	NPMIN	1								
28	ADV	NPMAX	16								
29	ADV	INTERP	1								
30	ADV	NLSINK	0								
31	ADV	NPSINK	8								
32	ADV	DCHMOC	0.01								
33											
34	DSP	MULTIDIFF	1	Alsways use if ncomp>0 as MT3D will otherwise only use D of first species for all species							
35											
36	RCT	ISOTHM	0	NO SORPTION							
37	RCT	IREACT	0								
38	RCT	IRCTOP	2								
39	RCT	IGETSC	0								
40											
41	GCG	MXITER	50								
42	GCG	ITER1	50								
43	GCG	ISOLVE	1								
44	GCG	NCRS	0								
45	GCG	ACCL	1								
46	GCG	CCLOSE	1.00E-03								
47	GCG	IPRGCG	0								

Figure 21: MT3D worksheet for one of the MT3DMS examples

22 SEAWAT worksheet

The worksheet with the name SEAWAT contains the parameters for the Variable Density Flow package (VDF) and the Viscosity Package (VSC) which SEAWAT version 4, (*swt_v4*) implements. Refer to the SEAWAT manual for a description. To easily scope with more species, we need more than one numerical value for some of the parameters (see figure below).

23 PER worksheet, to specify stress periods, including output control OC)

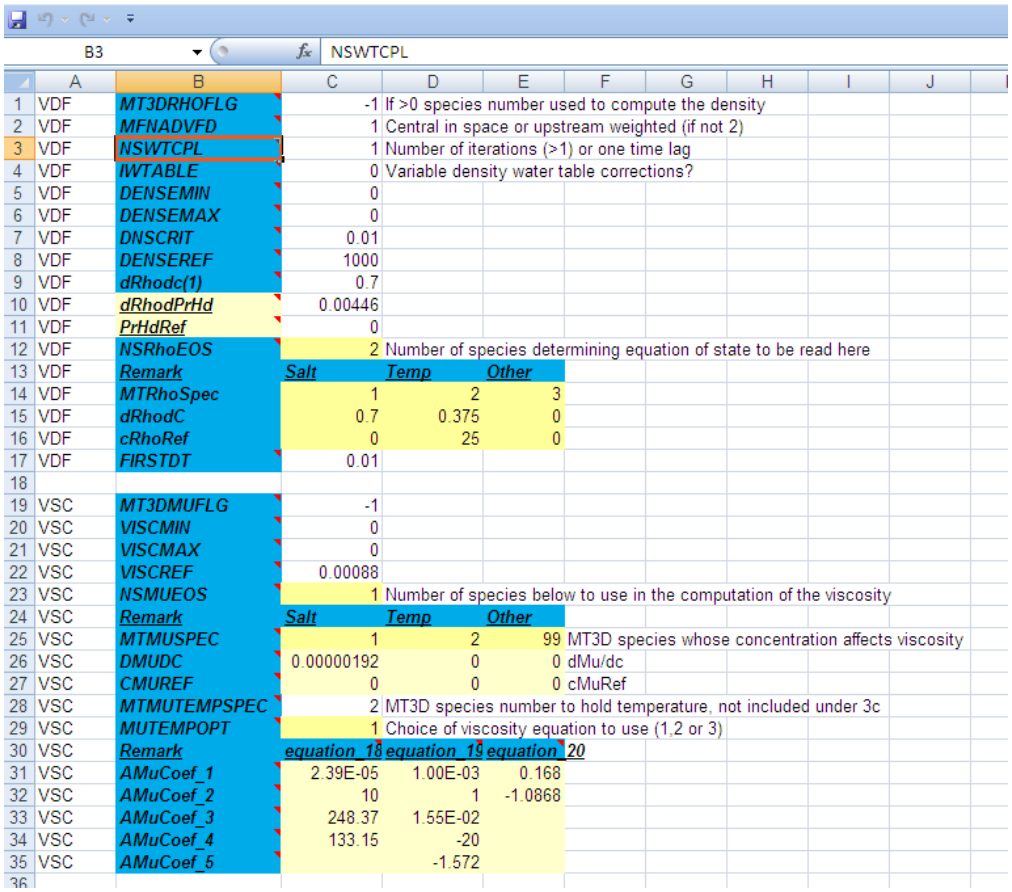
The worksheet PER defines the stress periods and their parameters. Like the LAY worksheet to be discussed hereafter, the PER sheet is arranged horizontally, that is, with one stress period defined per line. The top line of the sheet holds the name of the package (or packages) that require the parameter on the line below, in the second line. Note that a comment is asscociated with each parameter that explains its meaning. It is a piece of text from the MODFLOW manual, sometimes a parameters has additial meaning for mfLab. For instance, IHDDFL is interpreted as the time step frequency at wihch head or drawdown output is to be generated, where the counter is reset at the beginning af each stress period. So setting IHDDFL to a number larger than NSTP guarantees output of heads and or drawdown after complete stress periods only.

Line 3 and further lines contain the stress-period parameter values.
Each subsequent line defines a (set of) stress period(s).
mfLab allows much flexibility in defining these stress periods using the following rules:

1. NPER is the stress period number, or rather the last of a series of stress periods.
2. The highest specified stress-period number in the worksheet is taken the number of stress periods to be used by the model in the simulation.
3. Stress periods are sorted by *mfLab* so their order in the worksheet does not matter.
4. Lines with stress period numbers <1 are ignored. This allows to switch off stress periods easily.
5. Missing stress periods are filled in by *mfLab* prior to generating the input files for the programs to be run subsequentluy. Filling in is done backwar. Hence each gap in stress periods number implies that the missing stress periods get the value of the first defined stress period after the gap. Hence if only a single line is given, then the stress period number of that line defines the total number of stress periods, while all of them will have the same values.
6. If stress periods are defined more than once (i.e. several lines have the same stress period number), then the first of them wins (as a consequence of filling in the missing periods backward).

Hence, if stress periods 5 15 15 and 100 are specified, where 15 stress period number 15 is a duplicate, then stress periods 1-5 are equal to defined stress period 5, further, stress periods 6-15 will be equal to the first of the two encountered stress periods line with period number 15, and stress periods 16 to 100 will have the values defined for stress period 100. And the number of stress periods in the model will be equal to the highest stress period number encountered on the PER worksheet, which is 100 in this hypothetical example.

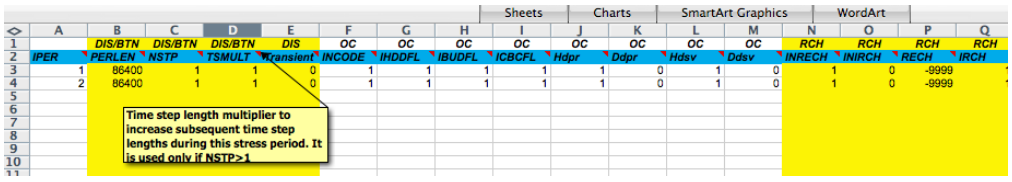
Output control (OC-package): The PER sheet also contains the parameters for output control. This contrasts with the specifications in the MODFLOW manual, which requires that output control be given per TIME STEP step and not per STRESS PERIOD. However, for practical reasons, namely to prevent the clutter and confusion of yet another worksheet with a separate (potensially very long) list which has to be managed separately, as well as to prevent the often very large number of lines required for output control, namely the number of stress periods times number of time steps within the stress periods, there is no separate OC worksheet in *mfLab*. Instead, the option of the parameters IHDDFLG, IBUDFLG and ICBCFLG have been extended in *mfLab* relative to the options provided by MODFLOW. In *mfLab* a value <> 0 means that output regulated by the particular flag will be produced at that time-step interval within each stress period.



B3 NSWTCPL

	A	B	C	D	E	F	G	H	I	J
1	VDF	MT3DRHOFLG	-1	If >0 species number used to compute the density						
2	VDF	MFNADVFD	1	Central in space or upstream weighted (if not 2)						
3	VDF	NSWTCPL	1	Number of iterations (>1) or one time lag						
4	VDF	IWTABLE	0	Variable density water table corrections?						
5	VDF	DENSEMIN	0							
6	VDF	DENSEMAX	0							
7	VDF	DNSCRIT	0.01							
8	VDF	DENSEREF	1000							
9	VDF	dRhodc(1)	0.7							
10	VDF	dRhodPrHd	0.00446							
11	VDF	PrHdRef	0							
12	VDF	NSRhoEOS	2	Number of species determining equation of state to be read here						
13	VDF	Remark	Salt	Temp	Other					
14	VDF	MTRhoSpec	1	2	3					
15	VDF	dRhodC	0.7	0.375	0					
16	VDF	cRhoRef	0	25	0					
17	VDF	FIRSTDT	0.01							
18										
19	VSC	MT3DMUFLG	-1							
20	VSC	VISCMIN	0							
21	VSC	VISCMAX	0							
22	VSC	VISCREF	0.00088							
23	VSC	NSMUEOS	1	Number of species below to use in the computation of the viscosity						
24	VSC	Remark	Salt	Temp	Other					
25	VSC	MTMUSPEC	1	2	99	MT3D species whose concentration affects viscosity				
26	VSC	DMUDC	0.00000192	0	0	dMu/dc				
27	VSC	CMUREF	0	0	0	cMuRef				
28	VSC	MTMUTEMPSPEC	2	MT3D species number to hold temperature, not included under 3c						
29	VSC	MUTEMPOPT	1	Choice of viscosity equation to use (1,2 or 3)						
30	VSC	Remark	equation_18	equation_19	equation_20					
31	VSC	AMuCoef_1	2.39E-05	1.00E-03	0.168					
32	VSC	AMuCoef_2	10	1	-1.0868					
33	VSC	AMuCoef_3	248.37	1.55E-02						
34	VSC	AMuCoef_4	133.15	-20						
35	VSC	AMuCoef_5		-1.572						
36										

Figure 22: SEAWAT worksheet in the Excel workbook of the Coast example for SEAWAT



Sheets Charts SmartArt Graphics WordArt

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1		DIS/BTN	DIS/BTN	DIS/BTN	DIS	OC	OC	OC	OC	OC	OC	OC	OC	RCH	RCH	RCH	RCH
2	IPER	PERLEN	NSTP	TSMULT	Transient	INCODE	IHDDFL	IBUDFL	ICBCFL	Hdpr	Ddpr	Hdsv	Ddsv	INRECH	INIRCH	RECH	IRCH
3	1	86400	1	1	0	1	1	1	1	1	0	1	0	1	0	-9999	
4	2	86400	1	1	0	1	1	1	1	1	0	1	0	1	0	-9999	

Time step length multiplier to increase subsequent time step lengths during this stress period. It is used only if NSTP>1

Figure 23: part of the PER worksheet, also showing a comment explaining the meaning of the variable TSMULT

This interval is taken as the absolute value of the particular parameter. *mfLab* makes use that is, there will always be output at the end of every stress period, but within a stress period output is produced only at the given frequency. So, to make sure you only have output at the end of each stress period (unless the flag is given value 0 indicating no output for the current stress period). Hene, to just obtain output at the end of each stress period use a large value for these parameters, larger than the number of time steps in the stress period. Use the value 1 for these two parameters if you want heads and budget output at every time step. With these data and rules *mfLab* generates the OC-control file, which will have tow lines for every time step.

Semantic extensions have also been implemented for all parameters starting with "IN". These parameters indicated whether a layer of values the respective parameter must be read for this stress period or whether the values of the previous stress period will be used. Hence INSURF thus refers to SURF, INEVTR to EVTR, INEXPD to EXPD and INIEVT to IEVT and likewise for the paremeters pertaining to the RCH package i.e INRECH to RECH and INIRCH to IRCH. These parameters have the following meaning:

if<0 the values of the concerned parameter of the previous period will be reused and nothing is read

if=0 the values will be obtained from the MATLAB workspace and the concerned parameter, i.e. SURF, EVTR, EXPD ,IEVT, RECH and or IRCH must be provided in the Matlab workspace. See below.

if>0 the value of the corresponding parameter in the worksheet LAY will be used for this stress period for all cells.

To provide one of the parameters SURF etc in the matlab workspace do the following. You must provide either a cell array with one cell per stress period or a 3D array with one array layer per stress period. This must be a 3D array. The length of the cell array must be at least as long as the highest stress period number that require its values. The same is true for the length of the 3rd dimension if a 3D array is used instead. The cell array has the advantage that those cells can be left empty that correspond to stress periods for which no values from the workspace are required, i.e. for those periods for which the corresponding "IN-parameter" is non-zero (that is previous values are reused or value is obtained from the Excel worksheet). This is because mfLab will only look at the cells corresponding to the layers for which the IN-parameter=0. In case a 3D array is preferred then ony the array layers corresponding with stress periods for which the IN-parameter is zero are used. The other values are immaterial. It is a good habit to use NaNs, as errors will immediately appear.

If a cell has only one value, *mfLab* assigns it to all cells af if a layer of values were given. The same is true if a 3D array is used. If the first and second dimensions are 1, *mfLab,* considers the value as a layer value. However, the array must be 3D for *mfLab* to decide which values in the array correspond to each stress period. You may trun any array in a 3-D one using the reshape or permute functions in Matlab. For instance if a is A vector then

A=reshape(A,1,1,length(A)) or A=permute(A,[3,2,1])
will do the trick.

Filling in a cell array may be done as follows

A=cell(NPER,1); % note the 1, as otherwise the array will be NPERxNPER instead of NPERx1

for iPer=1:NPER, A{i}=your values for this stress period; end

–

The EVT package requires the specification of SURF and EXPD, i.e. the elevation (according to the applied datum) above which the EVT equals the maximal value, specified as EVTR, and the depth below this surface at which evapotranspiration ceases. To make the EVT package work, SURF and EXPD must at least be specified for the first stress period, after which the can be reused using the value -1 for INSURF and INEXPD. To facilitate specifying the stress periods in *mfLab*, it will interpret the value in the SURF variable for the first stress period even if INSURF is -1 (the reuse option). The same is implemented for the EXPD value. If the INEXPD is -1 in the first stress period it will set EXPD to the value specified in the column EXPD. Defining the semantics this way, allows using only one stress period line in the worksheet to specify any number of stress periods (by setting the IPER variable in the first column to the desired number of stress periods.

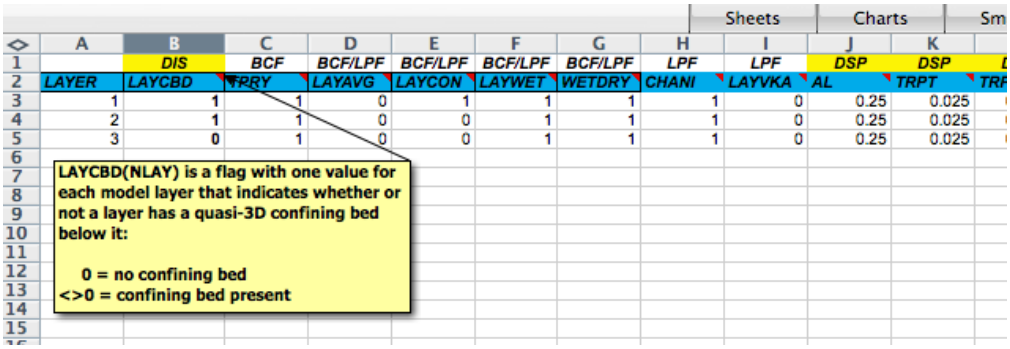
For further convenience

if INSURF(1)=-1 then SURF(1) is SURF elevation

if INSURF(1)=-2, then SURF(1) is distance of SURF above top of model (use negative value for distance below top of model)

24 LAY worksheet to specify layer information

The worksheet LAY is also oriented horizontally, like the worksheet PER. Worksheet LAY contains the parameters pertaining to the layers in the model which are specified on a per layer basis. The structure is the same as that of the PER sheet, but, of course, in this sheet each row gives the information of a specific layer.



	A	B	C	D	E	F	G	H	I	J	K
1		DIS	BCF	BCF/LPF	BCF/LPF	BCF/LPF	BCF/LPF	LPF	LPF	DSP	DSP
2	LAYER	LAYCBD	TRPY	LAYAVG	LAYCON	LAYWET	WETDRY	CHANI	LAYVKA	AL	TRPT
3	1	1	1	0	1	1	1	1	0	0.25	0.025
4	2	1	1	0	0	1	1	1	0	0.25	0.025
5	3	0	1	0	0	1	1	1	0	0.25	0.025

Figure 24: Part of the LAY worksheet, also showing the comment attached to the parameter LAYCBD

Only unique layers have to be specified. To allow this, the layer number in the first column of this sheet is obligatory. *mfLab* uses it to target given layer properties exactly. This way of specifying layers puts maximum flexibility in the hands of the user. The rules that *mfLab* implements are as follows:

1. The specified layers will be sorted prior to generating the input files of *mf++* programs. Therefore, the order in which layers are specified in this worksheet is unimportant.
2. Each specified worksheet line targets the layer according to its layer number
3. If duplicate layer numbers are encountered, the first one will be used
4. If layers with number > NLAY are specified, the first one >= NLAY will be used to start filling in the layer properties backward, i.e. starting with the highest one. Note that *mfLab* deduces the number of layers in the model from the number of layers in the IBOUND array, not from the LAY worksheet.The highest specified layer number must be greater than or equal to the number of layers in the model (=size(IBOUND,3)
5. Lines with layer numbers <1 are ignored. This allows layers to be easily switched off.
6. Missing layers will be filled in by *mfLab* backward, using the values the next higher one specified by the user.

So, in practice, if all layers are unconvertable (i.e. LAYCON=0) and they all have the same layer properties, then supplying only one layer suffices and its number must be greater than or equal to the number of layers in the model. If the first three layers are convertible, and layer 4 to 20 are the same as are layers 21 to 40, then specify only the layers 3, 20 and 40. The rest will be filled in by *mfLab* working its way back from the highest one.

Some of the parameters like WETDRY in the LAY worksheet can be specified directly in mf_adapt. This allows specifying cell-by-cell values instead of just one value for an entire layer. To invoke/use the parameter for WETDRY in the Matlab workspace, set the parameter WETDRY in the LAY worksheet equal to zero. In that case, if there exists a parameter WETDRY in the mf_adapt workspace of Matlab, then that parameter will be used. This parameter must be named WETDRY and is a cell array with one cell per layer in which

the desired values are stored as a Ny*Nx layer array. The parameter WETDRY must have at least as many layers as the highest convertible layer number (LAYCON<>0 in the LAY worksheet). The cell number must correspond to the layer numbers. If a layer has a single value that value is used for the entire layer. But only the layers are used that are both convertible, of which LAYWET<>0 and which have WETDRY==0. If WETDRY==0 but there exists no WETDRY parameter in mf_adapt, then MODFLOW's method is followed, meaning that rewetting will not take place (WETDRY=0).

25 Stresses (WEL, GHB, RIV, DRN, CHD, MNW) and point sources PNTSRC

Stresses WEL, GHB, DRN etc are data for different types of stresses or boundary conditions that are known as packages used by MODFLOW to implement specific behavior or boundaries (connections between the model and the outside world. PNTSRC (point source) is similar, but required to specify the input concentrations at the locations of these stresses. PNTSRC is read by the SSM package requried in MT3DMS and Seawat. These stresses have no special worksheets; they are defined in the workspace of Matlab. This is most easily done using function *bcnZone()*, i.e. "specify boundary conditions based zone zones and a zoneArray". BCN stands for boundary condition. It could also be called a stress. To show how the function *bcnZone*() works, type *help bcnZone*. This is the result:

```
>> help bcnZone
  gridObj/bcnZone: [BCN,PNTSRC] = bcnZone(basename,type,zoneArray,zoneVals,concVals)
  BCN means "Boundary Conditions"
      and PNTSRC means "Point source".
  bcnZone yield values for MODFLOW and MT3DMS/Seawat

  OUPUTS
     BCN = array or array of cells containing the list input for a
           particular boundary type (WEl, DRN etc). The list has the
           following items on a line
     [ stressPeriodNr  Layer Row Column values ]
           exactly as required by the MODFLOW package of the type that is
           specified, the only difference being that the stress period number
           preceeds the values that the Modflow package requires. This makes
           each input line unique, so that the order in which it is presented
           to mfLab does not matter. mfLab will sort the input accordingly
           and generate the input files for the different Modflow packages in
           the right order.
     PNTSRC = array similar to BCN, but required by MT3DMS/Seawat.
           It contains the concentrations as follows
     [ stressPeriodNr Layer Row Col Conc1 ITYPE Conc1 Conc2 Conc3 ... ]
           where ITYPE refers to a type of stress as defined in the MT3DMS
           manual and Conc1 etc to the input conc of the corresponding
           species for that cell and stress period. Note that input after
           ITYPE is only required if the simulation is done with more than
           one species.

  INPUTS
     basename = the basename of the project used for (almost) all input and
                output files
     type     =is string indicating the stress type. It must be one of
        'WEL' = well cells
        'DRN' = drain cells
        'DRT' = drain with return flow cells
        'RIV' = river or stream cells
```

```
    'GHB' = general-head boundary cells
    'CHD' = (varying) constant heads
    'MNW' = multi-node well cells
    'MLS' = mass-loading source loading cells
    'CCC' = constant-concentration cells

  zoneArray = an array of the size of the grid containing zone numbers.
      zoneArray defines the spatial distribution of zones in the grid.

  zoneVals = cell vector or cell array defining the input for the zones.
      zoneVals has one row per zone as follows:
      {zoneNr1 value value value ...;
       zoneNr2 value value value ...
       zoneNr3 value value value. }
  zoneNri = the zoneNr partaining to the value that follow. These
      values will be transferred to the input of the model and must
      therefore, match the inputs expected by the stress. For instance
      WEl expects only the well flow. CHD expects two heads. See modflow
      manual.
      regarding value:
         scalar: --> all cells with zoneNr get this value for all
                 stress periods.
         vector: with the same number of values as there are in the
                 zone zoneNr in the model --> each cell gets the value
                 pertaining to it, while the cells are the same for all
                 stress periods.
         string: the header in worksheet 'PER' in workbook [basename '.XLS]
                 This yields one value for each stress period. This
                 value will be the same for all cells in the zone.
                 A different value for each zone and each
                 stress period, requires a loop.

   concVals: conc conc conc with one "value" per species.
         conc can be a single value per species, or a vector with
         the number of values equal to the number of cells in the
         zone, or it is a string referring to the header of a column
         in the PER sheet with one value per stress period.
         In case we need a different value per cell and per stress
         period, we have to use a loop.

IMPORTANT: it is your responsibility to provide a
concentration for each component/species. mfLab cannot see of
you overrule the value NCOMP in sheet MT3D. It will,
therefore just add any species concentrations to PNTSRC as
provided in concVals (=size(concVals,2)). This has no consequences
unless size(concVals,2) is less than the number of species
you are using, i.e. less than the actual NCOMP.

TO 120411
```

26 wells and multinode wells

The models can have wells and multi-node wells. They can be specified through the stresses as indicated above. They can also be specified using an object oriented approach as decribed below. The object-oriented approach is described below.

Multinode wells come in two flavors:

MNW1 the original version by Halford KJ, Hanson RT (2002) User Guide for the Drawdown-Limited, Multi-Node Well (MNW) Package for the US Geological Survey's Modular Three-Dimensional Finite-Difference Ground-Water Flow Model, Versions MODFLOW-96 and MODFLOW-2000. UGSG Open-File report 02-293.

MNW2 the new version by Konikow LF, Hornberger GZ, Halford KJ, Hanson RT and Harbaugh AW (2009) "Revised Multi-Node Well (MNW2) Package for MODFLOW Ground-Water Flow Model. USGS, Techniques and Methods 6-A30.

MNW1 is contained in the last update of mf2k, after which no further changes to this code has been or will ever be made. The MODFLOW version, MF2005, takes all future innovations. Nevertheless, many still use some previous version of mf2k or use MT3MDS or Seawat, all of which do not have this new MNW2 package built in. Therefore, we cannot do away with the MNW1 package.

mfLab has implemented MNW1 in such a way that it perfectly mixes with the WEL package. The most complete example of the workbook for any model is *<<parameters.xls>>* under *mflab/examples*. You can always use that workbook if you need an update of some of the worksheets in any of the workbooks of your own modelse without changing any of your other parameters. It is a matter of copying the worksheets from the file parameters.xls to your own workbook and adapt what is ncessary for the problem at hand.

MNW require many more parameters than ordinary wells. Because its parameters supersed what is required for an ordinary well, one can use the MNW1 or MNW2 worksheets also as input to generate ordinary wells. For instance switching MNW1 off in the NAM worksheet and switching the WEL package on is enough to switch MNW1 wells into ordinary wells. You can find examples in *mflab/examples/mf2k/MNW1* or *mflab/examples/mf2k/MNW1-Reilly*, that implement examples from the original manual.

To specify wells, insert a new worksheet with a suitable name like "wells" or "mnw1" and specify the required informaiton, one line per well, with headers at the top line so that mfLab can use the correct columns. Wells require at least columns *'nr'*, *'x' 'y' 'z1' 'z2' 'rw'*. Well objects can then be generated by calling *wellObj* like this

```
help wellObj
 well= wellObj(basename, WsheetNm|well [,gr,HK [,QsheetNm [,CsheetNm],active]);
 wellObj -- definition of class wellObj by calling
 well = wellObj(basename,wellSheetNm) where
  basename is the workbook holding wellSheetNm
  wellSheetNm is the worksheet with a table defining the wells. See
  constructor by typing
 help wellObj.wellObj.
 wellObj defines wells for use in groundwater models. The objects hold
 property values and share methods that facilitate using wells.
 Some methods need a grid object. But a grid object is not necessary
 to create well objects. This is because upon creation, well objects
 only contain their pertinent data. Later on, when a grid object is
 available they may be put into the grid, upon which additional grid
 values are added to the wells, so that each well then also holds its
 position in the grid. See the methods of wellObj and of gridObj for
 more information.
```

MNW1 wells (i.e. MNW1 objects) are generated the same way as wellObj by calling the MNW1 constructor.

```
MNW1 = MNW1Obj(basename, WsheetNm|MNW1 [,gr,HK [,QsheetNm [,CsheetNm],active]);
```

Any properties contained i the worksheet Wsheet Nm land in the UserData of the MNW1 objects. The same happens with the data in the table when generating ordinary wells. The difference is that the ordinary wells generally do not use this information, while the MNW1Obj do use this info. Still the user can use any such information if he/she so needs.

Of course, for the wells to work you need to have the WEL package switched on in the NAM worksheet and for the MNW1 package to work you must have switched the MNW1 package on in the NAM worksheet.

MNW2 is not yet completely implemented, mainly because it is not supported in Seawat.

26.0.1 Bug report/Issue MNW1

As it seems MNW1 must have at least 2 mnw wells, or otherwise the last stress period in the Budget file will miss the FLOWLOWERFACE, FLOWRIGHTFACE, and MNW arrays. Workaround: Use at least 2 MNW1 wells and set the flow of the second one equal to zero. Below is seen that happens. First the budget file was read using

B= readBud([basename '.BGT']);

Then type

B(end-1).label

B(end).label

to get this

```
ans =

    'CONSTANTHEAD'    'FLOWLOWERFACE'    'FLOWRIGHTFACE'    'MNW'    'RECHARGE'    'STORAGE'

ans =

    'CONSTANTHEAD'    []    []    []    []    'STORAGE'
```

This shows the missing field in the budget of the last stress period, which is a bug in MNW1.

mfLab functions (Matlab code)

27 Generalities

mfLab functions are Matlab m-files and sometimes Matlab scripts, which, together, comprise *mfLab*. These so-called m-files (mfiles) for short, have been arranged in directories under *mfLab/mfiles* in a more or less logical way to prevent too much clutter. The different mfile directories contain functions that are useful for model generation and for the interpretation and visualization of model output. Some help with the handling of coordinates and computational grids; other functions may be more generally applicable, for instance to grab a background figure from GoogleMaps.

What follows is a generic description of the contents of the m-files directories. The description cannot be complete as new mfiles are added and some may change more or less frequently, mostly to add functionality and, sometimes of course, to remove bugs. The best way to explore what is in the directories it to have a look. In Matlab you may navigate to each directory and then use the help of the particular function

```
help some_mfile
```

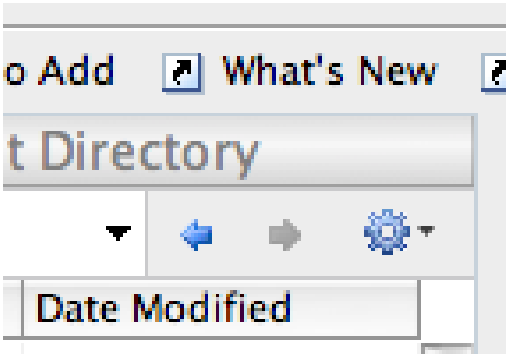
where, of course "some_mfile" is to be replaced by the file to inspect.

This will provide basic help.

Also, the "Current Directory" window Matlab's desktop provides this help at the bottom.

The best way to get an overview is to use the file *Contents.m* in each of the mfiles subdirectories. These have been generated by Matlab by gathering the help information of all the mfiles in the directory.

You can generate and update this type of information youself from inside matlab using the toothed wheel above the "current-directory" window.



28 Description of the individual m-files directories

The mfiles/write directory contains *mf_ setup.m* and all mfiles that generate the input files for the target models (i.e. MODFLOW, MT3DMS and SEAWAT for the time being). Each time a new package for one of these models is to be implemented in *mfLab*, a new write function must added for the new package together with a new accompanying block in *mf_*setup.m file. mf_setup gathers and assembles the parameters that are required by the particular write function, which writes the input file required by the package t disk.

The philosophy of reusing code by keeping code similar has been applied in *mfLab* as much as possible. This way, the input files for the WEL, GHB, DRN, RIV and CHD packages are generated by the same routine, *writeBCN.m* (BCN stands for Boundary Condition). Write functions for other packages like r*echarge* and *evapotranspiration,* have been kept as similar as possible as well.

Perhaps worthwile noting that there is a single mfile *waray.m* that does almost all of the array actual writing. It is named after *rarray,* the array reading function used by MT3DMS. It writes both the arrays to the input files of the MODFLOW and the MT3DMS packages, regardless of whether the arrays contain integers or floating point numbers, or whether the arrays are written as a list ore as a full 2D layers, and regardless of whether the array should be written with a or without an array header; *warray* does it all in a simple and straightforward manner. This guarantees consistency across all generated input files.

Writing by *mfLab* is generally done in fixed format. This may seem counter-intuitive, but, as a matter of fact, it proves to be very robust and it is backward compatible. The robustness stems from the experience that with fixed formats odds that things accidentally go "right" are very small. Consistency is necessary to find bugs with most certainty. A disadvantage with using fixed formats is that the orignal MODFLOW developers allowed too little space for many variables. While floating point numbers require least 12 spaces to guarantee sufficient accuracy under all circumstances, many variables in MODFLOW and MT3DMS fixed format are granted only 10 spaces.. Hopefully this will change in the future.

What follows is a description of some (not all) of the mfiles that come with *mfLab.* The main idea is that these descriptions serve to make the user aware of their existence and usefulness. Clearly, all functions have standard help on board, which implies that typing "help <function name"> in the command line of Matlab produces the help of that particular function (if the matlab paths have been set).

We will discuss the mfiles directory by directory.

28.1 mfiles/gridcoords: strmatchi, cellIindex, cellIndces, rd2wgs, wgs2rd

```
[E.N]=rd2wgs(xrd,yrd)
```

translates "Rijksdriehoekscoordinaten" (Dutch National Coordinates) to wgs world global system 1984 coordinates used by Google Earth. I obtained the routine from a colleague of TNO in Utrecht. I translated it to Matlab code and tested it.

```
[xrd,yrd]=wgs2rd(E,N)
```

translates wgs84 coordinates to rd coordinates. As it is impossible to deduce from the rd2wgs code how to invert the function, I implemented it as an optimization problem in which the *wgs* coordinates are known and the *rd2wgs* function is available.

```
idx=strmatchi(name,names,opt)
```

is used to find the labels read from the worksheets and get the corresponding column or row index to then grab the correct numerical values. It has some extra flexibility beyond Matlab's own *strmatch*. The arbument opt is used to prevent an error message when *strmatchi* fails.

```
I=cellIndex(ix,iy,Nx,Ny)
```

Compute global index of 2D array (same as Matlab's *sub2ind*)

```
I=cellIndex(ix,iy,iz,Nx,Ny,Nz)
```

Compute global index of 3D array (same as Matlab's *sub2ind*)

```
LRC=cellIndices(I,dims,'LRC')
```

yields the individual COL ROW and LAY indices of a an array whose dimensions are known and a list of global array indices are given. The latter are obtained using the find function and some logical condition, like I=find(IBOUND==-1), then LRC=cellIndices(find(IBOUND==-1), size(IBOUND),'LRC') immediately yield the corresponding list of Layer Row Col indices that are required to specify boundary conditions WEL, DRN, RIV, GHB and CHD. A little more advanced than Matlab's *ind2sub*.

```
[X,Y,well]=make_grid(well,aroundAll,aroundEach,dmax,dmin,factor)
```

Generates a finite difference grid taking into account the location of wells in struct *well* desiring a local, more detailed grid, a desired distance to the model boundary, a maximum and minimum cell size, and a factor by which the local grid around each well increases. After generating the grid, it is cleaned up such that no cell will be smaller than the prescribed argument dmin.

28.2 mfiles/read, readDat, readBud, readMT3D

This directory contains the functions used by *mfLab* to read files. The functions always used in any *mf_analyze* are *readDat*, *readBud* and *readMT3D* because they can read the unformatted files produced by MODFLOW (budget, heads, draw-downs) and by MT3DMS (concentrations). These functions are similar in their use and also in the options they offer to pick out precisely the data that are required for the interpretation and visualization. Some examples of simple and advanced use have been given earlier in section 16.

```
H=readDat(fname[,periods[,tsteps[,lays[,rows[,cols]]]]]);
```

reads heads, draw-down or SWI zeta file into convenient struct

```
B=readBud(fname[,labels[,periods[,tsteps[,lays[,rows[,cols]]]]]]);
```

reads MODFLOW's budget file into convenient struct

```
C=readMT3D(fname[trpstps[,lays[,rows[,cols]]]]);
```

reads MT3DMS concentration file into convenient struct

The directory further contains all the functions necessary to read input files for MODFLOW, MT3DMS, SEAWAT and MOCDENSE. These files are used to read in a complete existing model as defined through it's input files into the Matlab workspace. It is therefore possible to read in any existing model, regardless of source into the workspace and to start modeling from it.

28.3 Initial concentrations and heads from previouw output

It is straightforward to use heads and or concentrations from previous simulations as initial vlaues in a new one. This is is done using the functions readDat and or readMT3D described above. If you know which combination of stress period and or time step you are interested in to use as intiial heads, a way to do this works as follows

```
H=readDat(fname,iPer,iStp);
STRTHD=H.values;
```

which yields the 3D matlab array with the corresponding values.

Similarly obtaining initial concentrations, would be done the some way

```
C=readMT3D(fname,,iPer,iStp)
STCONC=C.values;
```

reading the particular heads and concenratios by qualifying the stress peirod and time step prevents reading in the entire head or concentration file, thus saving time and memory space.

The functions mf_getHds(mfile,time) and mf_setConc(mfile,time) get the heads or concentrations at a given time, or at least the heads or concentrations as close as possible before the time specified, that is for time $t \leq t_{specified}$. Using the less-equal conditions is advisable to prevent round off mismatches causing the coputer to considere two real numbers to differ, while, in reality they are (meant to be) equal. However, this function exploits the accuracy of the specified time to compare the simulation times in order not to miss data due to roundoff.

28.4 mfiles/write

This directory contains all routines that generate input files for target models. It also contains *mf_run.m*, which is the backbone of *mfLab* from which all write routines are invoked.

28.5 mfiles/etc

Various useful functions.

```
avicompress(basename);
```

Compersses avi files tremendously on macs. It was necessary because Matlab does not provide for compression. However, it uses a UNIX command *mencoder*, which is a free software package that comes together with *mplayer*. On Macs it is easy to install because the Mac has full Uni underlying it. Instructions can be found on the internet. In just a couple of seconds, your GB avifile is reduced to 3 or so MB and can be played on any viewer even the most recent version of Quicktime.

```
cr=ContourRange(B,db[,[Lay][,fieldname]);
```

Computes a nice range of values for countouring. The values are rounded using their increment db and span all values in the data. An error is issued if more than 200 values would be generated. You can given it the entire array of any size or shape or structures like that of the heads, concentrations or budget you read in using *readDat*, *readMT3D* and *readBud*. You may specify layers and you may supply a field name. If the Lay is omitted or empty it does all layers. If fieldname is omitted it assumes fieldname "values".

```
data=dbfread(dbfilename);
```

dbase-file reader. Usage: [DATA]=DBFREAD(dbffilename[,'v']), where dbffilename is the completepath-filename needed with or without '.dbf'. The oprion 'v' means verbose, i.e. more info about contents of the dbf file is produced. Default is non-verbose. DATA(NField) is a structarray holding the contents of the dbffile. It contains the fieldnames and meta values. Data(iField).values contains the record values of the field. Data(iField).values is a numeric array if its field type is numeric. Otherwise it is a cell array. The dbf specification is given at the end of this m-file. There are two variants of dbfreaders in the directory *dbfreadMatlab* and *dbfreadnew*, but these are not necessary better. I might throw them out one day.

```
plotshp(shape);
```

Plots shapes created with *readshp*. *See* 'http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf'. Shape is a struct array of shapes which includes the shape data extracted from the dbf file. See also WRITESHP, READEXP, WRITEEXP

```
[shape,SHAPE,data]=readshp(FName);
```

reads ESRI shape files. [shape,SHAPE]=readshp(FName). Use no extension. See specification in % 'http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf' %. A shapefile consists of a mainfile an index file and a dbase file % e.g. basename.shp, basename.shx, basename.dbf, basename the same, fnames using DOS 8.3 convention. Shape is a struct array of shapes which includes the shape data extracted from the dbf file. SHAPE is overall info over all shapes. Data is the contents of the dbf file (not needed, also per record in shape-struct). See also WRITESHP, READEXP, WRITEEXP, PLOTSHP

```
h=colorCell(I,xGr,yGr,clr);
```

Paints cells specified by their 2D array index I and the grid line coordinates xGr and zGr. Color is used for rendering.

```
H=maskHC(H,MASK);
```

Sets values in H to NaN for MASK==0. H can be is cell array heads obtained by reading the heads file with *readDat* or the concentrations read in by readMT3D or flowdata read in by readBud. H can also be an array of the same size as MASK. MASK is an array or scalar. If mask is array (such as IBOUND) all zeros are reset to NaN. If mask is a scalar, such as HNOFLOW or HDRY in MODFLOW, then if mask>0 all values > 0.999*MASK are set to NaN and if mask<0 all values < 0.999*MASK are set to NaN. If mask=0, all values equal to zero are set to NaN. This function is used to cleanup heads (remove HDRY, HNOFLOW etc.), concentrations and flows after reading them in.

```
cmap=jet2(len);
```

Generates and sets colormap comparable to jet(len). I like these colors better for coloring salinity gradients in groundwater models. Len is the length of the colormap. Together with setcolormap you can now handle all possible data in a single figure without bothering about the colors any more. You may want to consult Matlab's (complicated) description of this after all not too difficult issue.

```
cmap=mf_setColormap(ax,ranges,submaplengths,mapnames);
```

Sets figure colormap such that each axis uses the cmap with given length for the values in ranges. ax is an array of axis handles. Ranges is an array with values that span the color range lengths if the length of each color axis. Mapnames is a set of colormap names. There is no ouput except the colormap produced. This will however already be set by in this function. You may want to use jet2 as colormap (see function jet2).

```
[c1c4]=mf_clim(c2,c3,i2,i3,L);
```

Yields the extremes of the colormap values to be used as set(ax,'clim' [c1 c4]) to make the axis color adhere to the portion i2..i3 in the figures's color map. L is the length of the current colormap. This tackes an problem stumbled often over. Namely that if you add other data to you pot (axis) the color scheme changes to adapt the figure to the total range of values plotted.

```
B=mf_Psi(B[,iy]);
```

Adds a 2D stream function array to each cell of the budget struct read-in by B=readBud(budgetfilename). It is assumed that the model is a cross seciton in the zx plane. You may choose a specifid row. But stream functions are only userful in 2D cross sections and radial sysmmetric vertical sections. The 2D stream function arrays have fieldname B.Psi. You may contour the stream function of Cell(i) by contour(xGr(2:end-1),zGr,B(i).Psi[,contourrange]).

```
HFB=setHFB(xm,ym,ILay,xv,yv,c)
```

Generates the array defining the horizontal flow barrier for the HFB6 package. The variables xm and ym define the grid (cell centers), xv and yv are the vertices of a polygon, assumed closed, ILay is the list of layers for which the barrier is to be set and *c* is the barrier resistance. The barrier resistance *c* can be a single value in which the hoirzontal flow barrier will have the same resistance in all layers or *c* can be a vector defining the resistance for each layer in the list. *c* will always expanded to equal the list of layers.

The choices made in the function *setHFB* limits setting the barrier to closed contours with only one resistance value per layer. This may be too restricted in some cases, but the function is quite useful in most situations. The function may be extended by allowing c to be value of the resistance of each edge defined by

the polygon, so that it can vary along the polygon. This has not been done yet. It would require the function griddata.

28.6 mfiles/fdm directory, modelsize, modelsize3, fdm2, fdm3, fdm2t fdm3t

This directory contains a set of complete finite difference models entirely written in Matlab. They are used in a modeling course at the TU-Delft, in which students build their own finite difference models in Matlab. The course is included in the pdf file in this directory and should be sufficient to get acquainted with these models. The models allow extremely compact groundwater modeling in Matlab, but don't supply the wealth of options included in regular MODFLOW. However, these models can be very useful for fast modeling and verification of the MODFLOW model or with analytical solutions. Included are models for 2D, axially symmetric, 3D, steady and transient as well as density flow and finally particle tracking. See the manual for instructions.

Some of these functions are wrappers around other ones, to reduce the amount of code.

The function *modelsize*, which is calle as follows:

```
[xGr,yGr,xm,ym,Dx,Dy,Nx,Ny]=modelsize(xGr,yGr);
```

is a useful grid housekeeping function that is used in almost every model. It guarantees that grid lines are sorted and oriented into the correct dimensional direction and that all coordinates are unique with duplicates removed. xGr,yGr are grid line coordinates, xm, ym are cell center coordinates, Dx and Dy the width of the cels and Ny,Nx the number of cells in a row and column repectively.

The 3D version is modelsize3, which is called like this:

```
[xGr,yGr,zGr,xm,ym,zm,Dx,Dy,Dz,Nx,Ny,Nz]=modelsize3(xGr,yGr,zGr);
```

Below the call to most of the finite difference models are show. These model are in the fdm directory These models are kind of small jewels. Use help followed by the function name to get specific information on their usage.

Phi is the computed head, *Qx, Qy, Qz* are the flows across the cell walls, *Qt* is storage in the cells during a time step and *Psi* is the stream function. *x, y, z* are grid coordinate vectors, t is time, *kx, ky, kz* are the cell-by-cell conductivities, *FH* the fixed heads and NaN elsewhere (no IBOUND necessary), *FQ* are the prescribed flows (injection positive), *S* is primary storage, *Ss* is specific storage and *IH* is initial head. *Gamma* is the relative density, that is: $(\rho - \rho_f)/\rho_f$.

```
[Phi,Q,Psi,Qx,Qy]=fdm2(x,y,kx,ky,FH,FQ)
%Steady state 2D finite difference model written entirely in Matlab

[Phi,Q,Qx,Qy,Qz]=fdm2(x,y,z,Kx,Ky,Kz,FH,FQ);
%Same but 3D, written entirely in Matlab

[Phi,Qt,Qx,Qy,Qs]=fdm2t(x,y,t,kx,ky,S,IH,FH,FQ,varargin);
%Transient 2D finite difference model written in Matlab

[Phi,Q,Qx,Qy,Qz]=fdm3(x,y,z,kx,ky,kz,FH,FQ);
%Transient 3D finite difference model written in Matlab

Phi,Q,Psi,Qx,Qy]=fdm2dens(x,y,kx,ky,FH,FQ,gamma)
%Steady state 2D finite difference model with density flow

[XP,YP,TP]=fdm2path(x,y,DZ,Q,Qx,Qy,por,T,markers,XStart,YStart)
%Particle tracking in a 2D steady state finite difference model
```

29 The gridObject and its methods

The best way to generate and handle model meshes or grids, is using the grid object. The grid objects can hold all details of your grid, and it can provide any information computable from grid coordinates. It has

many methods to do this. It can also plot itself.

A grid object can be requested in various way but the most complete one is this

```
gr = gridObj(xGr,yGr,zGr,LAYCBD,MINDZ,AXIAL)
```

The most often used form is

```
gr = gridObj(xGr,yGr,zGr);
```

xGr, yGr and zGr are sets of coordinates in any order, which will be sorted by the *gridObject* and doubles will be removed (after having rounded the coordiantes to within a millimeter), The xGr always runs form low to high and the yGr and zGr form high to low.

LAYCBD is a vector indicating whether or not a model layer as a confining bed associated with its bottom. This is a vector in which zeor means no associated confining bed. If left out, no confining beds are assumed.

MINDZ is the minimum thickness of the layers, including confinging beds. The default is 1 mm.

AXIAL is a flag in which nonzero means that the model is axially symmetric. Nothing is internally done with this flag. AXIAL does not influence the grid, it triggers *mf_setup* to multiply relevant parameters by $2\pi r$, where r is the absolute value of the grid's x-coordinate.

Obtaining information from the grid object is done with its methods. Type gridObj and then press the blue word methods to see which methods have been defined for the gridObj class. The number of methods will likely increase further with time. Examples of information retrieval:

```
xGr  = gr.xGr;
XGR  = gr.XGR;
xm   = gr.xm;
Nx   = gr.Nx;
[Ny,Nx,Nz] = gr.size;
DZ   = gr.DZ;
DZlay= gr.DZlay
V    = gr.Vlay;
Vcbd = gr.Vcbd;
A    = gr.AREA;
HK   = gr.const(hk);
% etc etc see methods of gridObj, type gridObj and press on the blue word "methods"
```

Packages

The available packages can be seen in the worksheet NAM of the workbook "parameters.xls" in the directory mflab/examples. Packages to be used in any simulation can be switched on and off by putting a value 1 or 0 in column 4 in that worksheet on the corresponding line. So if input has been prepared of a package, and one want to run a simulation without using this information, switch off this package in the NAM worksheet. If a package does not seem to work, check is the package is "on".

Note that some packages have to be on in combination to allow a simulation. See the workbooks in the various examples. mfLab generally figures out what type of simulation (modflow, mt3dms or seawat or SWI) on the and of the packages that are "on" in the worksheet NAM. If the input has been prepared for a SWT run, it will also suffice for an MT3DMSRUN and for a MODFLOW run. If the input has been prepared for an MT3DMS run, it sufficies also for a MODFLOW run. To this end, the nam files for the different simulators are generated automatically, which may be good for testing. However, it is often just as fast to switch on and or of some appropriate packages and run *mf_run* again for a new simulation.

30 GHB, RIV, DRN, CHD, WELL, MNW packages

These packaged handle boundary conditions or stresses. RIV (rivers) and DRN (dains) are variants of GHB (general head boundaries). CHD allows defining prescribed heads at given cells, which may vary within and between stress periods. The multi-node well package MNW is a variant of the regular well package WEL.

MODFLOW wants the input of these packages is on a node to node base, consisting of a list in which each line contains a tuple consisting of the LAYER ROW and COLUMN numbero of the cell followed by

values that depend on the particular package. For the WEL package, the oly extra value is the flow (injection positive). mfLab requires a list in which each tuple is preceded by the stress period number. This makes each tuple unique, so that they can be presented or geneated by the user in any order, mfLab will sort them out.

The easiest way to generate the arrays for these packages is using the method of the grid objects called bcnZone, to be called like

```
BCN = gr.bcnZone(basename,BCNname,ZONEARRAY,zoneVals,concVals);
```

For instance, for drains, the call could be

```
DRN = gr.bcnZone(basename,'DRN',IBOUND,zoneVals,concVals);
```

Here is DRN the required list, basename is the name of the problem used for the basename of all generated files. 'DRN' is the code denoting the type of boundary condition, required to sort out which parameteres will be given in zoneVals. IBOUND is the zone array. IBOUND can generally be used as zone array by putting zone numbers in it. IBOUND is only interested whether its values are smaller than zero, larger than zero or equal to zero. It, therefore, hardly interferes with its use as a proper zone array.

zoneVals specifies the zone and the values to put at the location of the zones. Its form is

```
zoneVals = {izone1 values1 values2; izone2 values1 values2; izone3 values1 values2}
```

where izone? is a scalar denotes the zone number that corresponds with the zone in IBOUND. The values go to the bcn list, in this case DRN. DRN require the elevation and the conductance. Hence values1 must be drain elevation and values2 drain conductance. The values can be a scalar, in this value will pertain to all cells in the zone. The values may also be a vector the length of which matches the total number of cells in the zone (in the order of their cell's global index). In that case each cell of the zone may obtain a different value. Lastly these values may be a string. In that case, it will be interpreted as the header of a column in the PER worksheet. This column hold one value per stress period. This way, each cell of the zone can obtain a different value in each stress period.

Although this method has proven extremely useful, powerful and flexible, it cannot provide different values to all cells in all stress periods. Of course, it can be arranged in Matlab with some code, but to prevent too much complexity it was left out of the method.

The last input, concVals has the form

```
{conc1 conc2 conc3; conc1 conc2 conc3; conc1 conc2 conc3}
```

where each row (i.e. data separated by ';') contains the concentration values for each of the species simulated in the model, i.e. with MT3DMS or SEAWAT. We need as many conc values as there are species and as many lines as there are zones to be targeted. Zone numbers are not uses, as they are taken from the zoneVals. Like it was the case with the zoneVals, the values coc1, conc2, conc3 etc, may be scalar in which case all cells of the corresponding zone gets this value for all stress periods. It may be a vector with concentration values that is exactly as long as the number of cells in the corresponding zone. In that case all cells of the zone obtain the values form the vector, which remain the same through the entire simlation, i.e. all stress periods. The values conc1 etc may also be strings. In that case they refer to the header of a column in the PER worksheet form which the corresponding zone gets its values for the stress periods, with one value per stress period but the same for all cells of the zone. Hence the same possibilities and limitations apply as for the zoneVals.

zoneConc may be left out of the call altogether, if MT3DMS and SEAWAT are not run (only MODLOW is run).

31 WEL and MNW packages

Wells and multi-node wells can be handled in the way described before, but their is a far better way, by using well and multnode well objects. Well objects can be requested and filled automatically from a table in a workbook. They can then be put in the model grid, so that its posiiton is connected to cells, and they can be provided with their flow and (injection) concentrations if needed by MT3DMS and or SEAWAT. Finally,

after the simulation, they can be provided with the computed concentration at their screen cells. This allows efficient graphing from input and output well concentrations.

There are various ways to request and fill well objects. The best way may be to type

```
help wellObj
```

The most used forms are given here

```
well = wellObj();
well = wellObj(basename, wellSheetNm);
well = well.toGrid(gr,HK);
well = well.getQ(basename, QsheetNm);
well = well.getC(basename, CsheetNm);
well = wellObj(basname, wellSheetNm, gr, HK, QsheetNm, CsheetNm, use)
```

32 RCH, EVT and ETS packages

These packages allow defining recharge and evapotranspiration on a cell-by-cell basis. The input can be fully defined in the worksheet PER. In that case values will be valid for entire model layers. By using a zero in the worksheet cell for one of the parameters ...

INRECH INIRCH for the RCH package, or INSURF INEVTR INEXDP INIEVT for the EVT packae or INSGDF for te ETS package, the corresponding parameter values for the cell values is assumed to be defined in mf _adapt.m instead of reading it from te worksheet. This concerns RECH IRCH for the RCH package, SURF EVTR EXDP IEVT for the EVT package and PXDP and PETM for teh ETS package. Notice that PXDP and PETM can have multiple layers because more than one segment can be defined. Once column per titled PXDP and PETM defined one extra sector. If no columns with these names are present, then a single segment is assumed and, therefore the ETS package yields the same answer as the EVT package. One may switch of the multi-sector computation of ET simply by switching off the ETS package and switching on the EVT pacakge in the worksheet NAM. In fact, the ETS package makes EVT obsolete. Nevertheless it's there.

33 RCH

The RCH pacakage can be used to specify just recharge or net recharge by defining it to be equal to RCH-EVT. It's up to the user. If any cell in INRECH is less than 1, RECH must be specified in *mf_ adapt*. You can specify RECH as an array of cells one for each stress period. Each cell contains an Ny times Nx array with recharge values or a scalar, which implies that this value is to be used for the whole layer in this stress period. If INRECH is negative, the values of the pervious stress period will be resused. RECH may also be specified as a 3D array, with one layer for each stress period. In case INRECH>0 for any layer, the value is the worksheet PER will be used and the specified array for this stress period will then be ignored.

Notice that the labels above the columns contain a comment that explains the meaning of the variable in question.

The concentration of the recharge flux consituents is given in collumn CRCH in the worksheet PER. For more than one species use underscore with follow-up number. CRCH_1 CRCH_2 etc. You can add as many columns to the sheet as necessary. *mfLab* looks for the columns starting with the "CRCH" in the order given.

Table 1: Kolommen in PER worksheet voor RCH, EVT and ETS

N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD
RCH	RCH	RCH	RCH	EVT	EVT	EVT	EVT	EVT	EVT	EVT	EVT	ETS	ETS	ETS	ETS	ETS
INRECH	INIRCH	RECH	IRCH	INSURF	INEVTR	INEXDP	INIEVT	SURF	EVTR	EXDP	IEVT	INSGDF	PXDP	PXDP	PTEM	PTEM
1	1	-2.50E-04	1	1	1	1	1	165	2.50E-04	30	1	1	0.166667	0.666667	0.2	0.1

34 EVT

EVT is the standard package for evapotranspiration, where ET is maximum as defined by EVTR when the water table is above SURF and zero when below EXPD, while it varies linearly in between.

EVT behaves similarly to RCH. Negative values in the columns INSURF INEVTR INEXDP and INIEVT implies reused of the corresponding values SURF EVTR EXDP IEVT and EVT of the previous stress period will be reused. If the value in INSURF INEVTR INEXDP and INIEVT is zero, than the corresponding data are assumed to be defined in *mf_ adapt*.

The concentration of the evaproration flux is given in CEVT in the worksheet PER. This concentration is used for the EVT package and for its alternative, the ETS package.

35 ETS

ETS is the same as EVT but allows more than a single linear segment to defined the elevation zone between SURF and EXPD. Each intermediate point in this curve requires two columns in the worksheet PER (PXDP and PETM). PXDP is the porportion of the distance between SURF and EXPD measured from SURF and PETM is the proportion of the maxmum ET (EVTR) of that point. Mflab looks for columns starting with PXDP and PETM respectively from left to right. Note that PXDP must increase from column to column, whereas PETM must decrease correspondingly.

Notice that the concentration of the EVT flow is given in CEVT in the worksheet PER. It is used for either the EVT or ETS package.

Examples

36 Overview

A number of mfLab examples is available in the directory mfLab/examples. There are sub-directories for *mf2k*, *mt3dms*, *swt_ v4* and *swi* . More are to be added and some of them are still under construction today (091207). Each example resides in its own directory and each such directory always contains 3 files

1. *mf_ adapt.m* – the Matlab (c) m-file in which the model is created in terms of Matlab (c) arrays;
2. *mf_ analyze* – the Matlab (c) m-file that extracts the results of the model and visualizes them (and perhaps does extra processing or interpretation);
3. *<<basename>>.xls*, the *xls* (Excel) file holding parameters and information on stress periods and layers. *<<basename>>* is the name given to the model. This name is arbitrary and is stated at the top of mf_adapt.

As *mf_ adapt* and *mf_ analyze* are local and always named the same, you should (could) not keep more than one model in a single directory.

You can launch the simulation by typing mf_run in the command window of Matlab. After

```
mf_run
```

In the command window check that the model or models (MODFLOW, MT3DMS, SEAWAT etc). If not, mf_analyze will fail as it misses all or some of the model output.

Then type

```
mf_anlayze
```

has finished type *mf_ analyze*.

All examples are documented directly in their *mf_ adapt.m* file, where the problem and the approach are explained as comments that is interlaced with Matlab code.

It is also good to realize that each part of the code can be run by using the cell approach of Matlab (run each section of the code marked by %%). But also, that you can also run each line or even part of it separately to see what exactly it does; and, you can change code, add your own etc. In short there is ample opportunity to experiment and adapt to your personal wishes.

Moreover, to start a completely new model, the best approach is to copy a directory of a model that has some similarities with the one you want to construct and then adapt it. This is probably essential with respect to the model's workbook, holding the parameters. Most of the time chances are that you hardly need to make any changes to it, except for the specification of your stress periods and your layers (even the layers seldom need change between models, as the actual hydraulic parameters, like porosity, conductivity, storage coefficient, start concentration, are always specified in mf_adapt and never in the worksheet).

So, in Matlab, into the command window of Matla, naviate go one of the example directorys and issue the command

```
mf_run
```

You should see that the program is now running as it issues message to the screen about what it is dowing.

If you happen to get an error telling mf_run is undefined, make sure the paths are known to Matlab. If you have your *mfLab* shortcut button on Matlab's shortcut toolbar,

All examples should run if you do the following:

Make sure that the directories of the m-files are known to Matlab (c) (use a shortcut in the toolbar on top of the Matlab window (see figure) then you probably only have to press it, as below it are the commands to set those paths (see instructions in section on page 14).

If all went well, type the following command to visualize the results

```
mf_analyze
```

Or type:

```
mf_run; mf_analyze
```

on one line to run both file in sequence without a pit stop in the middle. Howevet I advise to always allow this pitstop, to check if the model code ran well. Otherwise you may inadvertendly find yourself analyzing old data all the time.

37 examples/mf2k

This directory (for the time being) contains only one example, called *ex1* (see further down). The main reason not to present more examples here, although there may be more in the future, is that the examples given for MT3DMS and SEAWAT are, implicitly, also MODFLOW examples. MT3DMS needs all MODFLOW files and SEAWAT needs the same files as does MT3DMS + one extra. Therefore, if *mfLab* generates files for SEAWAT it generates not only the name file for SEAWAT, *swt_v4.nam*, but *mt3dms5.nam* and *mf2k.nam* as well. Hence, in case of a SEAWAT model you can run MT3DMS immediately, which creates a solution without any density effect, and you can run *mf2k* directly. This may help debugging and it may be worthwhile to compare the solution with and without density effects.

38 Example, classical ex1

The example in this *examples/mf2k/ex1* directory is a simple case that was presented in the original MODFLOW manual of [4], Appendix D. It was reused in the manual of MODFLOW2000, generally referred to as *mf2k* ([5]). In the manual it is solved both with and without the parameter options that were introduced

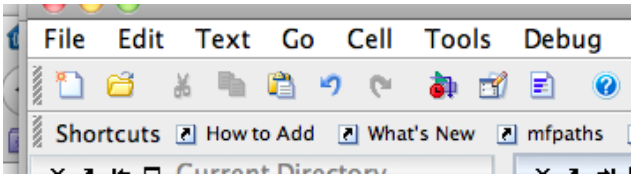


Figure 25: Shortcut toolbar with mfpaths button

with *mf2k* to facilitate calibration using its internal sensitivity and parameter optimization process. These parameters and these calibrating processes have been left out of the most recent version, MODFLOW2005 so they will be obsolete in the future, i.e. replaced by external programs UCODE and PEST. The provide example does not use the parameters. In general they are considered too complicated to use and not needed in the Matlab environment. However, the parameter options have been implemented in *mfLab*, but not really tested to date.

The example is fully documented in its *mf_ adapt.m* file. Please refer to that file for further information.

39 Calibration with PEST

39.1 Example ex1pest in examples/mf2k/ex1pest

This example is the same situation as that in ex1. It is from the mf2k manual. As in that manual we use the LPF package instead of the BCF packge. That is, instead of transmissivities, we specify horizontal conductivities HK and vertical conductivities VK and instead of VCONT between the aquifer we specify teh VKCB, the vertical conductivity of confining units between layers, when present.

The change from BCF to LPF is straightforward and documented in the script mf_adapt.m on that directory. It follows the mf2k manual in this but does not use the parametr functionality of mf2k.

39.2 Calibration by PEST

More interesting is the calibration using PEST done in that directory as well to demonstrate the use of PEST in the mfLab environment. PEST can be downloaded from

http://pesthomepage.com

Setting up PEST requires the construction of a number of files (template files, instruction files, and the PEST control file. In these a number of parameter values must be specified as explained in the PEST manual. In order to make remembering parameters easier, I put the more pertinent ones, that is, the ones that hardly need to be changed between models, in an extra worksheet called "PEST" in the workbook of this example. The parameter names in that worksheet all have an Excel comment attached to them which describes their function. This is the same as in the other worksheets.

Other PEST variables pertain to the specific case because they depend on the actual parameters and obsrvations, and will vary from one case to the next. These variables, therefore, will have to be edited for every model. In order to facilitate making the required pestfiles including editing, I wrote the script

genpestfiles.m

which is in the same directory.

This script specifies the basename of this case by reading the file *name.mat* that *mf_ adapt* already generated. It further holds the specification of the parameters, of the parameter groups, the observations and the files pertaining to them. The meaning of each of the parameter is written as comments near the locations where they are needed in the file. Their possible optons are given on the spot and need not be remembered. It should, therefore, be straightforward to adapt *genpestfiles.m* to a new situation.

Once pertinent parameters in the worksheet have been adjusted (if deemed necessary), and the file *genpestfiles.m* has been adapted to match the current case, you can run *genpestfiles.m* in Matlab to obtain the required pestfiles. The pestfiles can (should) be checked by the check facilities PESTCHEK, TEMPCHEK and INSCHEK that come with PEST.

To run PEST use is made of three simple files *run.m*, *run.bat* and *pestrun.bat*

- The file *pestrun.bat* is a DOS batch file that runs PEST with the [basename].*pst* file as argument. This one line was put in a batch file to allow prepending the path to the PEST executable as an alternative of setting the path on the Windows environment.
- The file *run.bat* is the batch file that launches Matlab with the mfile *run.m* as its argument. This will start up the model construction and run the models.
- The file *run.m* is the Matlab script that Matlab reads and executes when Matlab is loaded, and does the following:

1. set the paths in the Matlab environment, so that subsequent subsequentn instructions can find the mfiles belonging to *mfLab*.

2. to run *mf_ run.m* (which, as always, runs *mf_ adapt.*m, generates the model input files and runs the model itself (or models themselves))

3. to exit Matlab when finished.

In the example, some parameters are adjustable and some are fixed. During the parameter estimation process, PEST concludes that the correlation between some of the parameters is 1.0 (one). Therefore, the parameters that have been chosen for are not estimatable with the given small number of observations. So, some parameters should be kept fixed or tied to other parameters, or more observations be acquired. Nevertheless, it is astonishing that PEST is able, even in this badly conditioned case, to optimze the parameters to almost their true values, which are attained with all parameters are equal to 1.

39.3 How does the calibration with PEST work in the mfLab environment?

Matlab already parameterizes the construction of any model. It is, therefore, relatively easy to include any set of parameters directly in this construction. There is no need to use the parameter functionality of *mf2k,* nor is there any ned to adapt input files of whatever the model to accommodate PEST. In Matlab, we work, therefore, with a single model input file, which is a simple list of values of the parameters we with to deal with. PEST generates this list using the parameter values it computes in the calibration process. PEST does so with using the template file for this model input file. This template file is also the simplest template file that is possible, because it only needs to tranform the list of parameter values of PEST into an equal list of parameter values in the model input file. In fact, in the *mfLab* environment, we strictly speaking do without every template and use PEST's parameter list directly. But to stay in line with the PEST procedures, we use the simplet one that is possible.

Matlab then calls *mf_ run.m*, which in turns calls *mf_ adapt.m*, which reads the parameter file (the model input file) and assigns the values to the parameters in *mf_ adapt.m*. *mf_ adapt* then generates the model arrays using these parameters. When done it hands back control to *mf_ run*. The script *mf_ run.m* continuous and constructs all the input files for the models to be run as usual. *mf_ run* then calls the required models, in this case only MODFLOW (mf2k). When the model has finished, it hands over control to *mf_ analyze.m*. This mfile reads the binary (and possibly other) output files of the model and extracts the computed observations for this run. Computed observations are the values at the locations of the observations and whatever other required information suh as discharges. These computed observations are written to the model output file in the form of a simple list of values. This list with computed observation values is the only file that PEST cares about. PEST extracts these values using a PEST instruction file.

It then and decides what to de next. Normally implies running the model again, so that the described cycle is repeated for as long and often as PEST considers necesary to optimize the parameters.

39.4 Remarks

Essential ingredients turned out to be:

- the -wait option in the comment to let the batfile run.bat wait until matlab was finished. The total command line to run matlab in batch mode is:

- matlab -wait -nosplash -nodesktop -r run.m -logfile run.log (option -minimize is optional)

- This command line does not work as the command line specified in the PEST control file, as the command PEST issues to run the models. This command must be encapsulated in a batch file, see run.bat.

- The mfile run.m conveniently sets the mfLab paths in matlab, invokes mf_run and exits when ready. Due to this, nothing has to be changed to mf_adapt.m to accomoated PEST, except to use the desired parameters to construct the model arrays.

- The parameter AFTERMFSETUP was defined in mf_adapt and implemented in mf_run.

IT is used as AFTERMFSETUP=mf_analyze to ensure that *mf_ analyze* will be run immediately after the models have finished.

mf_ analyze utilizes the existense of absence of the variable AFTERMFSETUP to decide whether or not it runs under PEST. If under PEST it just gets the computed observations and des not care about vizualizing the output or do other things. This way, no extra mfiles are requried ot run PEST.

I tried JACUPDATE=999 as advocated in the manual, but this led to PEST hanging, maybe in an endless loop. Setting it to 0 caused normal behavior of PEST and normal termination.

40 HFB-package (Horizontal Barriers)

The Horizontal Flow Barrier package allows inserting barriers between adjacent cells in a column or a row. Their MODFLOW input form is

LAY ROW1 COL1 ROW2 COL2 Hydchr

where indices 1 and 2 denote adjacent cells and Hydchr is the hydraulic characteristic of this barrier. The latter is hydchr $1/c$ or k/d where c [T] is the hydraulic resistance of the barrier, k [L/T] its conductivity and d [L] its thickness. Here in Holland we prefer resistance as that is more intuitive barriers as it is for aquitards, so that one does not need a strange word like Hydchr and also immediately understands what it means. To facilitate providing input one may use the function

```
HFB=mf_setHFB(zone,ILay,c);          % possibility 1
HFB=mf_setHFB(XY,xGr,yGr,ILay,c);    % possibility 2
HFB=mf_setHFB(XYC,xGr,yGr,ILay);     % possibility 3
HFB=mf_setHFB(XYC,xGr,yGr,ILay,c);   % same as possibility 2
```

Hence, many ways to use the same function.

zone is an array of zeros outside a closed barrier and ones inside

ILay is a list of layers for which the barrier applies

c is a list of resistances for these layers. If $length(c) < NLay$, the last value of c will be used for all remaining layers. A single value assues all layers have the same resistance.

XY is a list of coordinate pairs defining the barrier in real-world coordinates. The barrier needs not be closed.

XYC three columns first two same as XY and the third column containing the resistance of the fence between poitns i and i+1. If the last point has $c > 0$ the fence will be closed, else it will remain open. To open parts of a long fence defined by a list of coordinates, make the corresponding $c\_values < 0$ or NaN. The second call has 5 arguments, so the last one is a list of resistances (may be a single value) and the first one may have three columns of which the thirtd is the resistance for parts of the barrier as explained below. If this is the case, i.e. both XYC contains c-values and the last arguments contains them, the last arguments will overwrite the first.

xGr the x-coordinates of the grid lines

yGr the y-coordinates of the grid lines

The HFB is used in example

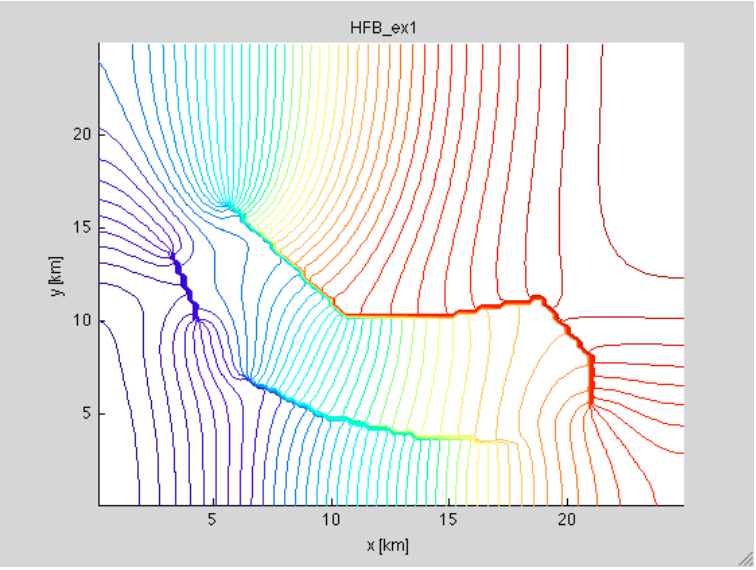


Figure 26: Example of use of HFB package where the barrier ahs some holes as can be seen from the continuity of the corresponding head lines. examples/mf2k/HFB_ex1

41 *mfLab* knows about Axisymmetric flow

mfLab knows how to deal with axially symmetric flow situations (sometimes called radial flow situations, although that would mean 3D spherical flow). If *mfLab* finds that there exists a vaiable AXIAL or RADIAL or AXIALSYMMETRIC in the workspace, it sets the variable AXIAL to the respective value and its sets AXIAL=0 if none of these variables exist.

If *mf_run* discovers the variable "AXIAL" to be nonzero, it treats the model as axially symmetric. To accomplish this it does the following (See [**?**]):

Multiplies HK, VKA, SS, SY, HY, TRAN, RECH, EVTR, PEFF, PRSITY2, RHOB with $2\pi r$. It also sets LAYAVG to 1 to compute the intercell conductance logarithmicalla and to 2 to do the same for convertible layers.

It does not check the grid. it treats the x-axis as the distance from the axis. For instance if x-direction (column direction) both positive and negative values, it treats both as distance from the axis by multiplying the above arrays with $2\pi\,|r|$ insteand of just r. In fact, you run two axial symmetric models. Countious use of this to get a full axial cross section (both sides) is probably the most-used appication of this.

When using a grid with negative and positive x-values the problem should be symmetrical around x=0. The example in figure 27 shows what the result may look like. The x-axis shown is only a small fraction of the entire axis which runs between plus and minus 5000 m. The problem is fully axisymmetric. This means that the extraction of the wells at the left and right of the center are in fact the same ring and their values are added. This prolblem also uses rechare to simulate infiltration from a basin in the center of the model. This rechare area runs from $-RBASIN$ to $+RBASIN$. Because every cell-recharge is multiplied by $2\pi\,|x-0|$ implies that the recharge an both sides of the zero are added. Therefore, only half the desired recharge should be used. Of course, if you only use half the model, that is if yous set the x-as starting at zero, all is ok immediately.

Further it does not care about the number of rows, although generally one should only use one row in case of axially symmetric models. One might use multiple rows in this case for specifc puposes, like simlation of multiple axially symmetric problems in a single run, by setting the anisotropy in y direction to zero (or virtually so as MODFLOW does not allow zero. Use CHANI $= 10^{-8}$ for example).

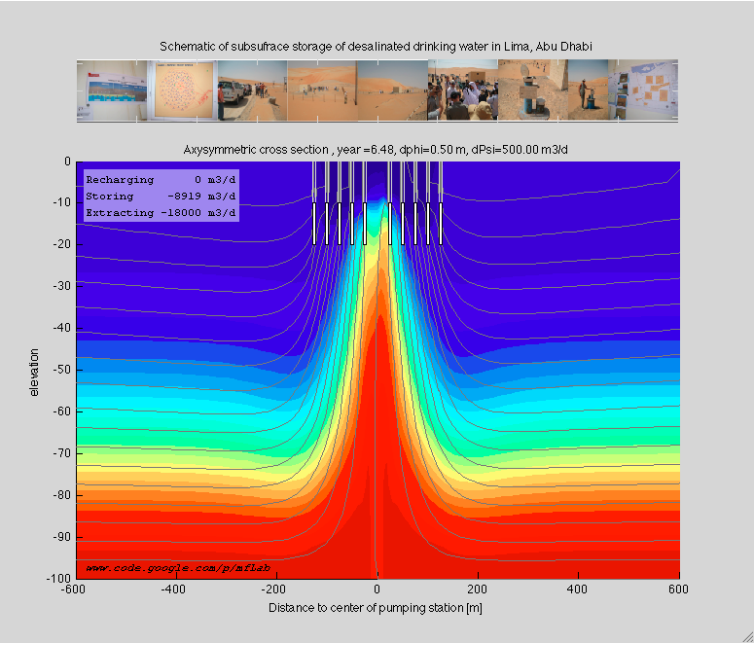


Figure 27: Example see site Lima-AbuDhabi. Desalinated water is infiltrated through a basin of 30 m radius for three years, it is then left in storage for another three years and in the 7th year the water is suddenly needed, leading to high salinities in the water if the abstraction is too long ad in this picture. The center wells that were in the center of the fresh desalinated water initially, laker are the first to attract the saltwater as the picture shows.

41.1 What to take care of when running Axisymmetric models

1. First, mfLab treas the absolute value of the x-coordinate ad the distance to the heart of the model. Therefore, to convert a non-axi-symmetric model to an axisymmetric model, make sure the zero of the grid coincides with that of the center, which is often the coordinate of the well.

2. Secondly don't forget to subtact also the well's screen x and y coordinates, so that they become zero (i.e. the center of the grid).

3. Thirdly, when generating a grid that is axisymmetrical, mfLab makes sure that there is no grid line exactly at the model center. This makes sure the well will be in the center cell.

4. Fourthly, be aware that all coordinates are multiplied by $2\pi r = 2\pi |x|$. This is true for both the coordinates to the left as for those to the right. This means that if the center of the grid is somewhere in the middle, both the coordinates to the left and those to the right of the center will be multiplied by $2\pi r$. You will thus have two axi-symmetrical models in one. That is the well in the center extracts half of its water from the left full axisymmetrical model and the other half from the right axisymmetrical model. If the drawdown computed by the model will, therefore, be half that computed analytically by that of Thiem. In fact, this model should be regarded as two full size axi-symmetrical models, that are glued together at the center model column. These models are virtually independent, as they only connect at the center where $x = 0$, where r is very small. Hence to obtain a correct Thiem solution, the extraction at the center must be multiplied by 2.

5. All other extractions not at the center, so at $|x| > 0$ represent extraction screens that are $2\pi r$ long around the center, whose total extraction is the value given to them. Hence a well with extraction Q at r from the center of the model is in fact a screen around the center with this radius that extracts a total flow equal to Q. Putting such a well both to the left and to the right of the model center repesents two screens each at distance r from the center and both extracting their full flow. Still the drawdown cannot

be added, as both the left and the right side of the center represent full size axisymmetric models that are virtually indepdenent.

6. Definitely the best way to model axisymmetric flow is to make sure the grid starts at $x = r = 0$ and further only applies positive coordinates.

7. To obtain a full scale drawdown or concentration figure in the case of axisymmetric flow, just mirror the heads and concentrations at the right side to the left side. This can be done automatically using a proper function.

42 Boussinesq, unconfined sloping base aquifer

This examples explores the difference between the MODFLOW way of modelling a sloping base unconfined aquifer with the exact analytical solution, which solves the Boussinesq equation (Steward, 2007) for uniform discharge (no recharge). In MODFLOW a sloping base aquifer is by stepwize adaptation of the top and bottom of the layers according to the base of the aquifer. This way ,the model becomes a staircase. The governing partial differential equation, according to Boussinesq is however has been solved analytically (Steward, 2007) and can, therefore, be compared with the numerical solution.

We will first explore the analytical solution.

Darcy's law combined is

$$Q_x = Q_s = -kh\frac{\partial \phi}{\partial s}$$

The discharge is uniform, therefore, Q_x is constant. Further, $Q_x = Q_s$. The aquifer which has inclination angle θ (positive when the slope is upward in the $x-$ or s-direction, see figure 28, so that $\partial B/\partial s = \sin\theta$. The stream lines are essentially parallel to the base of the aquifer. Therefore, there is no head loss perpendicular to the bas, along n. With a water thickness h measured along direction n, the head equals

$$\phi = B + h\cos\theta$$

Hence, Darcy for the inclined aquifer in it natural $s - n$ coordinates beomes

$$Q = -kh\frac{\partial\left(h\cos\theta + B\right)}{\partial s} \tag{1}$$

$$Q = -kh\left(\cos\theta\frac{\partial h}{\partial s} + \sin\theta\right) \tag{2}$$

On the other hand, when the Dupuit assumption is strictly appied, then one assumes $h \simeq H$ and $ds \simeq dx$ and $\phi \simeq H + B$. This is, in fact, three times incorrect. Nevertheless, this leads to

$$Q = -k^* H\left(\frac{\partial H}{\partial x} + \tan\theta\right) \tag{3}$$

If $k = k^*$ then both equations differ and more so the larger $|\theta|$. Both equations can be made equivalent by setting (Steward, 2007)

$$k^* = k\cos^2\theta \tag{4}$$

This is clear from

$$\begin{aligned} Q &= -kHcos\theta\frac{\partial H\cos\theta}{\partial s/\cos\theta} - kH\cos\theta\cos\theta\tan\theta \\ &= -kh\cos\theta\frac{\partial h}{\partial s} - kh\sin\theta \end{aligned}$$

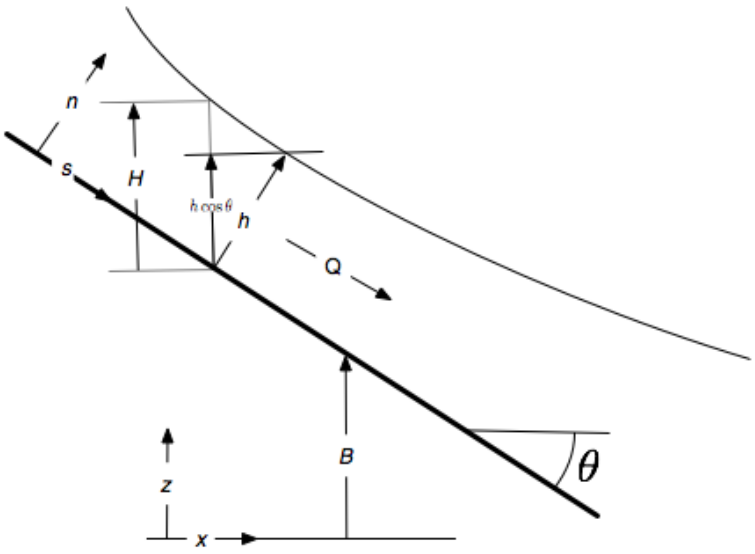


Figure 28: Groundwater flow on a slopt Boussinesq (1879)

This replacement was frst made by Steward (2007). It efficiently converts models and solutions based on three Dupuit-like assumptions to the Boussinesq equation in $s-n$ coordinates, which only assumes that streamlines are parallel to the base of the aquifer. The so-called normal depth, that is, the depth on a long slope with inclination angle θ given uniform discharge Q is obtained by setting $\partial h/\partial s = 0$ or $\partial H/\partial x = 0$. Hence,

$$h_0 = \frac{Q}{k \sin\theta}, \quad H_0 = \frac{Q}{k^* \tan\theta} \tag{5}$$

Both equations are equivalent if $k^* = k\cos^2\theta$.

Continuity with recharge and storage N leads to the governing partial differential equation

$$\frac{\partial Q}{\partial s} = N\cos\theta - S\cos\theta\frac{\partial h}{\partial s}, \qquad \frac{\partial Q}{\partial x} = N - S\frac{\partial H}{\partial x}$$

Where we take S to be the projection of the storaga on the horizontal plane. These expressions which are both equivalent because $dx = ds\cos\theta$. This implies that models based completely on the Dupuit assumption, which include finite difference models like MODFLOW can be corrected by setting conductivity as in (4). Steward futher prooves that stepped models, in which the bottom of the aquifer varies in steps approach models with truly sloped bottoms as the stepsize is further and further reduced. However the correction of the condutivity remains essential. The advantage of the correction given by Steward (2007) is that models like MODFLOW can be made to yield very accurate results for even steep slopes if we correct the conductivity according to (4).

Note that the correction also holds for confined flow conditions. In the $s-n$ coordinate system we would have

$$Q = -kh\frac{\partial \phi}{\partial s}, \quad Q = -k^* H\frac{\partial \phi}{\partial x} \tag{6}$$

where h is the now fixed aquifer thickness measured perpendicular the the flow (parallal to floor and bottom of the aquifer) and H the now fixed aquifer thickness measured vertically along the z-axis. Hence, for a sloping aquifer of constant thickness, $H = h/\cos\theta$, while with $dx = ds\cos\theta$, leading to

$$Q = k^* \frac{h}{\cos^2\theta}\frac{\partial \phi}{\partial s}$$

which means that the right (MODFLOW) expression of (6)is equivalent to the left one if we set $k^* = k\cos^2\theta$.

In conclusion, the conductivity of the layers a stepped finite difference model should generally be corrected by (4). This error due to ignoring the \cos^ \theta is only 3% for a10 degree slope, 12% for a 20 degree slope and 25% for a 30 degree slope and 50% for a 45 degree slope. So, generally it is of little importance. Nevertheless a correction is possible, wich can be made beforehand using the local inclincation of the slope of the aquifer (bottom) which should be obtained from its elevation, in both x- and y-directions.

Polubarinova Kochina (1962) developed an analytical solution based on (3) for uniform discharge , which can now be used in corrected form as follows (4):

$$\ln \frac{\frac{H}{H_0}-1}{\frac{H_1}{H_0}-1} + \frac{H}{H_0} - \frac{H_1}{H_0} = -\frac{B-B_1}{H_0} = -\frac{(x-x_1)\tan\theta}{H_0} \tag{7}$$

Steward (2007) derived an analytical solution in $s-n$ coordinates, yielding

$$\ln \frac{\frac{h}{h_0}-1}{\frac{h}{h_1}-1} + \frac{h}{h_0} - \frac{h_1}{h_0} = -\frac{B-B_1}{h_0\cos\theta} = -\frac{(s-s_1)\sin\theta}{h_0\cos\theta} \tag{8}$$

and h_0 and H_0 the normal depths according to (5) and h_1 is known at s_1, while H_1 is known hat x_1. These equations are implicit in h but explicit in x and s respectively. Hence, computing s as a function of h or x as a function of H may be the easiest computational approach.

The possible solutions are shown in figure 29. The left figure shows that when the specified head is above h_0, line b, the head will become more and more hoirzontal downstream and will never approach h_0. Physically this is because downstream of any point where $h_1 > h_0$ the aquifer thickness is larger than h_0 and, given the uniform discharge matches the slope, the gradient must be less than this slope. Hence, downstream of this point the head must flatten more and more to match the given uiniform discharge. This is shown in the left picture of the figure. On the other hand, downstream of a point where $h < h_0$ (line a and point A) there can be no solution. This is because both h and the head gradient are smaller than those necessary to discharge Q, namely h_0 and the aquifer base slope. Therefore, the discharge can never be obtained downstream. What happen if a fixed head is maintained lower than h_0 indicated in the figure is that the discharge will be smaller, so that it exactly matches the aquifer slope and the head downstream everywhere equals h_1, because no more water can flow downward than the quantity that corresponds to h_1. Of course, the actual slope may be smaller if a head is also maintained not to far downstream.

The figure to the right focuses on h upstream of point A. Both curves are physically possible. The lower than h boundary conditon, line a, and hence flow-through area is compensated for by a head gradient that is steeper than the slope of the base of the aquier. Further upstream the head will approach that corresponding to the normal depth.

42.1 Comparison with analytical solution

The analytical solutions (8) and (7)are implicit in the head and explicit in the distance from the point where the head is know. We will therefore, compute the distance as function of $\xi = h/h_0$ it as

$$x - x_1 = \frac{h_0}{\tan(-\theta)}\left\{\ln\frac{\xi-1}{\xi_1-1} + \xi - \xi_1\right\} \tag{9}$$

Where $\xi_1 = h_1/h_0$. A solution is only possible if both ξ and ξ_1 are positve or negative, i.e. h and h_1 both greater than h_0 or both smaller than h_0. There are two fundamental cases, one is $\xi_1 < 1$ and the other is $\xi_1 > 1$. In the first case $0 \le \xi \le 1$ in the second case $\infty \ge \xi \ge 1$. In the frist case $\xi < \xi_1$ is upstream of point A and $\xi > \xi_1$ downstream. In the second case $\xi > \xi_1$ is upstream and $\xi < \xi_1$ downstream of point A. The solution is computed for the case $\xi_1 = 2$ and $\xi_1 = 0.5$, and shown in figure (32).

The solution gives the head for an inifnitely long downward slope, so that in any case the head will ultimately be equal to h_0 $\xi = 1$. There must be another solution valid for an infinitely long upward slope, where the head far upstream equals h_0 an, and deviates near point A to adapt to the boundary condition at A.

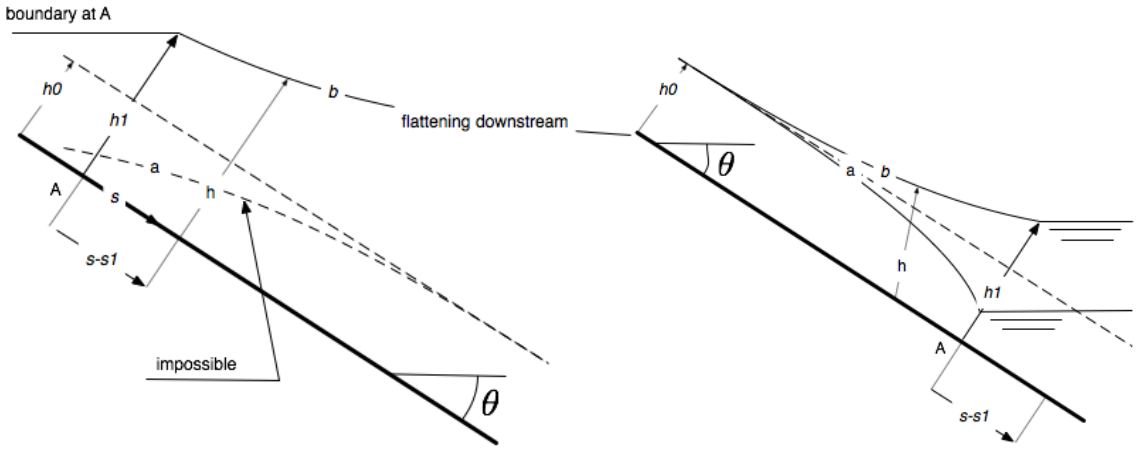


Figure 29: Boussinesq solution for uniform discharge. Left: downstream from point A. Right: Upstream of point A.

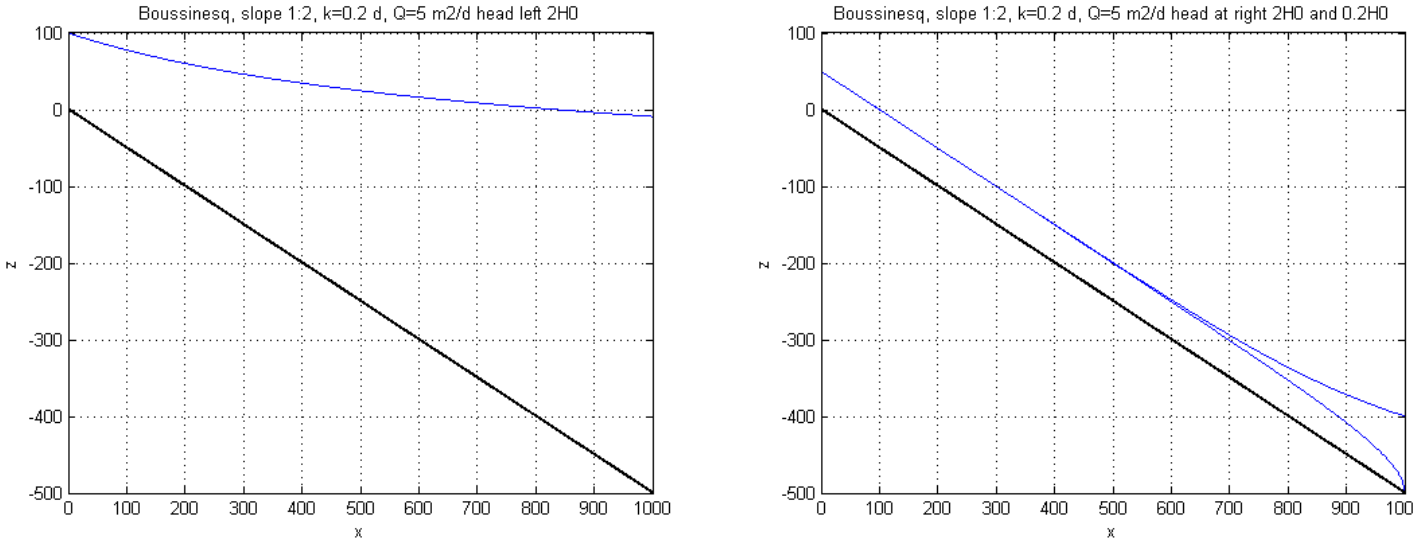


Figure 30: Numerical solution using $k = 0.2\,\mathrm{m/d}$, $\tan(slope)$=1/2, $Q = 5\,\mathrm{m^2/d}$, boundary $2 \times H_0$ and $0.2 \times H_0$, 1000 cells of 1 m, Picture left: Situation downstream of point where $h > h_0$. Picture right, head upstream of point where $h > h_0$ and where $h < h_0$.

The analytical solution is shown in figure 32; the mfile which generated the figure is shown in figure 31. For easy comparison with the numerical results in figure 30, the same numerical values were used while the solution was superposed on the slope.

In the analytical solution given in figure 32we chose a bounary conditions H=0.5H0 and H=2H0 at x=0. The two curve inlcudes the three numerical cases if considered relative to the point $x = 0$ where $h = h_1$ is given.

Finally, the analytical solution makes it possible to show the water depth in dimensionless form

$$\frac{s - s_1}{h_0} \tan(-\theta) = \ln \frac{\xi - 1}{\xi_1 - 1} + \xi - \xi_1 \tag{10}$$

which reveals over what distance the boundary has effect on the head (see figure 33).

The numerical solutions have been obtained using fdm3 and the m-files in the example directory examples/mf2k/Boussinesq. To obtain them we adapt the top of the aquifer to the head elevations a number of times, until the the head and aquifer top no longer change.

42.2 Numerical solution in mf2k

The file mf_adapt to set up the model to compute the Boussinesq solution with mf2k also runs the Matlab function fdm3.m in the mfiles/fdm directory of *mfLab*. The fdm is a 3D steady-state finite element model that is great for testing purposes and many other things. In this case it was used to experiment with the set-up until the analytical solution was reproduced. The fdm3 model is run completely inside Matlab. It is iterated while adapting the top of the aquifer after each run, until top and head converged to the solution of the Boussinesq equation. But mf_run continuous issueing the input file for mf2k and runs it. The result obtained with mf_analyze is plotted as red dots over the blue curves produced by fdm3. There is no difference between the two. Hence the Boussinesq solution can well be computed using mf2k through *mfLab*. The results are shown in figure 34. The blue lines produced by fdm3 are completely covered by the red dots which are the plot of the results from mfd2k.

42.3 Varying slope inclination

Figure .. gives the situation for a water table aquifer on a slope with variable inclination. This case was also computed both with fdm3 and mf2k through *mfLab*. In mf_adapt set A to 0 to get a straight slope.

```
%Boussenesq analytical Steward WRR 2007
% Analytical solution Boussinesq, Polibarinova Kochina (Steward, 2007)

slope=-1/2;  % tan(theta)
Q=5;         % m2/d
k=0.2;       % m/d

H0=-Q/(k*slope); %normal depth

% eta = H/H0, etal=Hl/H0
etalA=2.0; etaA = logspace(0,log10(4),50);
etalB=0.5; etaB = linspace(0.001,0.999,100);
sA=H0/(-slope)*(log((etaA-1)/(etalA-1))+etaA-etalA);
sB=H0/(-slope)*(log((etaB-1)/(etalB-1))+etaB-etalB);

xlabel('x-x0 [m]'); ylabel('phi'); title('Boussinesq analytical');
plot(sA,H0*etaA+sA*slope,'b',sB,H0*etaB+sB*slope,'r',...
    sA,sA*slope,'k',sB,sB*slope,'k',...
    sA,sA*slope+H0,'g',sB,sB*slope+H0,'g');
```

Figure 31: Computation of analytical solution in Matlab

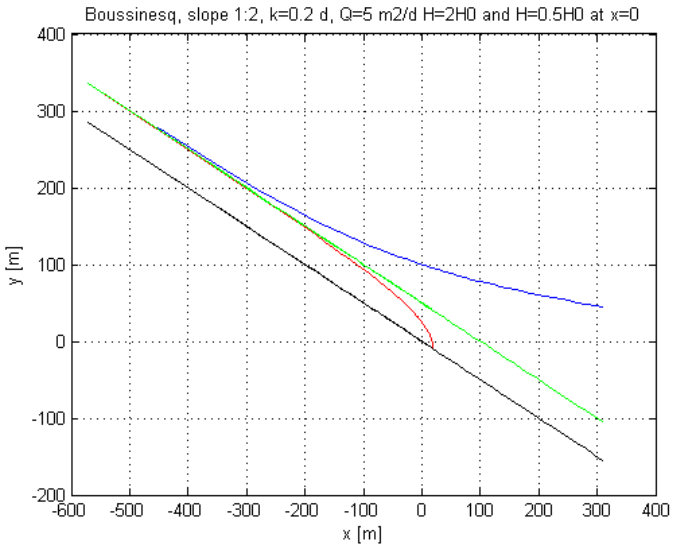


Figure 32: Analytical solution of Steward (2007), i.e. equation (8).

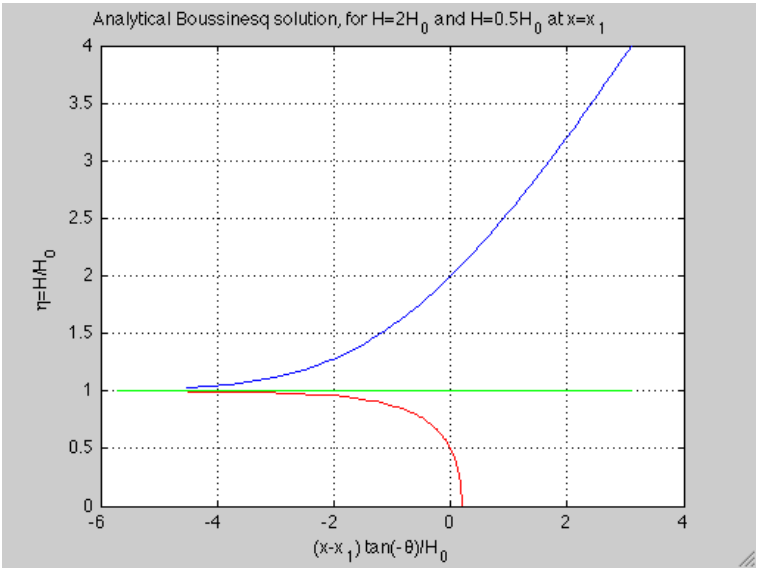


Figure 33: Analytical Boussinesq solution in in dimensionless form

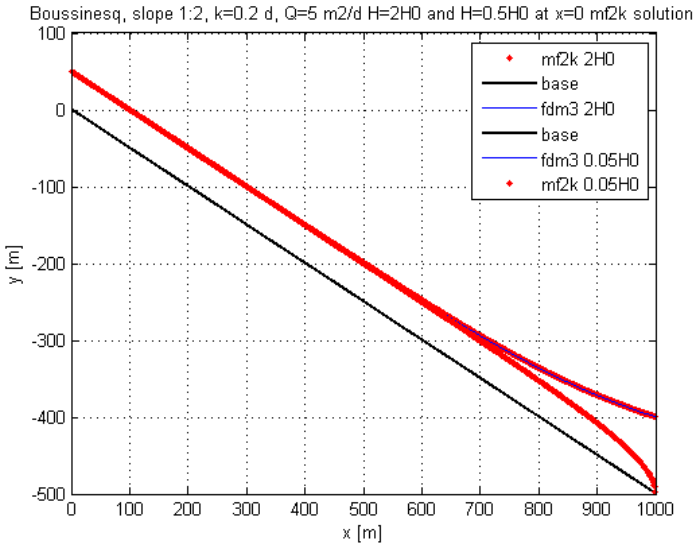


Figure 34: Head computed with mf2k through *mfLab* (red) and fdm3 (blue) for uniform discharge of $5\,\mathrm{m}^2/\mathrm{d}$ and head boundary condition at right-hand side of $2H_0$ and $0.05H_0$ respectively, with H0 the normal depth and $k = 0.2\,\mathrm{m/d}$.

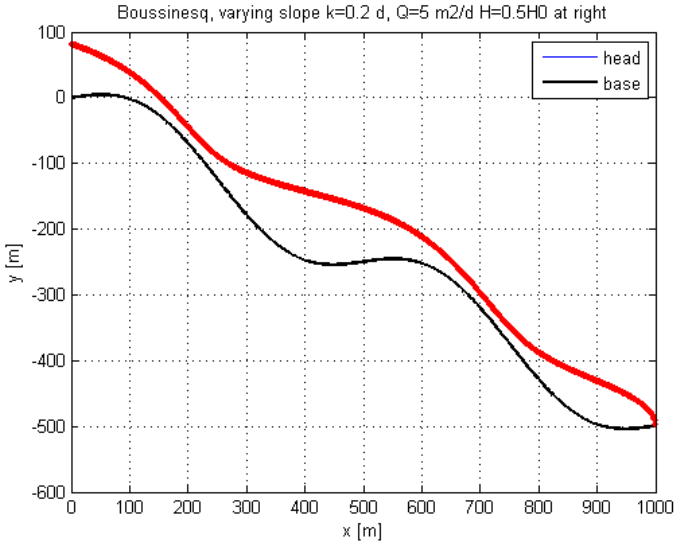


Figure 35: Contant groundwater discharge on a sloping base

43 Boussinesq with recharge, cell-rewetting problems

We may compute the flow on a sloping aquifer with recharge. Doing so, we stumble over notrious convergence problems due to the rewetting procedures implemented in MODFLOW. Wetting convergence problems have been an long standing headache for many modellers and even after 30 years of MODFLOW, these have not be tackled adequately in the official version. Hence, others have tried to overcome at least some of them (Doherty (2001), Painter et al (2008)). Doherty's approach allows to achieve convergence be it not completely satisfactorily as we shall see. First we give some hints regarding how to deal with portions of the model that run dry during a simulation and may be alternatingly dry and wet during the iterations of the solver, and thus prevent convergence and possible cause spurious uncertain en somtimes even wroing results.

43.1 Wetting

Convergence Options for MODFLOW with the wetting cells option on.

This is especially importanf for automated parameter estimation. Optimal parameter values can only be determined if MODFLOW produces a stable solution for all (intermediate) iterations. Cells going dry duing any iteration is a the known major cause of trouble, that is failure to converge. When this happens, a successful run may be posible by tweaking some convergence options. (Some non-USGS versions of MODFLOW have extra options as the ones added by John Doherty to the GMS-version. An important one is to not stop on non-convergence. A proably trivially simple one to implement in the USGS version as well. Another obvious options, added by Doherty to the GMS version is to assign bottom elevation to dry cells. This could be extended into "assign predetermined elevation to dry cells, where the user supplies a 3D array with these elevations, which could the bottom elevation of all cells or just the bottom elevation of the model, for example. All we can say, that MODLFOW is in strong need of some useful extra options to handle these nasty dry cells. A similar option in GMS lacking in the standard version is to just prevent cell drying.

Richard Winston, MODFLOW expert of USGS gives some additional instructions valid for the standard MODLFOW version:

1. Cells you know should never become wet, should be made inactive.
2. Adjust the value of the wetting threshold in WETDRY. (Higher is more stable but may be less accurate.)
3. Decide which neighbors will be checked to decide if a cell should be wetted using WETDRY. Often it is better to allow only the cell beneath the dry cell to rewet it.
4. You can use IHDWET to determine which equation is used to specify the head in newly wetted cells.
5. You can vary the wetting factor WETFCT.
6. In steady-state conditions you can adjust initial conditions to values that are close to your best guess of the final conditions to improve stability.
7. You can choose a different solver. The SIP, PCG1, and PCG2 solvers will work with the wetting capability. The SOR solver doesn't work well with the wetting capability. Note that cells can not change between wet and dry during the inner iterations of PCG solvers.
8. When using the PCG2 solver, you can set RELAX in the range of 0.97 to 0.99 to avoid zero divide and non-diagonally dominant matrix errors. (However, this is an infrequent cause of instability. If such an error occurs, PCG2 prints an error message in the output file and aborts the simulation.)
9. When using the PCG2 solver, you can set DAMP to a value between 0 and 1.
10. Unrealistically high conductances on boundary cells can contribute to instability. Check the conductances in the Drain, River, Reservoir, Lake, Stream, and General-Head Boundary packages. In the Evapotranspiration check the EVT Flux Stress[i] and EVT Extinction Depth which together control the conductance of evapotranspiration cells.

11. The two most important variables that affect stability are the wetting threshold and which neighboring cells are checked to determine if a cell should be wetted. Both of these are controlled through WETDRY. It is often useful to look at the output file and identify cells that convert repeatedly from wet to dry. Try raising the wetting threshold for those cells. It may also be worthwhile looking at the boundary conditions associated with dry cells.

12. Sometimes cells will go dry in a way that will completely block flow to a sink or from a source. After that happens, the results are unlikely to be correct. It's always a good idea to look at the flow pattern around cells that have gone dry to see whether the results are reasonable.

13. Related Links: Running MODFLOW Post Processing Solver Packages

43.2 Doherty's approach

John Doherty (2001) the maker of PEST parameter estimation package, concludes, like many others that wetting rewetting has been and still is probably the most frustrating part of MODFLOW, as it often prevents convergence when cells alternate between dry and wet during the iteration process. Regarding parameter optimization, the discontinuity of the sensitivities due to this often prevents convergence of the parameter optmization process entirely. Dehorty (2001) therefore made some simple adaptations to MODFLOW to prevent cells from going dry altogether and to guarantee cell-transmissivity continuity, even when the head falls below its bottom.

Next to the numerical necessity, he claims and rightfully so, that aquifers in reality never run completely dry, as the physical system merely becomes unsaturated. This implies, that the porous medium will still be capable of transporting water when head is negative, be it at a much reduced conductivity.

We may add to this, that if the head in an aquifer is exactly at its bottom elevation, there still exists a full capillary zone capable of transporting water at saturated conductivity. Only when the head falls further below the bottom of a layer than the thickness of its capillary zone, are we thrown back to full unsaturated conductivity and transmissivity. From this perspective, dry cells are generally physically incorrect.

Doherty (2001) add an exponential relation between head and elevation above the cell bottom, which is also valid when the head falls below the cell bottom and hence, mitigates the usuallinear relation that would yield a hard zero conductivity when the head is at or below the cell bottom. He claims that this solves wetting frustration in a wide range of situations.

Doherty (2001) sets the horizontal cell transmissivity is to

$$\begin{aligned} T &= K d_0 e^{-\frac{d}{\delta_1}} + K \min(d, D); & B > 0 \\ T &= K d_0 e^{\frac{d}{\delta_2}}; & B \leq 0 \end{aligned}$$

in which d_0 might be interpreted as the capillary zone thickness, d_1 is a thickness determining how fast the exponential term decays with wetted cell thickness B and d_2 determines how fast the conductivity drops when the head is below the cell bottom ($B \leq 0$).

Expressed in this way, d_0, d_1 and d_2 all have dimension [L] and all have positive values. Note that the way this is expressed here differs a little from the way applied by Doherty, but the idea here is to use physically interpretable parameters as much as possible.

To make the derivative of the cell transmissivity continuous across $d = 0$, gives

$$\left(\frac{\partial T}{\partial d}\right)_{d=0} = -K\frac{d_0}{\delta_1} + K = K\frac{d_0}{\delta_2}$$

hence,

$$\frac{d_0}{d_1} + \frac{d_0}{d_2} = 1$$

Therefore, the requirement that all variables in thi equation are positive, implies that both $d_1 > d_0$ and $d_2 > d_0$ implies that $\delta_1 < d_0$. We might, choose $d_1 = d_2 = 2d_0$. However, to deal with heads that during

initial iteration fall below the bottom of the aqufier, it seems wise to chose $d_2 > d_1$ so that the decline of the conductivity for heads below the aquifer bottom is not that fast. This mitigates non-linearity.

Using this approach we have

$$\begin{aligned} T &= K d_0 e^{-\frac{B}{d_1}} + K \min(B, D); & B > 0 \\ T &= K d_0 e^{\frac{B}{d_2}}; & B \leq 0 \end{aligned}$$

In order to achieve convergence we have experiment with both d_0 and d_2, note that d_1 is fixed once we chose the former two.

Another important factor in achieving convergence is to update the conductivity mildly by choosing a weighed average between the last value and the current update.

$$k = \beta k_{old} + (1 - \beta) k_{new}$$

where $0 < \beta \leq 1$.

43.3 Example

The above suggets that we need Doherty's special version of MODFLOW to apply his approach in *mfLab*. We don't have this version (it is included in the GMS user interface). However we can still use his apprach when we simulate the situation by means of the fdm3 model available in the mfiles/fdm folder of *mfLab*. The file fdm3.m is a matlab function, which in fact is an entire steady-state 3D finite difference model that can reasily be run. However, this model does not by itself correct cell thicknesses according to their wetting percentage. We can however place this function inside a loop and each time when it computed the new heads, we can correct the conductivities accoring to Doherty's approach described above. This is done in the script mf_adapt in the Boussinesq directory.

The model is single layer on a decending bottom with sinusoidal waves in it. Hence there are steep and less steep intervals along the slope. On the steep portions, we expect the wetted thickness to be much less than on the more horizontal portions. The head at the right side of the model is 10 m above the base of the aquifer there. Further recharge is given.

The slope to be modelled has a given recharge and a fixed head boundary condition downstream at the bottom of the slope. The computations will be done with three magnitudes of the recharge, 0.001, 0.01 and 0.1 m/d. The Doherty coefficients that worked well are $d_0 = 2.5\,\text{m}$, $d_2 = 5d_0$ from which d_1 follows. Clearly, these parameter values are much larger than perhaps desirable. As a consequence the head will easily fall below the aquifer bottom, because the conducitivity does not fall that fast with distance of the head below the cell bottom. Nevertheless a smooth head curve is obtained, as well as the numerical value for the transmissivity along the slope. Dividing this tranmissivity by the conductivity provides a good approximation of the local thickness of the saturated zone, which, if desired, can be added to the elevation of the aquifer bottom to obtain a better approximation of the water table.

The example deals with a slope aquifer bottom that varies in angle and at some points get really steep (1:1). The water table head has been computed for the data pertaining to the aquifer (conductivity) and three values of recharge/infiltration of respectively 0.001, 0.01 and 0.1 m/d.

Figure 36 shows the computed head and water table along the slope for 0.001, 0.01 and 0.1 m/d recharge. The left picture is the one computed with the Doherty settings as described. The right picture shows the corrected values. The saturated thickness has obrained from the computed transmissivity of the left figure devided by the conductivity of the aquifer material and adding this saturated thickness to the bottom of the aquifer.

It is clearly seen that the computed head, especially for the 0.001 m/d case, lies at least partly below the bottom of the aquifer, which was allowed by and a consequence of Deherty's method. It might loosely be seen as unsaturated conditions causing the head to fall below the bottom of the aquifer. The right picture shows the corrected values. As can be seen, it is hard to discern the curves for the lower Precipition rates from the bottom elevation of the aquifer. This implies that the saturated thickness is small compared to the vertical scale of this figure. The blue line, beloging to the lowest recharge rate of only 0.001 m/d, is always close to the bottmo of the aquifer. However, if the recharge is made 10 times as high, deviations occur at

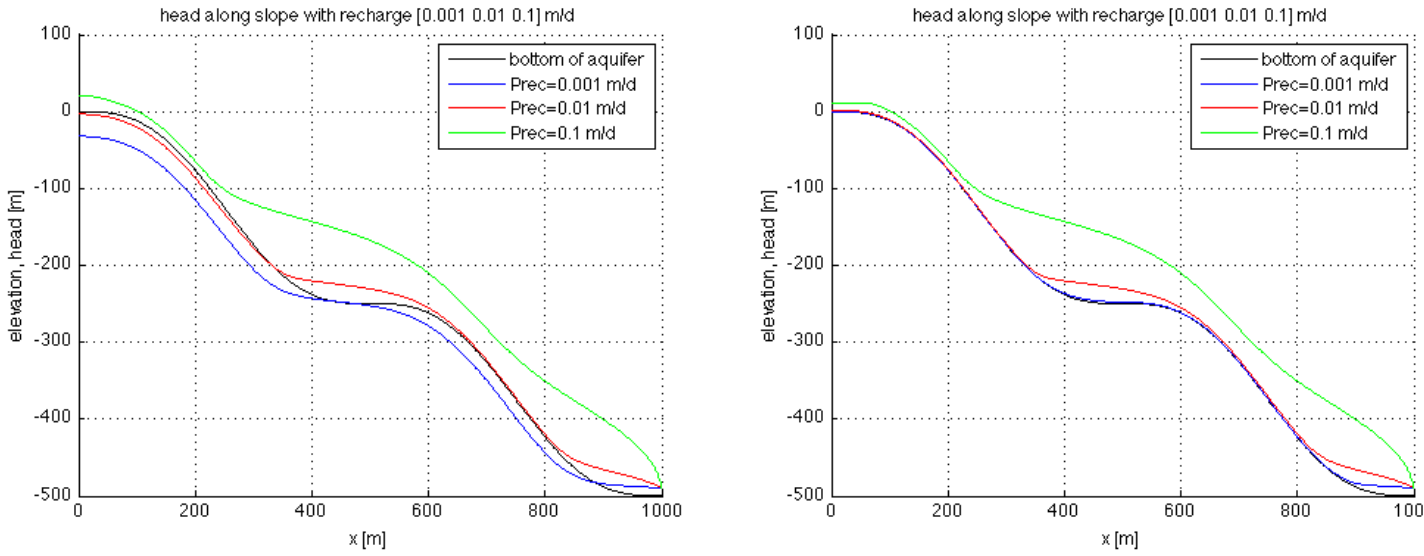


Figure 36: Water table above a sloping aquifer base with recharge (see text)

the horizontal plateau of te drawn slope. On the steeper portions of the slope, the head is still very near the aquifer bottom at the scale of the drawing. Only in the third case with again a ten times higher recharge, is the head lifted over several tens of meters above the sloping base along its entire length. Here we see no difference between the uncorreced and corrected situtiations of the water table.

Notice these model runs while mimicking Doherty's approach was entirely implemented in Matlab using the Matlab function fdm3.m. No further attempts were done to compute this case in Modflow because of the experienced problems with dry cells. The coputations could be easily done using Doherty's adapted Modflow code. Because I don't have that code, I mimicked it entirely in Matlab, which turned out to be successful.

examples/mt3dms

What follows is a selection of of the Benchmark examples in the manual of MT3DMS ([12], [13]). The description is kept as short as possible and will be partly insufficient. Please refer to the original manual and the script mf_adapt in the example directories for details and the used numbers.

44 1D-Uniform

This example concerns one-dimensional transport in a uniform flow field. The example is divided into 4 cases. Advection only (a), advection and dispersion only (b), advection dispersion and sorption (c), advection, dispersion, sorption and decay (d). Figure 37 shows the development over time for the four cases. The four cases have been computed in a single run, by setting the properties of the for layers according to the requirement of each case and setting vertical dispersivity to zero, so that no exchange between the four layers was possible during the run.

45 1D-Nonlinear

This example concerns one-dimensional transport with non-linear sorption. Both Langmuir and Freundlich sorption has been computed with *mfLab*. Figure 38 shows the development of the concentration curves over time with Langmuir sorption. Figure 39is another way of showing the results, it is the computed breakthrough at a fixed point. See description in the MT3DMS manual and the script *mf_ adapt* for details and numerical values.

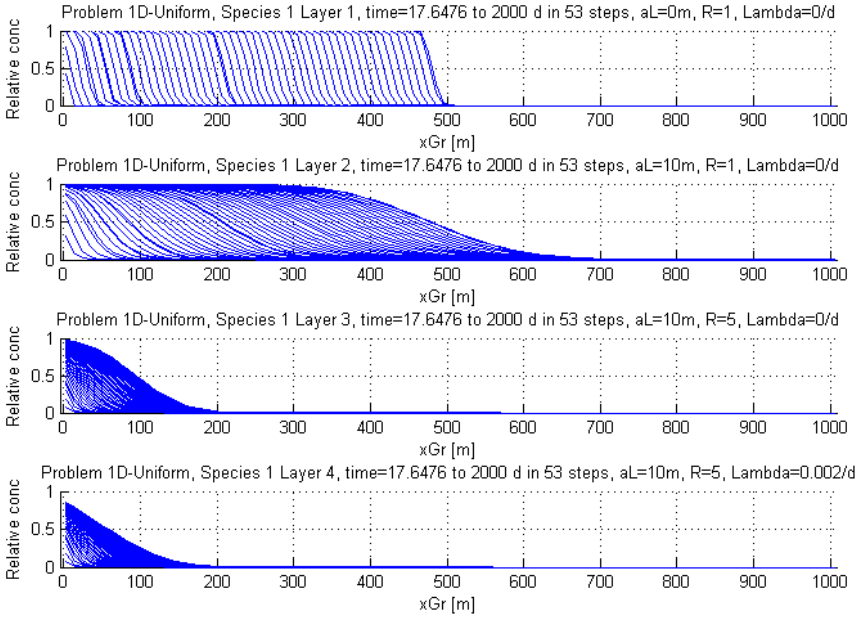


Figure 37: 1D-Linear: The four cases as described above. See *mf_adapt* in the example directory for numerical values

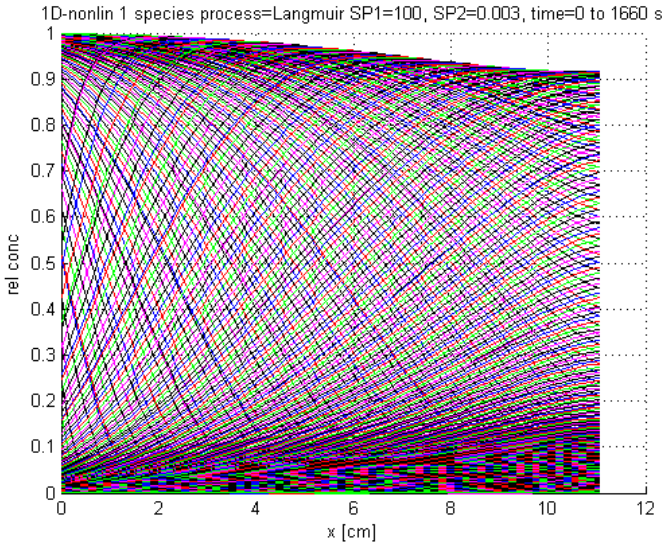


Figure 38: 1D nonlinear: Concentration curves at many times during the transport

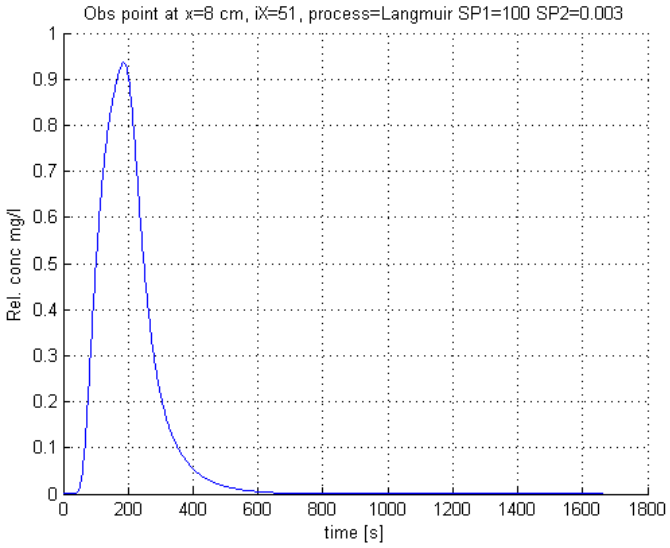


Figure 39: 1D-nonlinear: Breakthrough of the concentration at a fixed point (x=8 cm) for Langmuir nonlinear sorption

46 2D-Uniform

This example concerns two-dimensional flow in a uniform flow field in a relatively thin aquifer of infinite extent. Instantaneous vertical mixing can be assumed. The injection rate is negligible compared with the ambient groundwater flow. The model consisting of 46 columns, 31 rows and 1 layer with no-flow boundaries at the north and south and a given ambient flow from west to east. This ambient flow is implemented with a fixed head at the east side of the model and a fixed inflow at the west side. See the original MT3DMS manual and *mf_ adapt* in the example directory for details and the numerical values that were used. Figure 40 shows the concentration contours after 365 days of injection as compute with the MOC procedure. It requires merely a change of the MXELM parameter on the MT3D worksheet in the accompanying workbook to obtain the results with a different advection computation method. Figure 40 shows the spread of the contaminant after one year and 41 shows the concentration over time at various points of the grid.

47 2D-Diagonal

The example concerns two-dimensional flow in a diagonal flow field (see figure 42). This example is similar to the previous one, except that the direction of the ambient flow is at 45 degrees with the x-axis. The grid measures 100 columns by 100 rows of 10 by 10 m cells. The groundwater seepage is 1 m/day. To maintain this ambient flow, the required gradient was computed and used for the initial head field. This gradient was fixed at all boundaries of the model by setting the IBOUND value for the boundary nodes at -1 (fixed heads). Porosity is 0.14, longitudinal dispersivity 2 m, transverse versus longitudinal dispersivity is 0.1. The figure shows the relative constituent concentration after 1000 days injection at 0.01 m3/day computed using the TVD method for advection. Other computation methods are immediately compared by changing the MXELM parameter on the MT3D sheet.

48 2D-Radial

The MT3DMS manual gives an example concerning 2D transport in a radial flow field. Solute is injected in a fully penetrating well. The problem is intended to test the accuracy of MT3DMS as applied to a radial

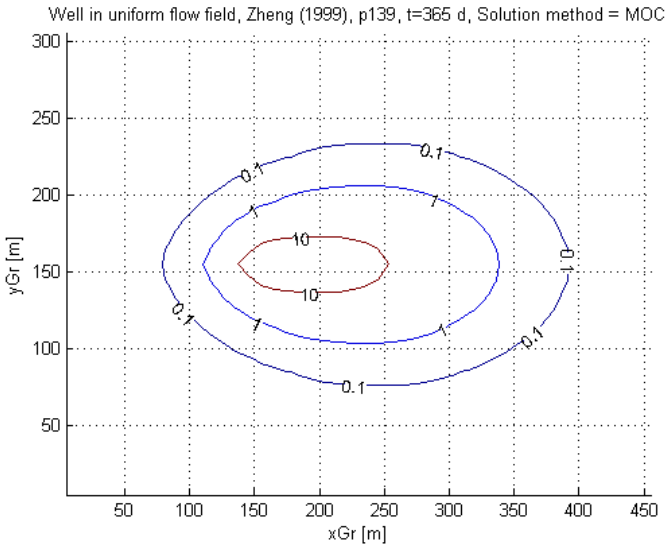


Figure 40: Concentration contours after 365 days of injection

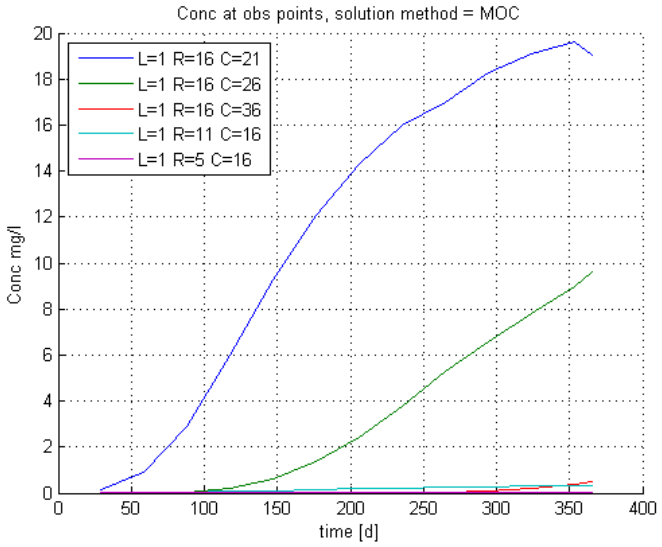


Figure 41: 2D-Uniform: Evolution of the concentration at different locations in the grid

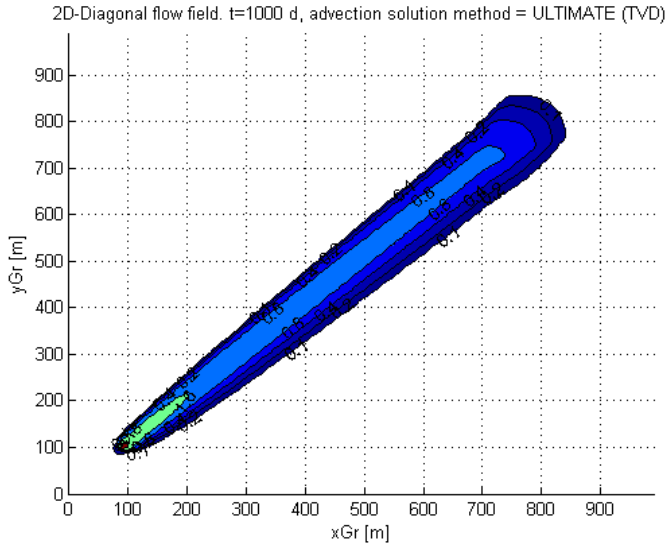


Figure 42: Diagonal flow field: Relative distribution of constituent after 1000 days injection in a uniform diagonal flow field

flow system. Figure 43 and 44. The assumptions are: constant injection, negligible ambient groundwater flow, aquifer is isotropic and homogeneous and of infinite extent and the flow field is steady state. Injection starts at t=0. The figures show the (relative) concentration after 27 days. Refer to the original manual of MT3DMS and to the script mf_adapt in the example directory for details and the numerical values used.

49 Salt test

This example was inspired by salt dilution tests done in extraction galleries in the Amsterdam Water Supply Dunes, Netherlands. The capacity of the 18 extraction galleries, 50 year old, 600 m long, concrete, gravel enveloped, 29-40 cm diameter "drains" has been investigated recently. Salt dilution tests have been used to measure the discharge at points in the galleries with an observation stand pipe. These points, however, are concrete boxes as shown in figure 45. Salt was entered in one of the upstream stand pipes and the electrical conductivity was measured continuously at downstream standpipes. Hence these sensors were always inside a box. The question was, to what extent the result depends on the location of the sensor in the box.

To this end a 3D model was made of 15x15x45 cells of size 4x4x4 cm each. The center 15x15x15 cells represent the box and the front side 5x5x15 cells and the back-end 5x5x15 cells represent a piece of the upstream and downstream gallery, which have a square cross section in this model. The cells surrounding the gallery have been made inactive.

Flow is prescribed at the left boundary and a constant head at the right hand side. Simulation divided into 3 stress periods. The first two are 5 seconds long and the last one 190 seconds. Salt is injected during the second stress period and traced through the model. The computed concentration in any model cell can be (has been) used as in a dilution test to compute the flow though the model. The idea was to determine to what extent the discharge is measurable with a sensor in any location of the model. The simulation yields nice pictures and as answer that any location is suitable.

Figure 48discharge measured at all points in the model. It shows that only near the injection point a sensor would not yield the correct total discharge through the model, which is due to a lack of mixing around

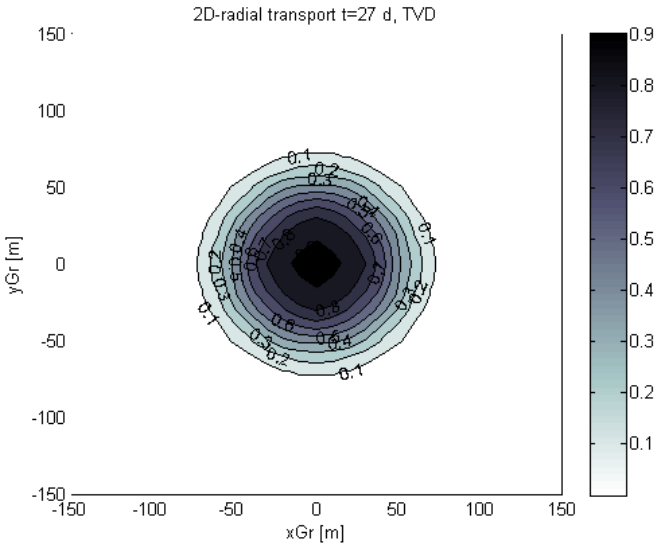


Figure 43: Injection with radial flow: distribution of constituent after 27 days of injection

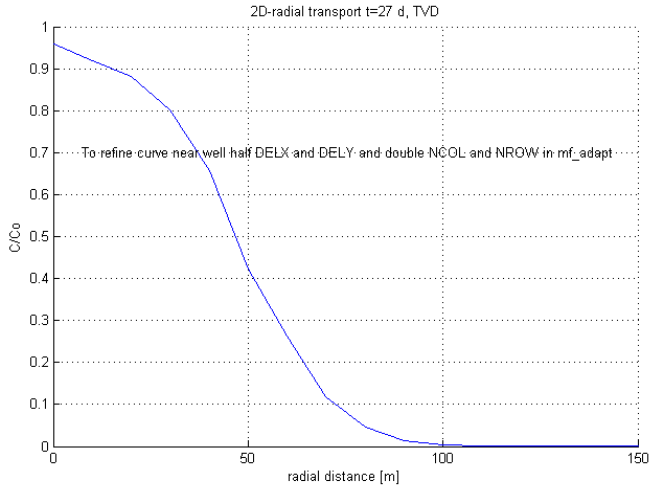


Figure 44: Injection with Radial flow: distribution of constituent after 27 days injection

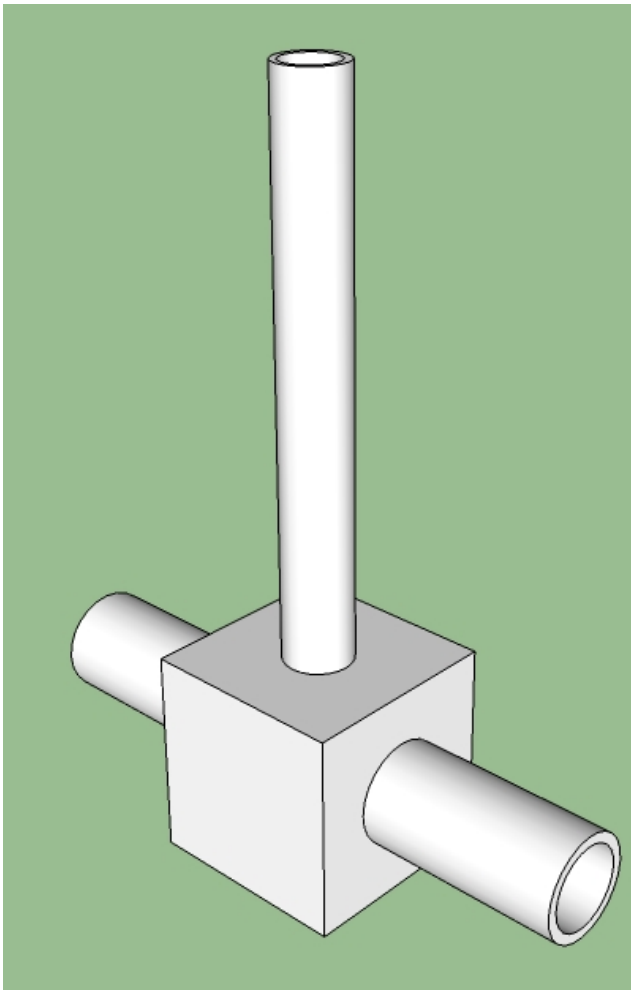


Figure 45: Drawing of gallery with connection box and standpipe for observations

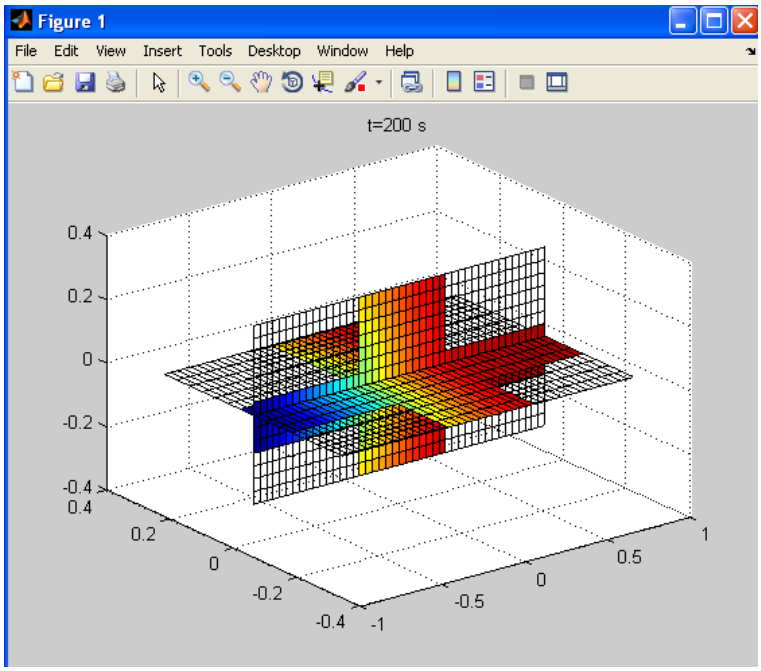


Figure 46: Salt dilution test: Salt distribution in the model after 200 seconds

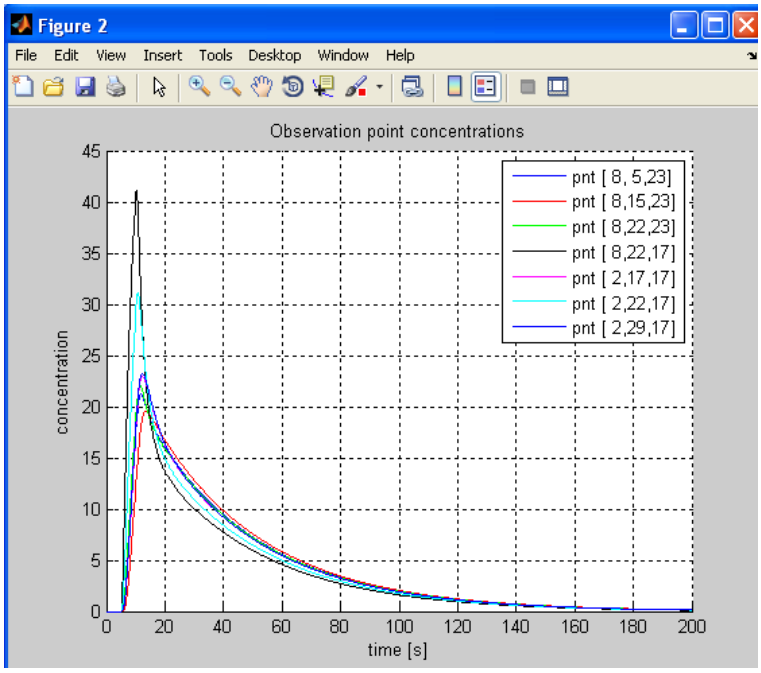


Figure 47: Salt dilution test, break-through curves for specific observation locations in the network

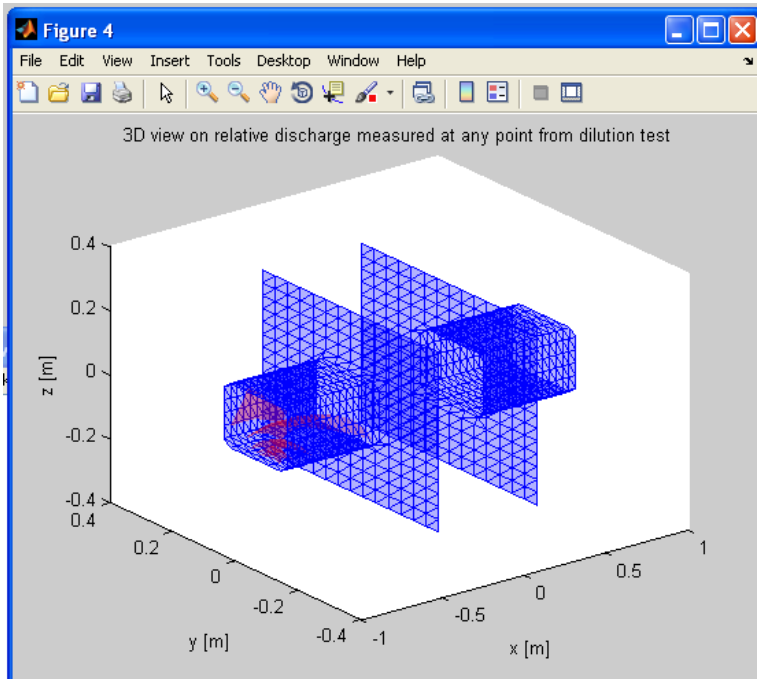


Figure 48: Salt dilution test, isoplanes showing in which part of the model the test would reveal a total flow less than the real total flow

the injection cell itself. Shown are iso-discharge surfaces, where discharge is the the discharge resulting from the salt-dilution test interpretation from a virtual sensor in every cell of the model.

Clearly, this model cannot really simulate the real situation, in which the flow is turbulent with whirls and, therefore, incomparable with the laminar flow simulated by MODFLOW and MT3DMS. Nevertheless it was a nice modelling exercise. It shows how efficiently such a 3D model can be built. It also shows some forms of post-processing, in this case analyzing the salt dilution test. It demonstrates the use of observation points in MT3DMS as well as the power and flexibility to analyze the dilution test for all cells of the model simultaneously.

To obtain a more realistic result, one for turbulent flow in the pipes and box, the model should be redone by a modelling package such as the multi-physics program *Comsol*, (www.comsol.com) which is able to solve the Navier-Stokes equations directly.

examples/swt_v4

A series of Benchmark samples presented by [9] have been implemented in *mfLab* and are presented in this directory. The examples are fully documented in their *mf_adapt.m* file.

50 The classic Henry problem

The classic Henry problem is presented and fully documented in the file *mf_adapt.m*. The files and the unit numbers have been chosen such that files can be exchanged one by one with the those in the corresponding example that comes with *swt_v4* as downloaded from the USGS site. This has helped me a lot with debugging.

Figure 49 shows the situation after 2 days (it is a very small problem with very high conductivity and a large discharge). I added the head contours (red vertical lines) and the stream function (curved yellow lines). Where the density is constant flow and head lines are perpendicular if the horizontal and vertical scale are equal. The stream function is obtained by integrating the horizontal specific discharge from the bottom of the model upwards. So each stream line represents the same total discharge between the bottom of the model and the line in question. Stream lines are also valid in density flow problems as long as the divergence of the flow is zero, which is the case in this vertical cross section. Integrating the horizontal discharge is done by

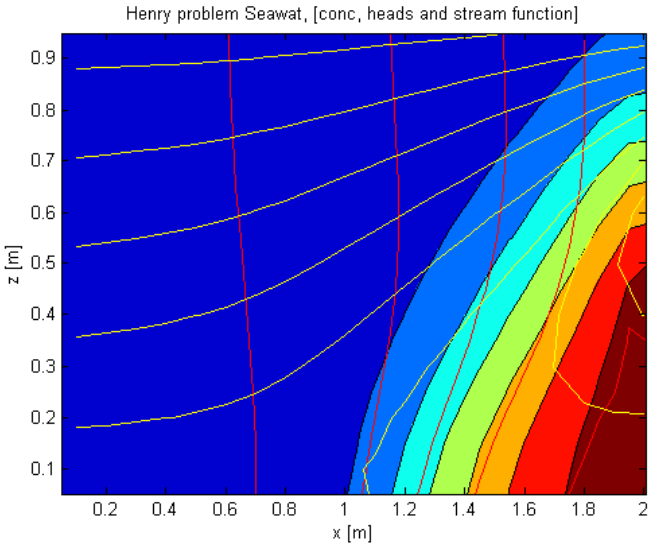


Figure 49: The Henry problem, freshwater heads (red), streamlines (yellow) and density as filled contours

summing the FLOWRIGHTFACE from the budget files along the verticals. The stream function is of great value especially in cross sections. The specific discharge can be readily determined from the stream lines in figure 49. As the stream lines show, salt water flows inward in the lower part of the right hand boundary. This flow is maintained by dispersion in the model, due to which there is a continuous discharge of salt water from the model that is compensated by the mentioned saltwater inflow at the lower part of the right hand boundary.

I have made some changes to the approach by [9]. In the first place, I used steady-state solution. Secondly, because the CHD package is used to specify the fixed-head boundary we don't need to specify where heads are fixed in IBOUND (no -1 cells in IBOUND). Thirdly, I included the CHDDENSOPT described by [9]pages 12-14.

51 The classic Elder problem

The classic Elder problem describes the flow in a cross section due to a high salinity at the top, which is caused by diffusion of salt from a fixed source. One of the results are shown in figure 50. The problem is extensively documented in the *mf_ adapt.m* file on the example directory.

52 Hydrocoin

The hydrocoin problem has been somewhat difficult as the GCG solver would not converge with the Cholesky decomposition method. However, it did converge with SSOR (see parameter PERCEL in the MT3D worksheet). It finally worked with good results that are shown in figure 51. I added some extra density lines compared to the figure in the SEAWAT manual and, especially the stream lines which are caused by the prescribed head at the top and which are influenced in the right half of the cross section by the density of the water that flowed over the salt dome at the bottom of the cross section The example is documented in its *mf_ adapt.m* file.

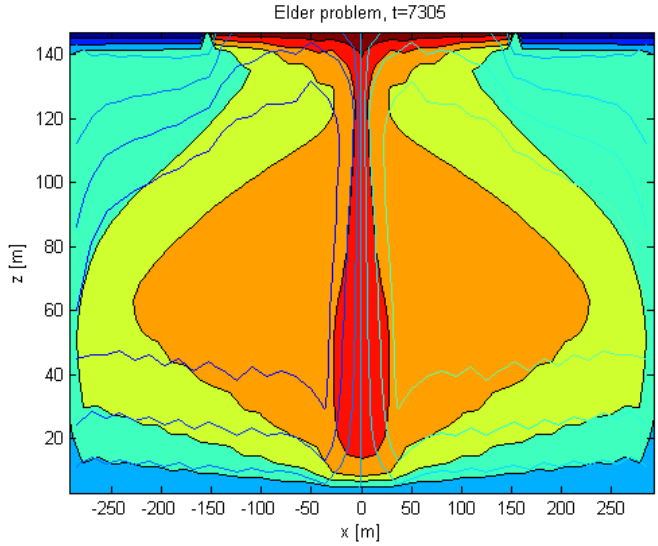


Figure 50: Result of Elder problem

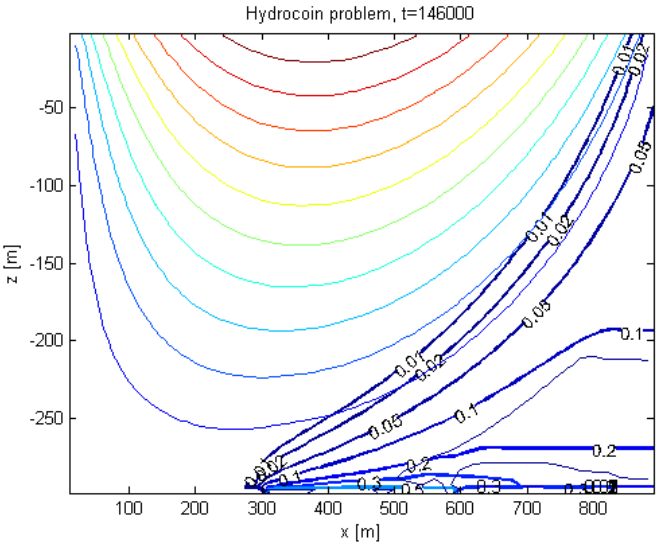


Figure 51: Results of hydrocoin example t=146000 d

53 Coastal flow

Figure 52shows the results of the coastal flow problem presented by [9], p23ff. The vertical cross section is bounded by the Ocean at the right while fresh water flows inward from the left. The ocean water has a temperature of 25C and the fresh water of 5C. The cross section is originally filled with cool ocean water, which is gradually displaced by the inflowing warm fresh water. The displacement is subject to density flow and viscosity variations due to the different temperatures. The displacement is shown in 4 steps of 10000 days. Langevin focuses in his figures mainly on the end situation, that is 400000 days in his computations. The figures here focus more on the dynamics of the displacement.

I added the stream function / stream lines and the point-water head lines as an extra illustration. The head lines show clearly the position of the interface. The blue lines are the 5, 50 and 95% salinity lines and the black vertical lines the 5, 50 95% temperature lines between the 5C of the fresh water and the 25C of the ocean water. Due to the exchange of heat between water and solids, the temperature displacement is about half as fast as the salt displacement. Also, the heat conduction causes a broader band between the 5 and 95% lines than is the case with the salinity.

The effect of the viscosity on the flow is signaled by the curvature of the read head lines as they cross the temperature change zone.

The problem was neatly encapsulated in *mfLab* by taking up the entire table of data used by Langevin in an extra worksheet in the workbook. The worksheet was named *"TableLangevin"*. The parameters for the layers could then directly be linked to this table. *mf_adapt* reads from the table what it needs to construct the model. This results in a very concise and intuitive model definition. Please refer to the files *mf_adapt.m*, *mf_analyze.m* and *coastal_flow.xls* in the example directory.

[9]build up this solution in 7 steps adding features from step to step. The example added to *mfLab* whose results are shown here, has all the processes and, therefore, is equivalent to example *coast7* of [9]. One can easily experiment with it. For instance, pressing the mt3dms.bat file that *mfLab* generated will run the MT3DMS with the same data files. That is, with the two species salinity and temperature, but without the density flow and viscosity feedback on the hydraulic conductivity. If the reaction package RCT is switched off in the NAM worksheet, there will be no delay of the temperature front, because sorption (exchange between water and solids) is no longer computed. One can also experiment with higher temperatures to see when fingering occurs due to large viscosity contrasts and the less viscous fluid displacing the more viscous one.

54 Other examples with Seawat

mfLab has many many examples of use of Seawat. The can be found in you working copy in the mflab/examples/swt_v4 directory. They are too many to elaborate them all here. Many also have anymations. Some of them are on the site under wiki

http://www.google.code/com/p/mflab
under WIKI/Animations

- Examples are
- Development of a freshwater lense below the Dutch dune area.
- Mechanism keeping the Okavango Delta in Botswana fresh.
- Chai Doab Pakistan natural saltwater development.
- Intrusion and upconing in the Dunes of the Amsterdam Water Supply
- Freshwater storage in the Desert of AbuDhabi
- Freshkeep (how to keep your well fresh when it attracts brackish water).
- etc.

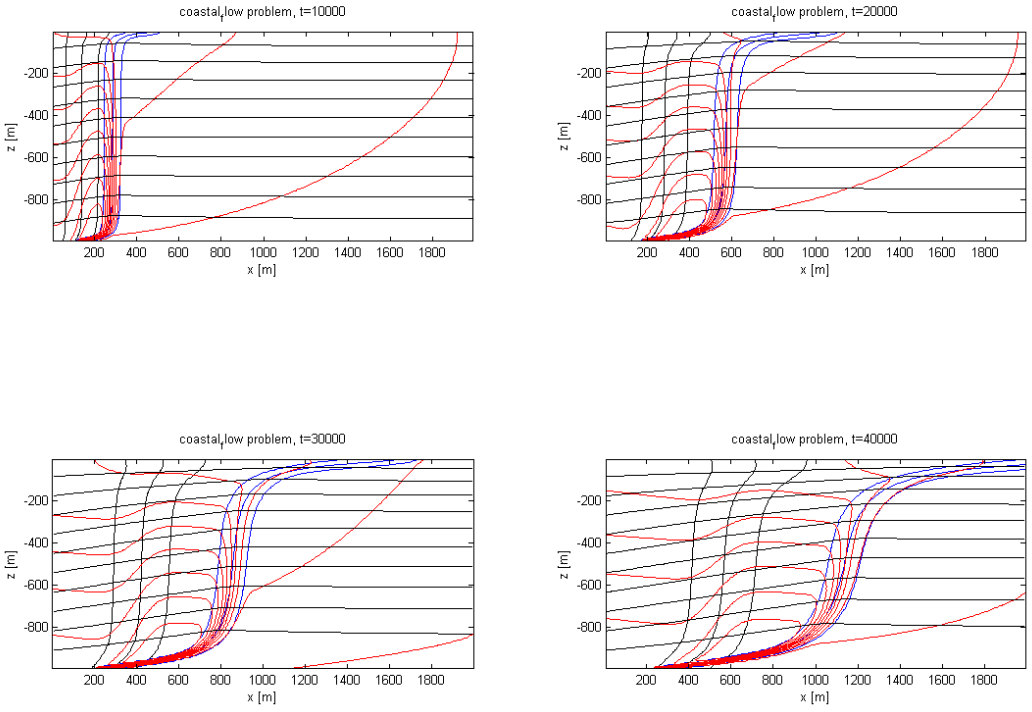


Figure 52: Coastal_flow problem presented by Langevin et al, 2008

examples/SWI

SWI is the so-called Salt Water Intrusion Package (see *http://bakkerhydro.org/*), which allows computation of density flow in existing models without redefining the grid. It works with interfaces, many of which may be defined, where the density may jump at an interface (stratified flow) or may vary linearly between two interfaces. There is no need to subdivide aquifers vertically, the package computes the variable horizontal flow components caused by the density distribution within the aquifers from the fresh-water head at the cell centers and the actual density distribution as defined by the interfaces and their positions. The package will track the position of these interfaces over time. Computing salt water intrusion this way in a large-scale regional model will cause very little computational overhead. In fact, it can be used in an existing model without redefining the grid in any way. A disadvantages is, however, that no dispersion can be taken into account (yet), salt cannot pass interfaces. Yet, the SWI package is an extremely useful tool for including density flow in many regional models. It is an virtually essential tool next to SEAWAT, which does require vertical mesh refinement but then computes dispersion correctly. SWI is free software. It can be obtained from *http://bakkerhydro.org/*. In fact, what is obtained is a version of *mf2k* with the package implemented. Hence, this version of *mf2k* can be used as your general *mf2k* version as it contains all other USGS packages but adds SWI. The accompanying manual also describes the examples. These examples have been modelled with *mfLab* and are presented hereafter. The examples are all documented using comments in their *mfLab* scripts *mf_ adapt* and *mf_ analyze*. Please refer to these scripts for details.

The examples below are described rudimentary for now. Refer to the manual from the mentioned website and the files *mf_ adapt* in the example directories that describe in detail the problems and the numbers used.

55 SWI example 1, rotating interface

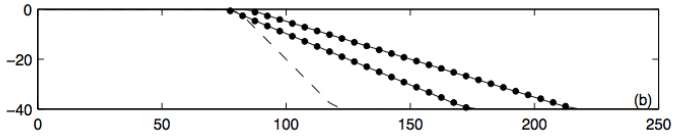
The first example is a rotating interface as shown in figure 53.

The interface positions computed with the *mfLab* implementation is given in the second picture of 53, showing the position of the interface at 100, 200, 300 and 400 days.

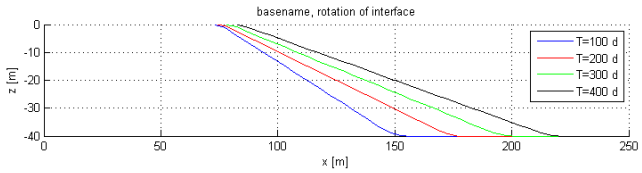
The results match. The second picture does not show the initial interface.

56 SWI example 2, rotating brackish zone

Figure 54from the SWI manual shows the setup of the second example and the results. The example concerns a rotating brackish front. The figure also demonstrates that there is very little difference between the results computed as a stratified density system and as a system in which the density varies linearly between two given salinity planes.



(a) from SWI manual



(b) *mfLab* results

Figure 53: Initial interface and position computed by SWI after 200 and 400 days

The first figure is from the SWI manual showing setup (a), comparison with SEAWAT (b) and comparison of the results of the stratified versus the non-stratified option of SWI. The second figure shows the results as computed via mfLab for the stratified option

The second figure of figure 54shows the results as computed through *mfLab* for the stratified option. To run the non-stratified option requires only changing the stratified switch in the accompanying worksheet for this problem.

57 SWI example 3

The third example from the user manual of SWI is a two layer system in which the top layers is in direct contact with the ocean floor at x<600 m. The situation is shown in figure 55taken from the SWI manual. The figure shows the initial interface position as a dashed line with the computed positions at different times by continuous lines. See the SWI manual and *mf_ adapt* in the example directory for more details and the numbers used.

The second figure of figure 55 shows the results computed through *mfLab*. See *mf_ analyze* in the example directory for the methods used to visualize the results.

58 SWI example 4, coast with a well

Example 4, figure 56, concerns two coastal aquifers with a confining bet in between where the top aquifer is in full contact with the ocean floor over a distance of 300 m. The situation is thus similar to that of example 3. However, there is a well inland extraction water. Due to the combined effect of density flow and the extraction, the interface moves inland and finally up-coning occurs as shown in the results in the lower part of figure 56. For details and the used numbers see the description in the SWI manual and *mf_ adapt* in the example directory.

59 SWI example 5, square island with well

The fifth and last example in the SWI manual concerns a square island surrounded by ocean. Its setup is shown in figure 57. Recharge causes a fresh water lens to develop and maintain, while water is extracted somewhat artificially over part of the area of the island as indicated in the figure below that was taken from the SWI manual.

Deformation of the freshwater lens occurs due to extraction over the area in the Northwest of the island. The problem is described in detail in the SWI manual and also in the local *mf_ adapt* script in the example directory.

Figure 58 shows the results as computed through *mfLab*. The figure shows the contours of the elevation of both interfaces and in a cross section.

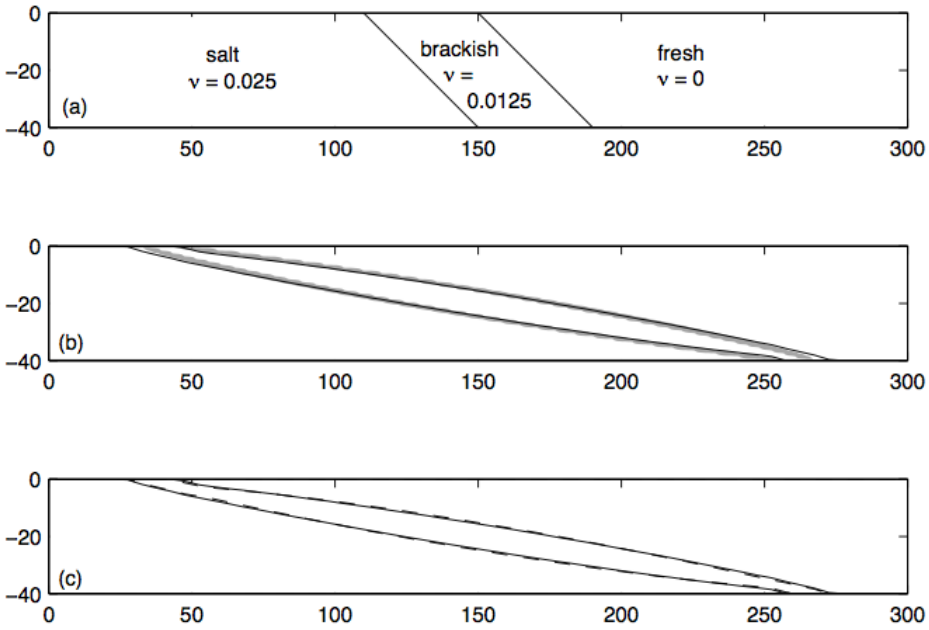
examples/mf2005

No examples yet. But I plan to include examples with the new interesting Conduit Flow Package.

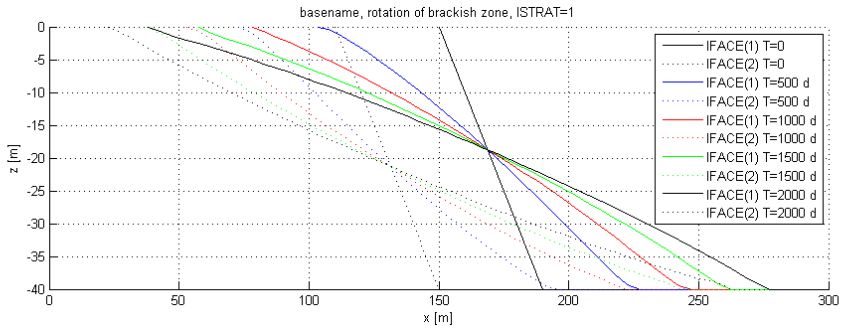
References

[1] Bakker, M & F.Schaars (2005) The Sea-Water Intrusion (SWI) Package Manual, Part 2, Module Documentation, Version 0.2. http://bakkerhydro.org/swi/swidownloads.html

[2] [Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.]

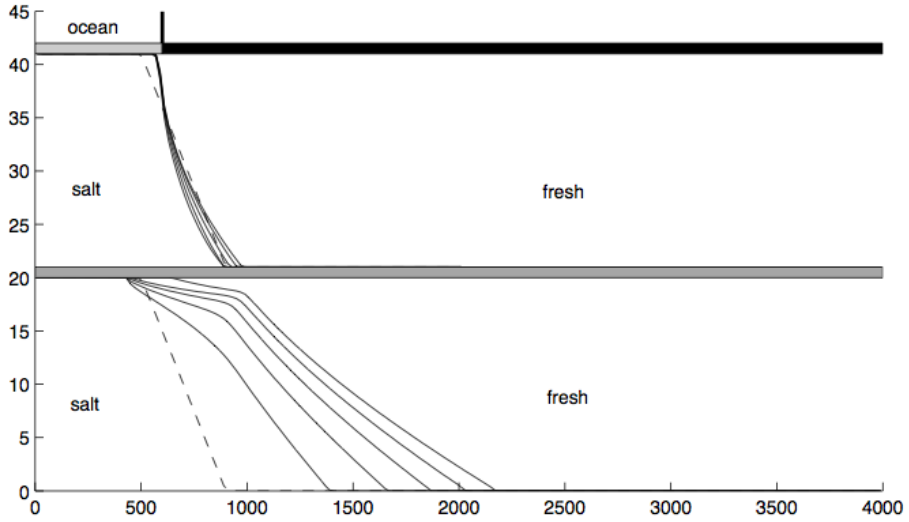


(a) SWI-manual

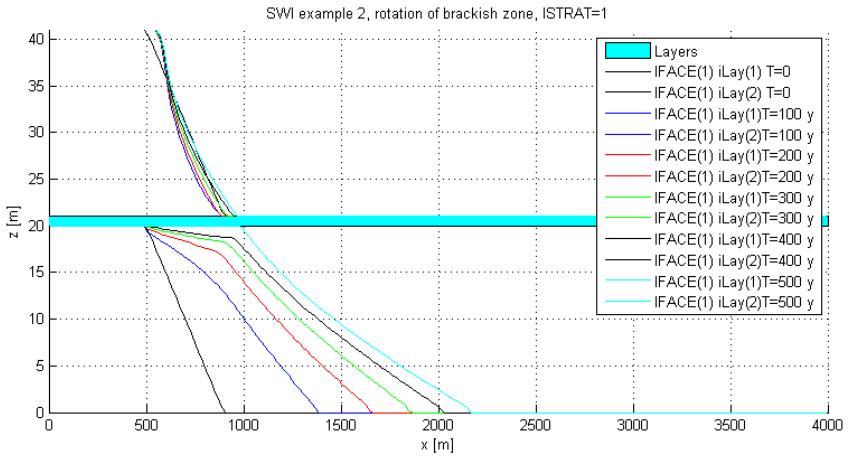


(b) mfLab

Figure 54: Results of brackish zone movement computed by SWI through mfLab for the stratified option

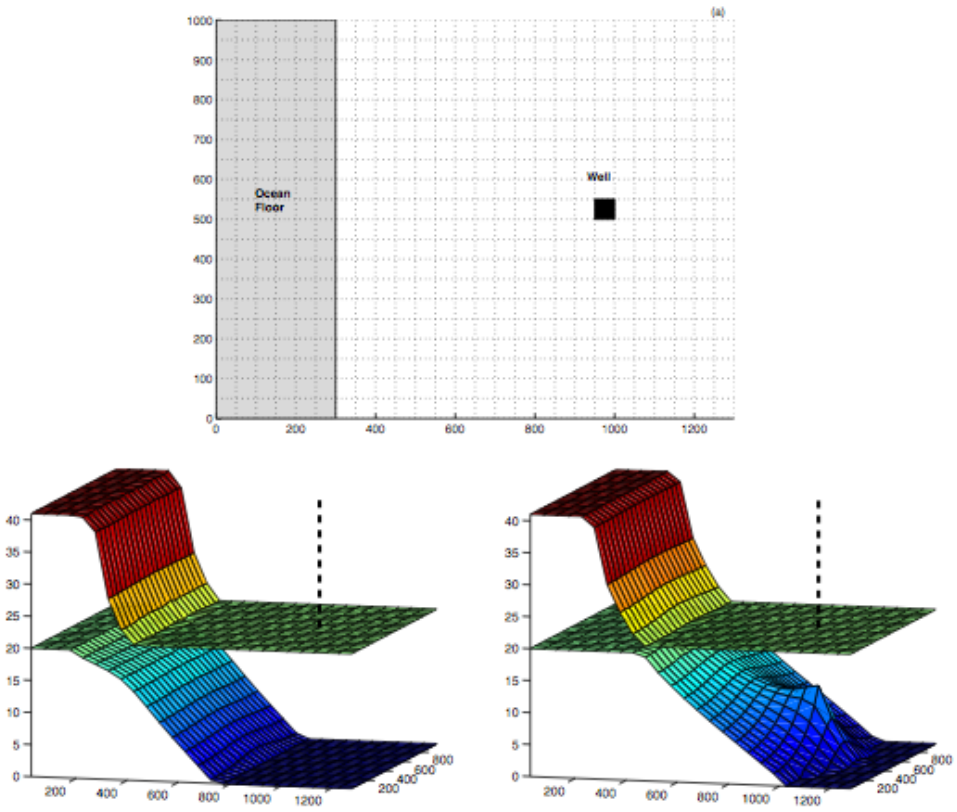


(a) SWI-manual, interfaces at 100 year intervals

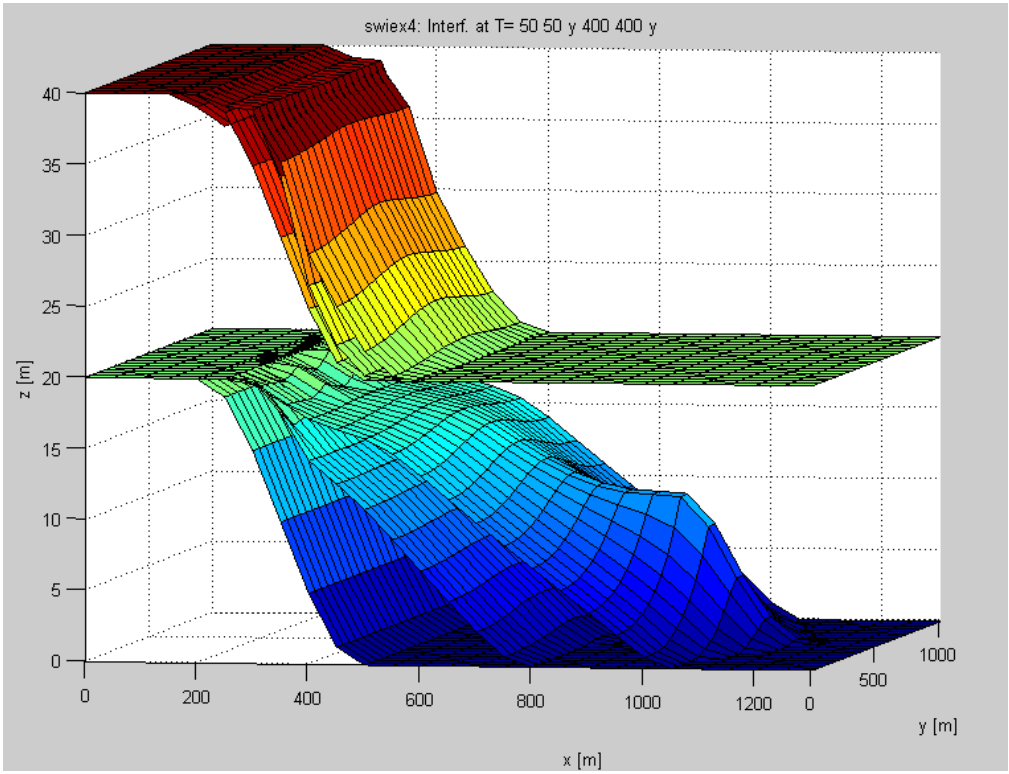


(b) *mfLab*

Figure 55: Initial position and position of the interface at 100 year intervals as computed through *mfLab*



(a) SWI-manual: setup, initial interfaces and results of intrusion and extraction after 400 years



(b) *mfLab*

Figure 56: Interface movement and up-coning as computed through *mfLab*

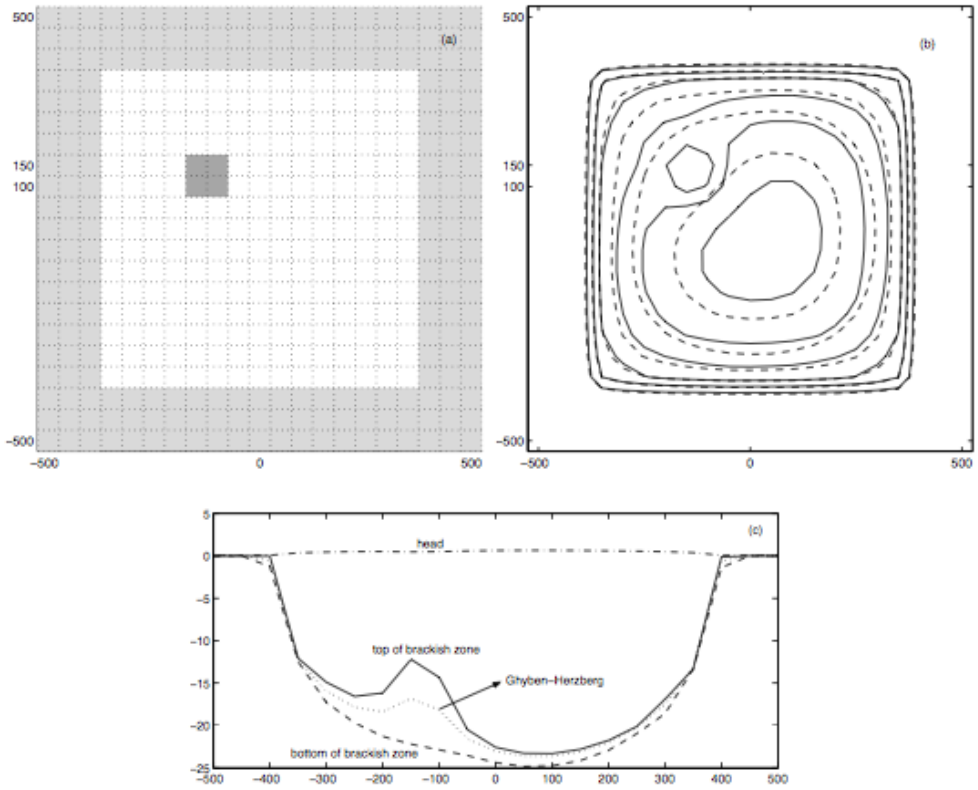
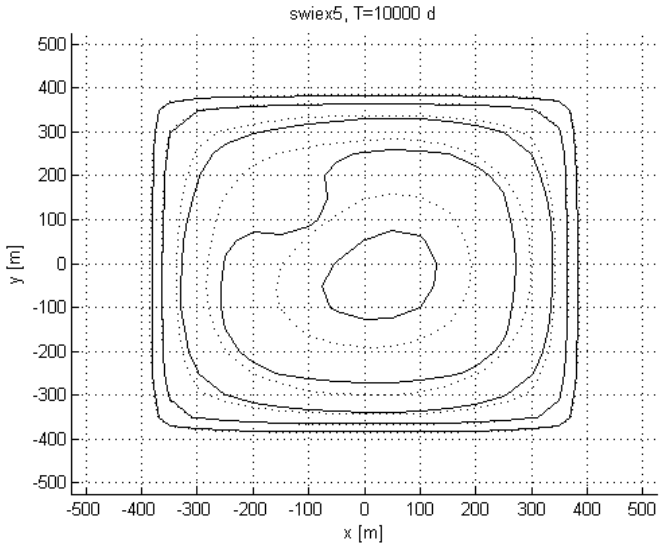
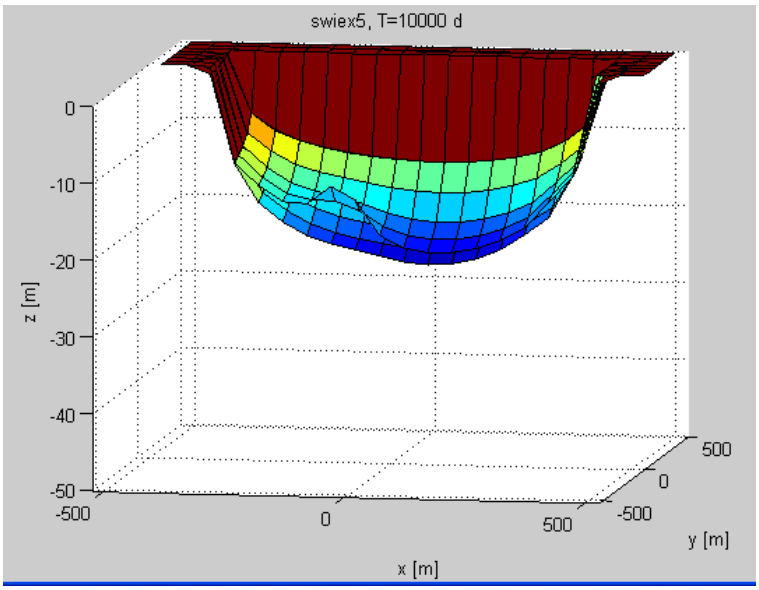


Figure 57: SWI example 5, problem from manual, square island with extraction



(a) *mfLab*, depth contours of the elevation of both interfaces after 1000 years



(b) 3D-cutoff showing the two interfaces after 1000 years in 3D

Figure 58: SWI ex5: Computation through *mfLab* visualized as depth contours and 3D-cutoff

[3] [Bakker, M & F.Schaars (2005) The Sea-Water Intrusion (SWI) Package Manual, Part 1, Theory, User Manual and Examples, Version 1.2. http://bakkerhydro.org/swi/swidownloads.html]

[4] McDonald, M.G. & A.W. Harbaugh (1988) A Modular Three-Dimensional Finite-Difference Ground-Water Flow Model. Techniques of Water-Resources Investigations of the United States Geological Survey. Book 6, Modeling Techniques. Chapter A1. This chapter supersedes the US Geological Survey Open-File Report 83-875.

[5] Harbaugh, A.W., E.R. Banta, M.C.Hill and M.G. McDonald (2000) MODFLOW-2000, the U.S. Geological Survey Modular Ground-Water Model - User guide to modularization concepts and the ground-water flow process. Open-File Report 00-92. US Department of the Interior, U.S. Geological Survey.

[6] Anderman, E.R. & M.C. Hill (2000) MODFLOW-2000, The U.S. Geological Survey Modular Ground-Water Model - Documentation of the Hydrogeologic-Unit Flow (HUF) Packages. US Department of the interior. US Geological Survey. Open File Report 00-342.

[7] W.B. Shoemaker, E.L. Kuniansku, S.Birk, S.Bauer and E.D. Swain (2007) Documentation of a Conduit Flow Process (CFP) for MODFLOW-2005. US Department of the Interior, US Geological Survey. Techniques and Methods, Book 6, Chapter A24.

[8] Guo W & C.D. Langevin (2002) User‘s guide to SEAWAT: A computer program for simulation of Three-Dimensional Variable-Density Ground-Water Flow. Techniques of Water-Resources Investigations of the US Geological Survey. Book 6, Chapter A7.

[9] Langevin, C.D., D.T. Thorne, A.M. Dausman, M.C. Sukop, W. Guo (2008) SEAWAT version 4: A computer program for simulation of Multi-Species Solute and Heat Transport. Techniques and Methods Book 6, Chapter A22. US Department of the Interior. US Geological Survey.

[10] Langevin, C.D., W.B Shoemaker & W. Guo (2003) MO FLOW-2000, the US Geological Survey Modular Ground-Water Model - Documentation of the SEAWAT-2000 version with the Variable Density Flow Process (VDF) and the Integrated MT3DMS Transport Process (IMT). US-Geological Survey Open-File Report 03-426. US. Geological Survey Office of Ground Water, Tallahassee, FL.

[11] Thoms, R.B., R.L. Johnson and R.W. Healy (2006) User’s guide to variably Saturated Flow 9VSF) Process for MODFLOW. US Department of the Interior. US Geological Survey. Techniques and Methods 6-A18.

[12] Zheng, C. & P.P.Wang (1999) MT3DMS: A Modular Three-Dimensional Multi-Species Transport Model for Simulation of Advection, Dispersion, and Chemical Reactions of Contaminants in Groundwater Systems; Documentation and User‘s Guide. University of Alabama for US Army Corps of Engineers.

[13] Zheng, C. (2006) MT3DMS v5.2. Supplemental User‘s Guide. Department of Geological Sciences, University of Alabama.

[14] Hsieh, P.A. & R.B. Winston (2002) User‘s Guide to Model Viewer, a program for Three-Dimensional Visualization of Ground-Water Model Results. Open-File report 02-106. US Department of the Interior. US Geological Survey. Menlo Park, California.

[15] Zheng, C., M.C. Hill & P.A. Hsieh (2002) MODFLOW-2000, the US Geological Survey Modular Ground-Water Model & User Guide to the LMT6 Package, the Linkage with MT3DMS for Multi-Species Mass Transport Modeling. Open-File Report 01-82. US Department of the Interior, US Geological Survey.