User guide for ***mfLab***
11 Aug 2010

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

August 11, 2010

Contents

Abstract

mfLab is an open software environment for effective groundwater modeling by combining Matlab with the MODFLOW suite of groundwater simulators. A working copy can be obtained from http://code.google.com/p/mfLab by a checkout with Subversion.

mfLab stands for "MODFLOW Laboratory". As such 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*. This is done 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 processed and visualized 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 I will stick to it for practical reasons, which are that the University of Technology Delft, where I teach, has a site license so that all students and staff have free access to it and are familiar with it.

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,

Chapter 1

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*), 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 of codes capable of modeling virtually every groundwater situation. These programs, 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. The open-source philosophy has caused that there is no foreseeable end to the development of the *mf++* suite of programs and packages. *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.
- 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.

Chapter 2

How does *mfLab* work?

2.1 Basic philosophy

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

This 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 as straightforward to build a 1000x1000x100 cell model as one of 10x10x2 cells. Matlab helps transparent construction of arbitrary complex models, which read data from databases, pre-process it as required to prepare the 3D model arrays.

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 making 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 to remember) parameters and flags 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, by at this moment I'm not aware of viable alternatives, although Open Office may be an option). The Excel workbook has many advantages: It is multi-page and, therefore, suitable to store sets of parameters in an ordered and accessible way. It 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 an environment in which one can do any computations, 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, which pop up when hovering over them with the mouse. This way, this parameter workbook also serves as 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, you seldom need to worry about parameters between runs.

A model in *mfLab* normally consists of 3 local files. Two of them are Matlab *m-files* containing Matlab code, and the third one is the mentioned Excel workbook. The two local m-files have fixed names: *mf_ adapt.m* and *mf_ analyze.m*, while the Excel workbook has a local name of your choice indicated further as *<<basename>>*, hence *<<basename>>.xls*.

The *<<basename>>* is set as the first line 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.

However, at any moment 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; 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 with 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, carry out a lot of pre-processing etc and running external programs like statistical codes to prepare the required input. However, realize 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 extra great tool when stumbling over errors and unexpected results. It allows setting breakpoints and 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 the 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 few lines to the end to *mf_ adapt.m*, without touching the just constructing 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, WEL, DRN, RIV, GHG and some more. These worksheets are described elsewhere in this manual. This Excel workbook has the name *<<basename>>.xls* and is local. As is the case with the two scripts *mf_ adapt.m* and *mf_ analyze.m*, the contents of <<basename>>.xls hardly changes between models, so that starting with an existing one is the best way to start a new model. Chances are that, except the stress periods and number of layers little has to be changed between models.

The Excel workbook will generally contain many more parameters than is needed by a specific 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 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. You are free to add whatever you like to the workbook, as *mfLab* only extracts what it needs. This facilitates documentation and experimentation. One way to experiment is to for instance copy an entire worksheet like MFLOW to MFLOW(2) and change MFLOW as desired. MFLOW(2) keeps the old MODFLOW parameters while in the new run *mfLab* only extracts the parameters from the worksheet MFLOW. On the other hand if you only want to see the parameters and sheets for the model you are now constructing, just hide the worksheets, columns and lines to remove distracting clutter from sight without deleting anything from the workbook. Hence using a workbook as a multi-page parameter container is quite flexible and helpful.

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

You can use all Matlab's and *mfLab*'s functions to further rework, show or analyze the results in an interactive way. Adding your refined lines to the *mf_ analyze.m* makes sure your analysis is automatically executed in every future run.

Chapter 3

How to get *mfLab* to work?

3.1 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 preinstalled 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 com-

mand. 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.

3.2 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***
  - *read*
  - *write*
  - *gridcoords*
  - *etc*
  - *fdm*
- ***examples***
  - *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_setup.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)

3.3 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*).:

addpath('C:\GRWMODELS\mflab\mfiles\read');

addpath('C:\GRWMODELS\mflab\mfiles\write');

addpath('C:\GRWMODELS\mflab\mfiles\gridcoords);

addpath('C:\GRWMODELS\mflab\mfiles\ect);

addpath('C:\GRWMODELS\mflab\mfiles\fdm);

Here, "C:\GRWMODELS" will be different on your computer so you must replace it accordingly.

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 3.2.

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 use when it is called.

3.3.1 Make the location of the executables known to *mf_setup*

To launch the *mf++* executables, *mfLab*, i.e. its script *mf_setup* has to know about them. Therefore, *mf_setup* calls the script "*setExeculables*.m" which

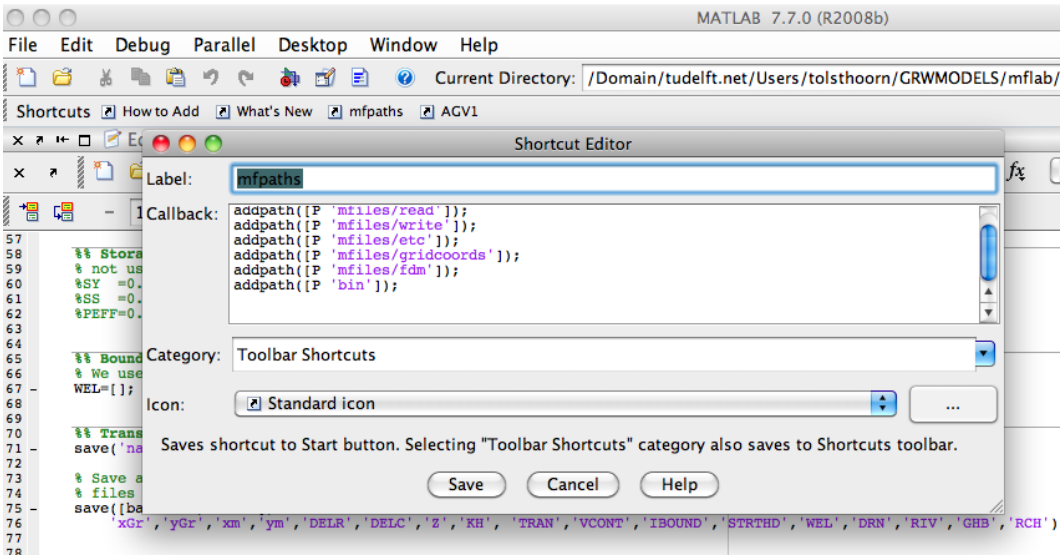


Figure 3.1: Make a shortcut on the shortcut toolbar to set the paths to the mfLab directories

is shown in figure 3.2. This script resides in the *mflab/mfiles/write* directory together with *mf_ setup.m*.

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.

If *mfsetpath* didn't work out as it should, you must replace the path to the models in *setExecutables.m* yourself.

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. 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.

```
% 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 3.2: Contents of the script *setExecutables.m* called from *mf_setup.m* line +/- 142

3.4 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_setup* 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_setup* 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_setup.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_setup 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_adapt*. *mf_setup* reads out the *<<basename>>.xls* 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_setup*.

mf_setup 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_setup* 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_setup 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_setup.m*.

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

3.5 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_ setup* 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.

3.6 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 3.3: 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.

3.6.1 Using xlsread

In Matlab, *xlsread* is called as follows when aiming to read Excel worksheets. Note that the behavior on Windows and non-Windows computers is a bit different. First, on computers without the Excel Com Server, run xlsread in "basic" mode (see below), on Windows computers leave out the "basic" argument, so the Excel Com Server will be used.

Next, on non-Windows computers (in basic mode), the Raw and Txt arrays include the columns to the left of the first empty and non-numeric Excel cell respectively, whereas in Windows computers they do not. On non-Windows computers the Raw array also contains the empty rows above the first non-empty cell of the Excel worksheet, whereas under Windows the Raw array does not. In short, on Windows computers the Num, Txt and Raw array never contain rows and columns outside the range in the Excel worksheet that respectively are empty, have no numeric values or have only numeric values.

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". 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.

3.6.1.1 On Windows computers with the Excel Com Server installed (Excel installed)

[*Num*,*Txt*,*Raw*]=xlsread(excelfilename,sheetname); % on windows machines with the Excel Com Server

excelfilame is the name of the Excel file with or without extension (.xls, .xlsx)

sheetname is the name of the worksheet in the Excel file.

Raw is a cell-array whose contents matches the rectangular range in the Excel excel sheet which just includes all non-empty cells.

Txt is a cell-array whose contents matches the rectangular range in the Excel worksheet, which just inclues all non-numeric cells. Numeric cells within this range become empty cells in the Matlab cell array.

Num is a numeric array whose contens matches the rectangular range in the Excel worksheet, which just includes all numeric cells. Non-numeric cells within this range become NaN (Malab's numeric "Not a Number").

Usage Having read the arrays Num, Txt and possibly Raw proceed as follows:

If the table has headers and no first column of labels Headers=Txt(1,:); clear txt

If the table has a a first colum of labels and a top row of headers Headers=Txt(1,2:end);

Labels =Txt(2:end,1); clear txt

Then the labels will match the lines in the Num array and the Headers will match its columns.

3.6.1.2 On Mac and other computers without Excel Com Server

[*Num*,*Txt*,*Raw*]=xlsread(excelfilename'sheetname,'*basic*');

The behavior of xslread on the Mac is a little different. The Raw and Txt arrays now comprehend the range of of cells in the Excel worksheet between cell A1 (upper left) and the last non-empty, or text cell, where the Num array still contains onlye the range of the Excel worksheet that has any numeric value in it.

Raw is a cell-array whose contents matches the rectangular range in the Excel excel worksheet between the left most corner (cell A1) and the last non-empty cell.

Txt is a cell-array whose contents matches the rectangular range in the Excel worksheet, which includes all non-numeric cells and which starts in the first column and at the row of the first non-numeric cell. Numeric cells within this range become empty cells in the Matlab cell array.

Num is a numeric array whose contens matches the rectangular range in the Excel worksheet, which just includes all numeric cells. Non-numeric cells within this range become NaN (Malab's numeric "Not a Number").

Usage Having read the arrays Num, Txt and possibly Raw proceed as follows:

If the table has headers and no first column of labels i=find(~isnan(Num(1,:)),1,'first'

Num=Num(:,i:end);

Headers=Txt(1,i:end); clear Txt;

If the table has a a first colum of labels and a top row of headers
i=find(~isnan(Num(1,:)),1,'first');

Num=Num(:,i:end);

Headers=Txt(1,i:end);

Labels=Txt(2:end,i-1); clear Txt

Then the labels will match the lines in the Num array and the Headers will match its columns.

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

It seems incredible in 2010 that FORTRAN compilers have no standard for the unformatted files they produce. This means that if MODFLOW is compiled with two different FORTRAN compilers both executables may produce unformatted output files of heads, draw-downs, and budget flow terms, as well as concentrations files that are different for the same problem. The USGS tries to make sure that the unformatted files produced by the Windows executables on down-loadable from their site only contain the bytes instructed by the code and that the include no extra brand specific "compiler features". However success is not guaranteed when using "standard FORTRAN", it just isn't standard from the point of view of their unformatted output. No doubt, Matlab can readout any unformatted file, but to make sense of it, it needs to know its structure.

*mfLab'*s unformatted file read functions *readDat*, *readBud* and *readMT3D,* which are used to read out the unformatted output files con tainting the computed heads, draw-down, budget and concentrations have been crated to tackle some of this potential incompatibilities. So normally it should not matter from which executable the unformatted output files stem for these functions to be able to read them properly. However, there is no guarantee that it will work with code from any FORTRAN compiler.

The functions start assuming that these files are binary, i.e. that they contain no extra bytes other than instructed by the FORTRAN code. This is what the USGS codes prefer and is standard for their Windows executables.

If this does not work, these functions try to figure out what's different compared to this standard. It assumes that there will be an extra number at the start and end of every record that the code writes and it tries to figure out how many bytes this number occupies. Until now this works well.

It does so by assuming that there are no extra bytes. It then tries to read

the first data label. If there are extra bytes in the file, the first bytes of the so read label are incorrect. It then relies on the experience of all unformatted files encountered until now, that the label is preceded by a null-byte. Hence it looks for the position of the last null byte in the read label. If there are no null bytes, than it's a standard USGS binary file, if there are null bytes, then the position of the last one provides the number of extra bytes in front of each record. The function then assumes that there are an equal number of extra bytes at the end of each written record, it checks to see of, using these bytes the file length is an integer multiple of a complete layer record, and, if so, proceeds reading taking into account these extra bytes. Normally, these extra bytes contain the Lent of the record, which may be useful to jump from record to record in cases where nothing is known about the file structure beforehand, so it then is a feature, but mfLab knows the file structure and simple ignores such bytes in reading the unformatted files. Clearly, if the programs are compiled with FORTRAN compilers using yet another scheme of undocumented features, it likely won't work. Also, if, for some reason the byte just before the first data label in the file is not a null-byte, this procedure will also fail. But generally users don't have to worry about the structure of the unformatted files, because output from MODFLOW and MT3DMS produced on a Windows computer can now also be read on a Mac or Unix computer and vice versa.

Chapter 4

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

4.1 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 4.1). 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.4.2. 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 $T = 0.02ft^2d$ 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

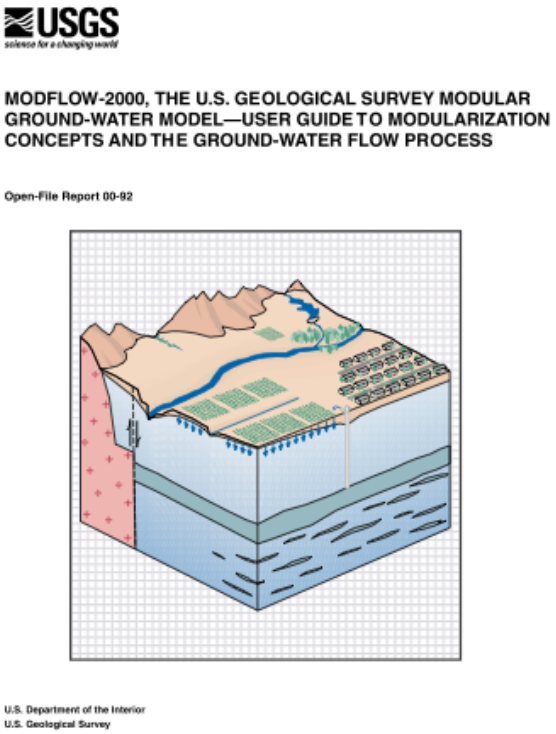


Figure 4.1: USGS: Open file report - MODFLOW2000 User Guide

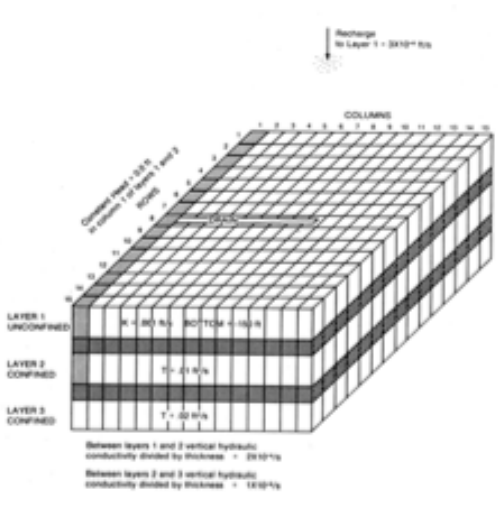


Figure 4.2: Model of example 1

drain.

4.2 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*.

We may start with specifying our model grid. This can be done by specifying the coordinates of the grid lines xGr and yGr.

xGr=(0:15)*5000;

yGr=(0:15)*5000;

Then we may compute the cell center coordinates, the cell sizes and number of cells using a convenient *mfLab* function *modelsize*:

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

It should be clear that all variables are vectors.

Look in the function *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, xGr is increasing while yGr is decreasing. This ensures that the top line of a printout of a model layer (or when you scroll the model layer matrix on the computer screen) corresponds with the highest yGr coordinate, as you would intuitively expect it.

Furthermore, the elements in xGr and yGr after having past *modelsize*() are sorted and duplicates are removed. This means that you may provide grid coordinates in any order and including duplicates. This something is very convenient, for instance, if a mixture of grids and finer sub-grids around individual objects have to be specified and 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 when used.

modelsize() also yields the coordinates of the cell centers, xm and ym, the size of the cells, Dx and Dy and the number of cells of the model, 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() *modelsize* It has a 3D variant *modelsize3*, which works as follows*:*

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

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).

IBOUND-array IBOUND=ones(*NY,NX,NZ*);

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

IBOUND has the size of the model and all values one. In the second step the fixed head cells of IBOUND are set to -1 as is required by MODFLOW. These are the cells in all rows, the first column and layers 1 and 2.

Orientation of Matlab arrays compared to MODFLOW arrays Note that all 3D arrays in *mfLab* are [*Row*, *Column*, *Layer*] oriented. This is the natural orientation in Matlab and is really most convenient and general to use, especially in the Matlab environment. For Matlab the Rows are the first dimension, the Columns are the second dimension and the Layers 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. It's also the sequence used in mathematics and linear algebra.

Writing the arrays correctly to the input files for MODFLOW is dealt with by the *mfLab* using the functions that you can find in the *mfLab/mfiles/write directory*. But you should never have to deal with them.

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 must be specified? In general, exactly those variables that are required by MODFLOW and the other programs you want to run. *mfLab* does its most to stick as closely to the original manuals as possible.

For instance, if the BCF packages is use, you may have to specify transmissivities, called TRAN in the manual and so *mfLab* expects to find an (Nrow,Ncol,NLay) sized matrix 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.

The names *mfLab* looks for can be found in the top of the *mf_setup.m* script, directly after the call to *mf_adapt*. *mf_setup* is the script to launch *mfLab*. In general, *mf_adapt* defines the grid and all 3D matrices constituting the model (see figure 4.3). 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*.

4.3 The accompanying Excel workbook *ex1.xls*

Every *mfLab* model has consists of at least three files, *mf_adapt* to construct the model, *mf_analyze* to extract, interpret and visualize its results and the workbook *<<basename>>.xls* to hold 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,

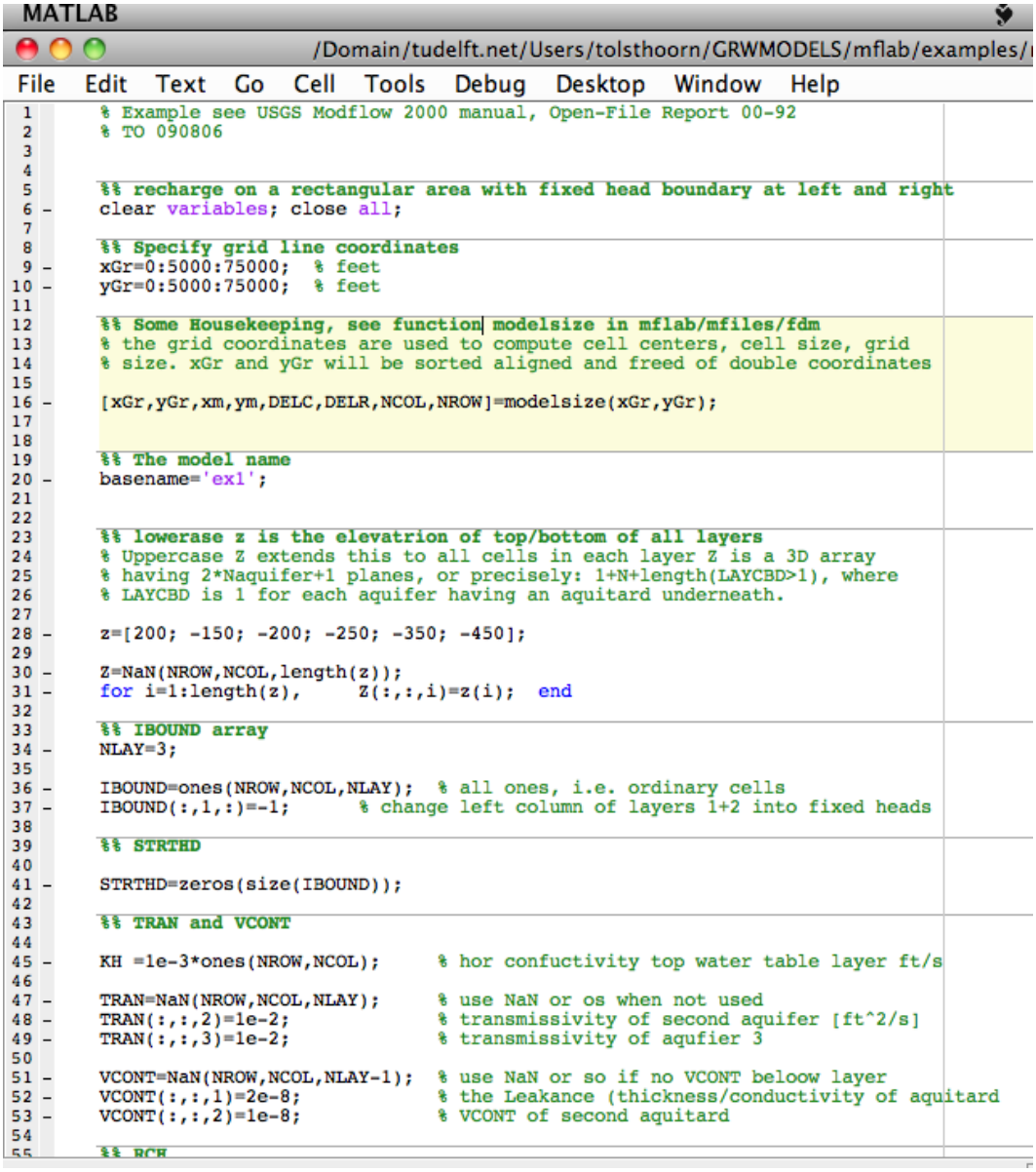


Figure 4.3: First part of script mf_adapt defining the model

refer to the the worksheet description elsewhere in this user guide.

4.3.1 Inspecting the accompanying Excel workbook

The workbook contains parameters relating to the model. These parameters are arranged in worksheets within the workbook. Figure 5.2 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.5.6) 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 5.7) 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 5.8 and 5.9). 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.

4.4 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 4.4).

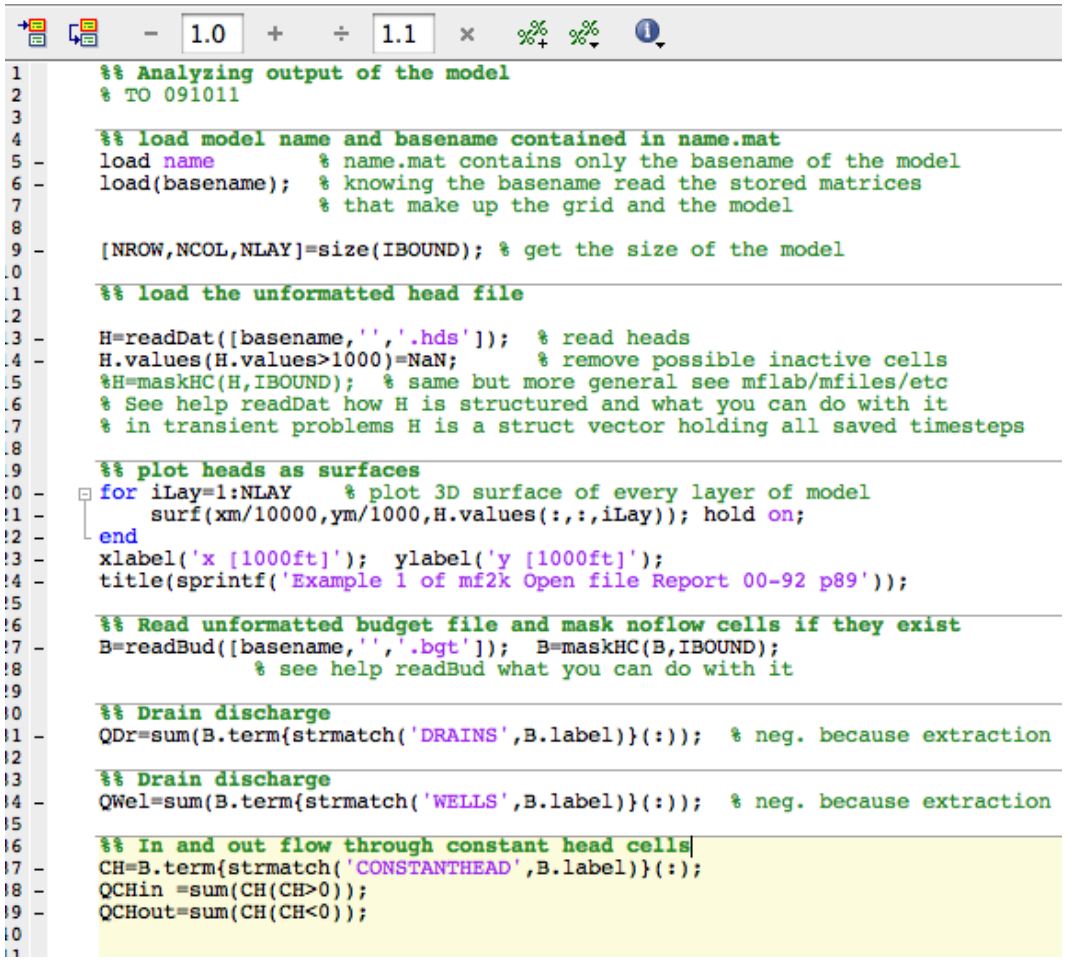
It is possible to invoke *mf_ analyze* automatically after running the models. While this may be convenient for production runs, it is generally not a good idea when the model is still under development. It is then preferred to let the model finish first, and check to see that it has terminated normally 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_ setup*, 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 4.4: 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.

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

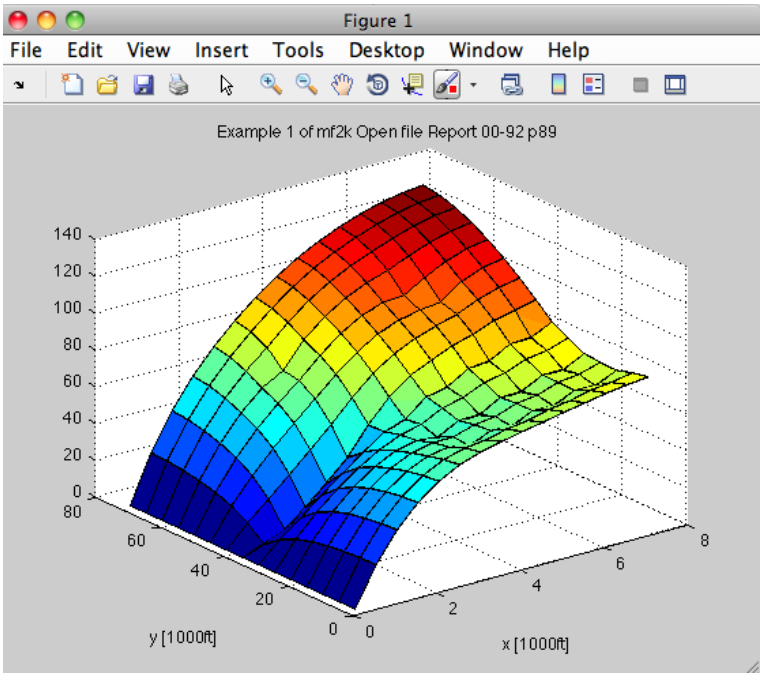


Figure 4.5: Water table shown as surface

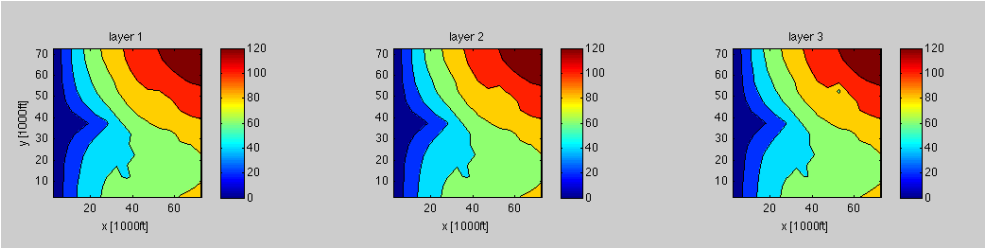


Figure 4.6: Head contours of the three aquifers

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.

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 4.7: 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 4.4 and 4.7ibelonging to this section.

Notice that the structs produced by *readDat* (reading heads and drawdowns) 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.

Chapter 5

Short description of the worksheets in the Excel workbook

5.1 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 5.1shows different worksheets in an an accompanying *mfLab* workbook.

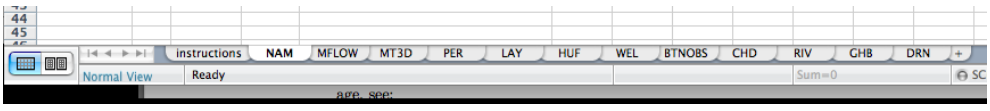


Figure 5.1: Bottom of Excel screen showing the tabs of the different worksheets in an accompanying *mfLab* Excel workbook

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.

5.1.1 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.

5.1.2 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.

5.1.3 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.

5.1.4 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.

5.2 Nam worksheet

The NAM worksheet, an example of which is shown in figure 5.2, 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:

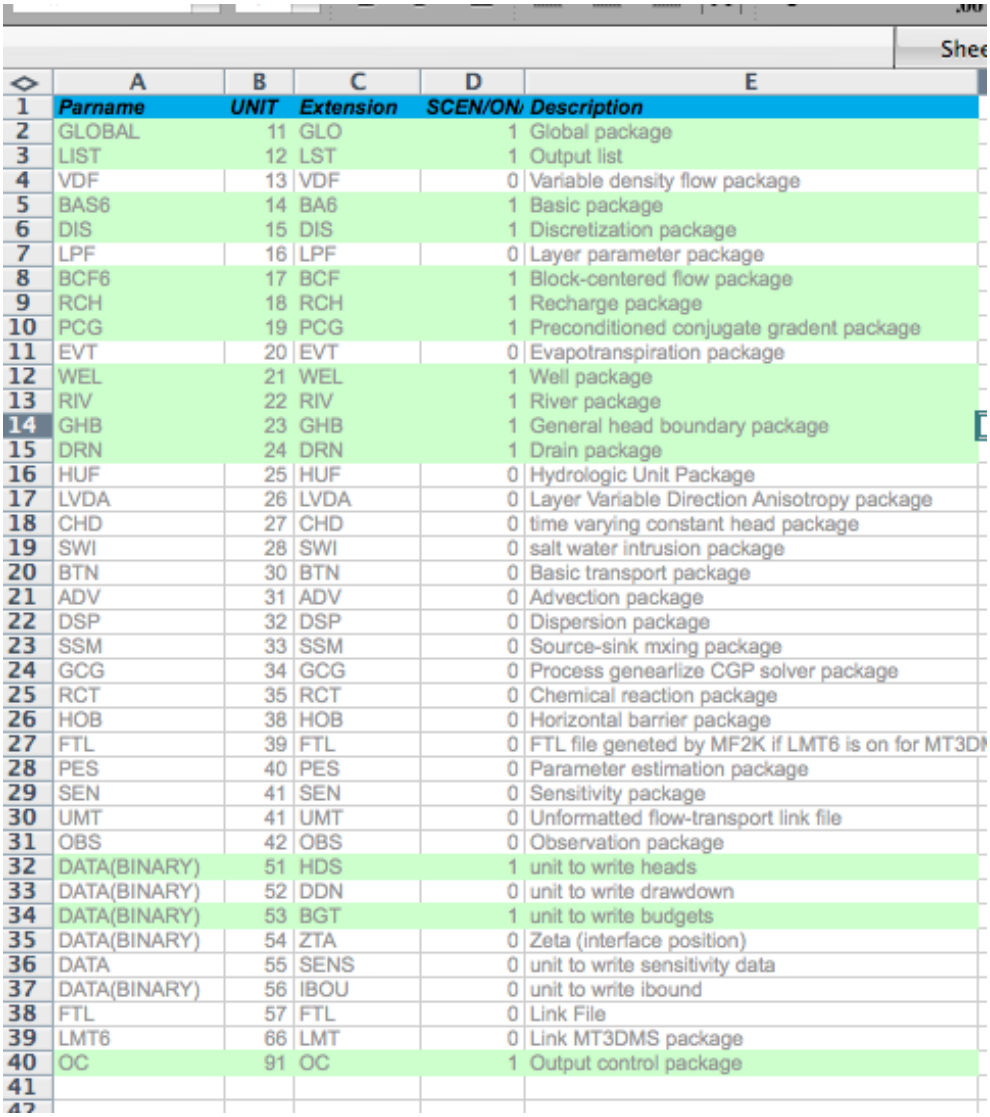
>0	Package is active.
0	Package is off, not used in the simulation
<0	(Only for VDF package, that is SEAWAT's Variable Density Flow package, means that the name file for SEAWAT is generated, but that the VDF package itself will not be used. This allows running SEAWAT as if it were MT3DMS, i.e. with density 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 5.2.

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.



	A	B	C	D	E
1	Parname	UNIT	Extension	SCEN/ON/	Description
2	GLOBAL	11	GLO	1	Global package
3	LIST	12	LST	1	Output list
4	VDF	13	VDF	0	Variable density flow package
5	BAS6	14	BA6	1	Basic package
6	DIS	15	DIS	1	Discretization package
7	LPF	16	LPF	0	Layer parameter package
8	BCF6	17	BCF	1	Block-centered flow package
9	RCH	18	RCH	1	Recharge package
10	PCG	19	PCG	1	Preconditioned conjugate gradent package
11	EVT	20	EVT	0	Evapotranspiration package
12	WEL	21	WEL	1	Well package
13	RIV	22	RIV	1	River package
14	GHB	23	GHB	1	General head boundary package
15	DRN	24	DRN	1	Drain package
16	HUF	25	HUF	0	Hydrologic Unit Package
17	LVDA	26	LVDA	0	Layer Variable Direction Anisotropy package
18	CHD	27	CHD	0	time varying constant head package
19	SWI	28	SWI	0	salt water intrusion package
20	BTN	30	BTN	0	Basic transport package
21	ADV	31	ADV	0	Advection package
22	DSP	32	DSP	0	Dispersion package
23	SSM	33	SSM	0	Source-sink mxing package
24	GCG	34	GCG	0	Process genearlize CGP solver package
25	RCT	35	RCT	0	Chemical reaction package
26	HOB	38	HOB	0	Horizontal barrier package
27	FTL	39	FTL	0	FTL file geneted by MF2K if LMT6 is on for MT3D
28	PES	40	PES	0	Parameter estimation package
29	SEN	41	SEN	0	Sensitivity package
30	UMT	41	UMT	0	Unformatted flow-transport link file
31	OBS	42	OBS	0	Observation package
32	DATA(BINARY)	51	HDS	1	unit to write heads
33	DATA(BINARY)	52	DDN	0	unit to write drawdown
34	DATA(BINARY)	53	BGT	1	unit to write budgets
35	DATA(BINARY)	54	ZTA	0	Zeta (interface position)
36	DATA	55	SENS	0	unit to write sensitivity data
37	DATA(BINARY)	56	IBOU	0	unit to write ibound
38	FTL	57	FTL	0	Link File
39	LMT6	66	LMT	0	Link MT3DMS package
40	OC	91	OC	1	Output control package
41					

Figure 5.2: NAM worksheet

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).

5.2.1 MFLOW worksheet

This MFLOW worksheet in the Excel workbook contains all MODFLOW parameters that are constant during an entire simulation (see figure 5.3). 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.

	A	B	C	D
1	DIS	ITMUNI	4	Time units for simulaton, 4=days
2	DIS	LENUNI	2	Length units for simulation, 2=meters
3				
4	BAS	HNOFLO	1.00E+30	Value of heads to indicate no flow cells
5				
6	BCF/LPF	HDRY	1.00E+20	Head indicating dry cells
7	BCF	IWDFLG	1	Wetting capability active if >0
8	BCF/LPF	WETFCT	1	Wetting factor in equation 33A, see Modflow 2000 manual.
9	BCF/LPF	IWETIT	10	Wetting outer iteration interval insolver
10	BCF/LPF	IHDWET	0	Use wetting equation 33A if zero, otherwise use equation 33B
11	LPF	NPLPF	0	Number of LPF paramters (Currently must be 0)
12				
13	RCH	NRCHOP	3	Recharge flag: 1=in toplayer, 2=in highest active layer, 3=specif
14	EVT	NEVTOP	3	EVT option same as NRCHOP
15				
16	OC	IHEDFM	12	MXITER is the maximum number of outer
17	OC	IDDNFM	12	iterations, that is, calls to the solution routine. For
18	OC	IHEDUN	0	a linear problem MXITER should be 1 unless more
19	OC	IDDNUN	0	than 50 inner iterations are required, in which case
20				MXITER could be up to 10. A larger number,
21	PCG	MXITER	30	generally less than 100, is required for nonlinear
22	PCG	ITER1	50	problems.
23	PCG	NPCOND	1	Flag used to set the matrix conditioning method
24	PCG	HCLOSE	1.00E-03	Head change criterion for convergence [L]
25	PCG	RCLOSE	4.00E-04	Residual criterion for convergence [L3/T]
26	PCG	RELAX	0.99	Relaxation parameter used with NCOND==1
27	PCG	NBPOL	0	Used with NPCOND==2.
28	PCG	IPRPCG	5	Printout interval for PCG. If zero it is set to 999.
29	PCG	MUTPCG	0	Flag that controls printing of convergence information from the s
30	PCG	DAMP	1	Damping factor. 1=no damping. Must be 0<DAMP<=1.
31				
32	SWI	ISTRAT	1	
33	SWI	NPRN	1	
34	SWI	TOESLOPI	0.001	

Figure 5.3: Part of the MFLOW worksheet

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.

5.3 MT3D sheet

The MT3D worksheet contains the parameters that are constant during an entire MT3DMS simulation. Part of it is shown in figure 5.4. The structure is the same as that of the MFLOW worksheet.

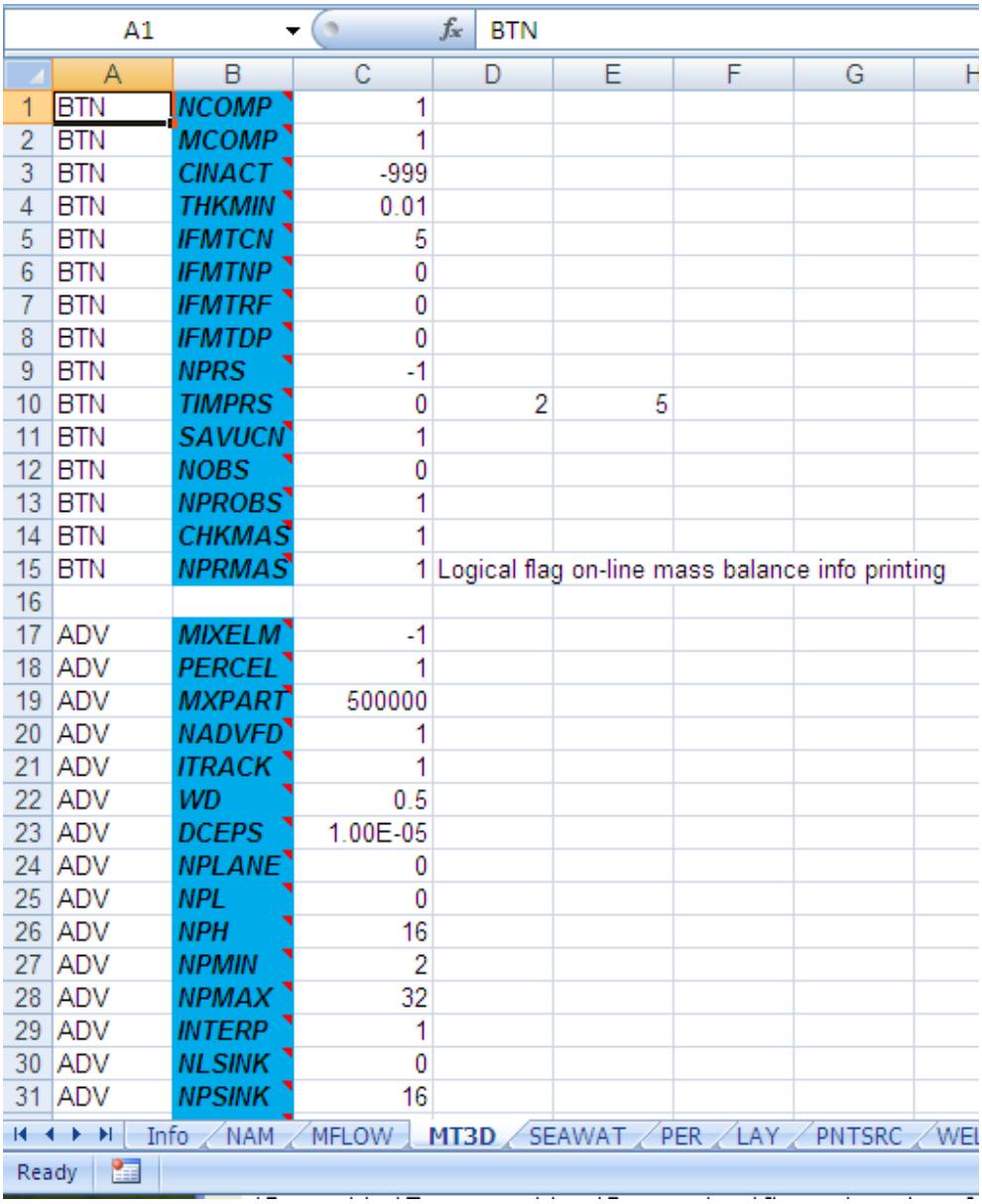
The TIMPRS parameter on the MT3D sheet may contains a series of numbers representing simulation times at which MT3D output is required. This parameter is active if NPRS>0. NPRS is then the number of times to be read for the TIMPRS. *mfLab* will only read as much as TIMPRS time values. A negative value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

5.4 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



	A	B	C	D	E	F	G	H
1	BTN	NCOMP	1					
2	BTN	MCOMP	1					
3	BTN	CINACT	-999					
4	BTN	THKMIN	0.01					
5	BTN	IFMTCN	5					
6	BTN	IFMTNP	0					
7	BTN	IFMTRF	0					
8	BTN	IFMTDP	0					
9	BTN	NPRS	-1					
10	BTN	TIMPRS	0	2	5			
11	BTN	SAVUCN	1					
12	BTN	NOBS	0					
13	BTN	NPROBS	1					
14	BTN	CHKMAS	1					
15	BTN	NPRMAS	1	Logical flag on-line mass balance info printing				
16								
17	ADV	MIXELM	-1					
18	ADV	PERCEL	1					
19	ADV	MXPART	500000					
20	ADV	NADVFD	1					
21	ADV	ITRACK	1					
22	ADV	WD	0.5					
23	ADV	DCEPS	1.00E-05					
24	ADV	NPLANE	0					
25	ADV	NPL	0					
26	ADV	NPH	16					
27	ADV	NPMIN	2					
28	ADV	NPMAX	32					
29	ADV	INTERP	1					
30	ADV	NLSINK	0					
31	ADV	NPSINK	16					

Figure 5.4: MT3D worksheet for one of the MT3DMS examples

for a description. To easily scope with more species, we need more than one numerical value for some of the parameters (see figure below).

5.5 PER (specification of 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) requiring the parameter in question. The parameter name is in the second line. Line 3 and further contain the stress-period parameter values. Each cell with a parameter name has a comment attached that explains it. This comment is usually taken verbatim from the *mf++* manual that defines it.

Each subsequent line defines a (set of) stress period(s).

mfLab allows much flexibility in defining the these stress periods using the following rules:

1. Stress periods are sorted by *mfLab* so their order in the worksheet does not matter.

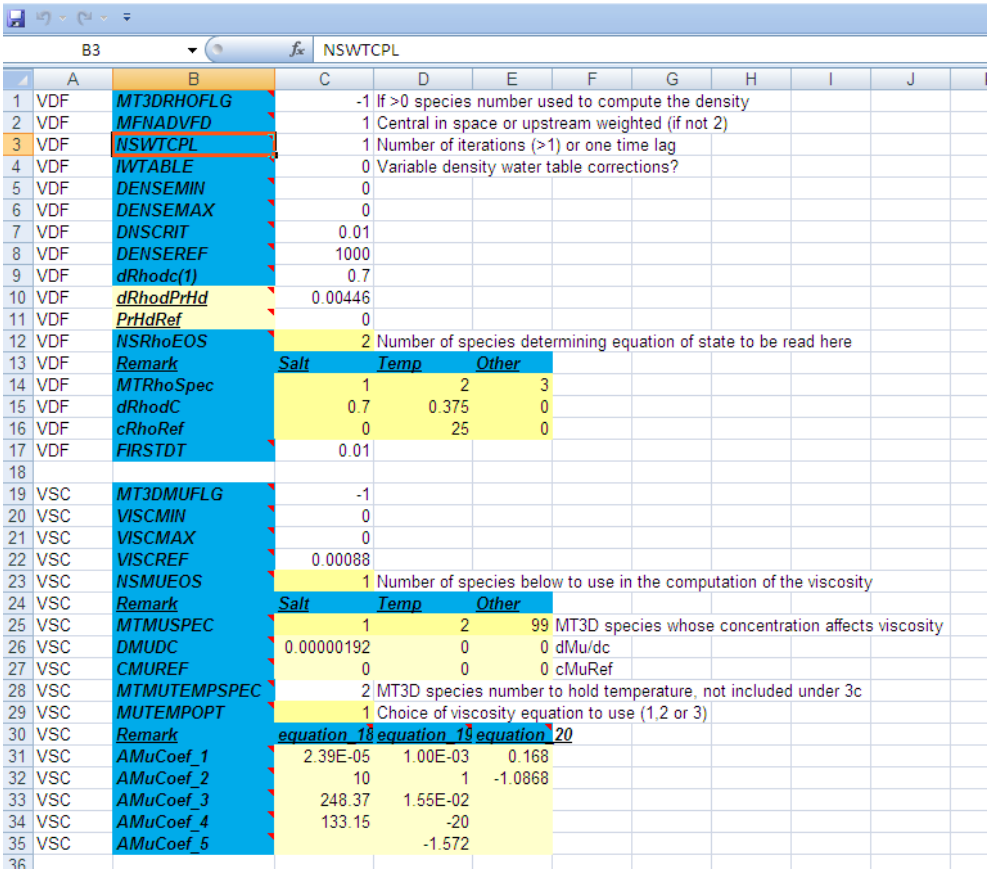
2. The highest specified stress-period number in the worksheet is taken the number of stress periods to be used by the model in the simulation. The stress period number is in the first column of this worksheet.

3. Lines with stress period numbers <1 are ignored. This allows to switch off stress periods easily.

4. Missing stress periods are filled in by *mfLab* prior to generating the input files for the *mf++* programs. This is done backward, implying that the next specified stress period is used to fill in its predecessors.

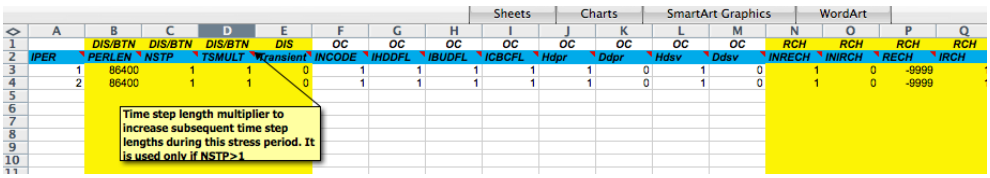
5. If stress periods are defined more than once (i.e. several lines have the same stress period number), then the first of them wins (as a consequence of filling in the missing periods backward).

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



	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 5.5: SEAWAT worksheet in the Excel workbook of the Coast example for SEAWAT



	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	
5																	
6																	
7																	
8																	
9																	
10																	

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

Figure 5.6: part of the PER worksheet, also showing a comment explaining the meaning of the variable TSMULT

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. 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)

5.6 LAY (specification of 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.

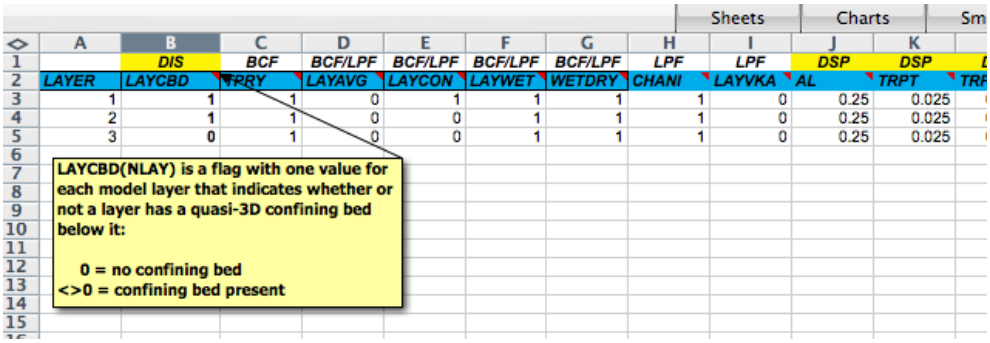


Figure 5.7: Part of the LAY worksheet, also showing the comment attached to the parameter LAYCBD

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).

5.7 Boundary conditions (WEL, GHB, RIV, DRN, CHD)

The workbook provides worksheets for each of the boundary condition options, i.e. WEL, DRN, RIV, GHB and CHD. These sheets all have the same

structure, namely [stressPeriod, Layer, Row, Col, parameter-values]

mfLab will only look for these worksheets if their respective package is "on" in the NAM sheet.

Each of these boundary condition types can also be directly specified in *mf_ adapt* or in both *mf_ adapt* and these worksheets. *mf_ adapt* will merge them. A zero number of boundary conditions are allowed, but a more effective way is to switch off boundary types that are not needed in the simulation. This is done in the NAM sheet.

	A	B	C	D	E
1	PERIOD	LAYER	ROW	COLUMN	Q
2	1	3	5	11	-5
3	1	2	4	6	-5
4	1	2	6	12	-5
5	1	1	9	8	-5
6	1	1	9	10	-5
7	1	1	9	12	-5
8	1	1	9	14	-5
9	1	1	11	8	-5
10	1	1	11	10	-5
11	1	1	11	12	-5
12	1	1	11	14	-5
13	1	1	13	8	-5
14	1	1	13	10	-5
15	1	1	13	12	-5
16	1	1	13	14	-5
17					

Figure 5.8: WEL worksheet of example ex1

	A	B	C	D	E	F
1	PERIOD	LAYER	ROW	COLUMN	ELEVATION	COND
2	1	1	8	2	0	1
3	1	1	8	3	0	1
4	1	1	8	4	10	1
5	1	1	8	5	20	1
6	1	1	8	6	30	1
7	1	1	8	7	50	1
8	1	1	8	8	70	1
9	1	1	8	9	90	1
10	1	1	8	10	100	1
11						
12						

Figure 5.9: DRN worksheet of example ex1.

To specify boundary conditions in *mf_ adapt*, a list array has to be defined in the Matlab workspace with the required structure. This structure is the same as described in the MODFLOW manual, except that *mfLab* requires that each line is preceded with the stress period number. This makes each line unique and guarantees unambiguous interpretation of the input lines.

The structure necessary for each boundary condition list can be readily seen from the labels in the respective worksheets.

The following rules apply:

1. If no boundary conditions are specified for a given stress period, then these boundaries are off in this period.

2. If a negative stress period number is used, then the boundaries of the previous stress period are reused and layer, row and column numbers of such lines are ignored (so any values can be filled in, I suggest to use zeros.

3. If both negative and positive stress period numbers are specified for the same period, then only the negative one will be used by *mfLab*.

Up to 5 auxiliary parameters may be specified, according to the official manual. Such parameters if specified will also be read by *mf_ setup*.

Because the number of boundary condition cells can be enormous, the spreadsheet may not always be the most convenient way to store these conditions.

The specify these boundary conditions directly in mf_adapt use the structure shown in the worksheet. The names of the list/array variables mf_setup expects to find for these parameters in the workspace are WEL, DRN, RIV, GHB, RIV respectively. mf_setup will simply merge the boundary conditions found in the various worksheets with those it encounters in the workspace at the time of *mf++* input file generation.

This merging may be convenient when most of a large set of boundaries are constant between scenarios and some are not. In such cases one may for instance put the cells to be changed the spreadsheet and leave the large list to be handled by *mf_ adapt* intact.

According to the SEAWAT version 4 manual ([9], p12-14) two auxiliary parameters can be specified with the CHD package. CHDDENSOPT and CHDDENS. These allow specification of the prescribed head boundary in terms of saltwater density. Please refer to that manual for details. CHDDENSOPT and CHDDENS are active automatically when they exist in the workspace of Matlab. See the Elder example for how to make effective use of them in *mfLab*.

5.8 PNTSRC (SSM package)

The SSM module of MT3DMS (and therefore also SEAWAT) requires specification of all sources and sinks, at least the sinks. As described in the previous section. The point sources, PNTSRC, have the following list structure.

[stressPeriod, Layer, Row, Column, CSS, ITYPE, CSSMS(1..NCOMP)]

CSSMS(1..NCOMP) may be omitted if only a single species is involved in the model. In that case, CSS is used as the concentration of the involved species. If more than one species is modeled, then CSSMS(1..NCOMP) needs to be included, where NCOMP the number of species. In these cases CSS is ignored but still needs to be specified according to MT3DMS.

The worksheet PNTSRC is almost identical to that of the boundary, conditions WEL, DRN, RIV, GHB and CHD. Here too, each line starts with the stress period number followed by the Layer, Row and Column indices and finally the parameters values. ITYPE indicates the type of point source as specified in the MT3DMS manual for the SSM package. In general, each flow-model boundary condition through which non-zero concentration may enter the model requires a counterpart in the PNTSRC list of the SSM package, except drains, because drains can only discharge. See the documentation of SSM in the MT3DMS manual.

Like it is the case with the boundary conditions WEL, DRN, RIV, GHB and CHD package of the flow model, PNTSRC entries may be defined in the both in the worksheet and in mf_adapt.m directly; the data will be merged before *mfLab* generates the input file for the SSM package.

MT3DMS (and SEAWAT for that matter) can model simultaneous transport of up to 100 species each with its own properties and retardation. To view an example of MT3DMS/SEAWAT use with more than one species see "Coast", which uses temperature as a second species next to density.

5.9 HUF worksheet (Hydrogeologic Unit Flow)

The HUF package is an alternative to BCF and LPF, it defines the subsurface in terms of hydrological units that are independent of the layers of the model. It, therefore, needs a layer definition separate from that of the computation grid. *mfLab*'s HUF implementation is still under development and not well tested. HUF has to be defined totally in terms of parameters. The parameters needed in the HUF sheet are the parameter name for each hydrological unit and the horizontal and vertical anisotropy value pertaining to each hydrological unit. The top and bottom of each hydrological unit must be defined by *mf_ adapt*. And in the HUF package must be switched

"on" in the NAM worksheet, while both the BCF and LPF packages must be switched "off".

5.10 BTNOBS worksheet

The BTNOBS allows the specification of concentration observation points. It only requires the layer, row and column numbers of cells for which a concentration over time output is desired during a MT3DMS scenario. The course of concentration at these observation cells can then be analyzed at a later time. There is a function *readObs* to read the produced observation data into Matlab for visualization.

Chapter 6

Mfiles (Matlab code)

6.1 Generalities

The mfiles are Matlab functions and sometimes Matlab scripts, which, together, comprise *mfLab*. The mfiles have been arranged in directories in a more or less logical way to prevent clutter as much as possible. The different mfile directories contain functions that may be useful for model generation and interpretation. Some help with the handling of coordinates and computation grids. Other functions may be more generally applicable.

The mfiles/write directory contains the mfiles *mf_setup.m* and all mfiles that generate the input files for the *mf++* target models. Each time a new package for one of these models is to be implement in *mfLab*, a new write function must be made together with a new accompanying block in *mf_setup*.m which gathers and assembles the parameters required by the write function.

The best way is to start writing a function for a new package it to start with one that is similar to the new one to be implemented. The result will then be obtained faster and will require less debugging. Keeping functions similar and elegant causes fewer headaches during development and, especially, during debugging at some point in the future.

The philosophy of reusing code and 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). Other write functions for other packages like recharge an evapotranspiration have been kept as similar as possible as is the case with the function that write the input files for the BCF and LPF packages for MODFLOW.

One output file that does almost all the work, the array writing function *warray.m*. It is named after the rarray (array reading function) used by

MT3DMS to read real arrays. It is used to write both the arrays tot to the int put files of the MODFLOW and the MT3DMS packages, regardless whether the arrays contain integers or floating point numbers of whether the arrays is written as a list of full 2D layers, regardless whether the array should be written with a or without an array header; *warray* does it all.

This guarantees that all functions that need to write out arrays will use the same routine, which makes *mfLab* robust.

All writing by *mfLab* is done in fixed format. This may seem counter intuitive (even to myself at the beginning), 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. This is necessary to find bugs with most certainty. A disadvantage with using fixed formats is that the MODFLOW developers simply allowed too little space for many variables. While floating point numbers requires 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.

Wat 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.

We will discuss the mfiles directory by directory.

6.2 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.

6.3 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 4.4.

H=readDat(fname[,periods[,tsteps[,lays[,rows[,cols]]]]]) read heads, draw-down or SWI zeta file into convenient struct

B=readBud(fname[,labels[,periods[,tsteps[,lays[,rows[,cols]]]]]]) read 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.

6.4 mfiles/write

This directory con stains all routines that generate input files for target models. It also contains mf_setup.m, the backbone of *mfLab*.

6.5 mfiles/etc

Various useful functions.

6.6 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, axial symmetric, 3D, steady and transient as well as density flow and finally particle tracking. See the manual for instructions.

[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,zGr,xm,ym,zm,Dx,Dy,Dz,Nx,Ny,Nz]=modelsize3(xGr,yGr,zGr) same but includes the third dimension

Below follow most of the finite difference models that the directory contains. These are small jewels. Use help file 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-rhof)/rhof.

[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

Chapter 7

Examples

7.1 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_setup in the command window of Matlab. After *mf_setup* has finished type *mf_analyze*.

Of course you could immediately type

mf_setup; mf_analyze

in the command window to have *mf_analyze* follow *mf_setup* automatically.

However, this is useful only if you are sure that the models that are

launched at the end of mf_setup (MODFLOW, MT3DMS, SEAWAT etc) will terminate normally. If not, mf_analyze will fail as it misses all or some of the model output.

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).

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 screen)

mfLab/mfiles/write
mfLab/mfiles/read
mfLab/mfiles/etc
mfLab/mfiles/gridcoords

Make sure that the MODFLOW directory specified in mfiles/write/mf_setup.m points to the correct directory on your system.

The same must be true for MT3D, SWI and SEAWAT and possibly other directories.

In Matlab cd (change dir) to the directory of the example you want to run, say

mfLab/examples/mf2k/ex1

and type in the command window

mf_setup

followed by (if all goes well)

mf_analyze

Or type:

mf_setup; mf_analyze

on one line to run both file in sequence without a pit stop in the middle.

7.2 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.

7.2.1 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 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.

7.3 Calibration with PEST

7.3.1 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.

7.3.1.1 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_ setup.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.

7.3.1.2 How does the caibration 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_ setup.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_ setup*. The script *mf_ setup.m* continuous and constructs all the input files for the models to be run as usual. *mf_ setup* 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.

7.3.1.3 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_setup 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_setup.

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.

7.4 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 7.1, 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

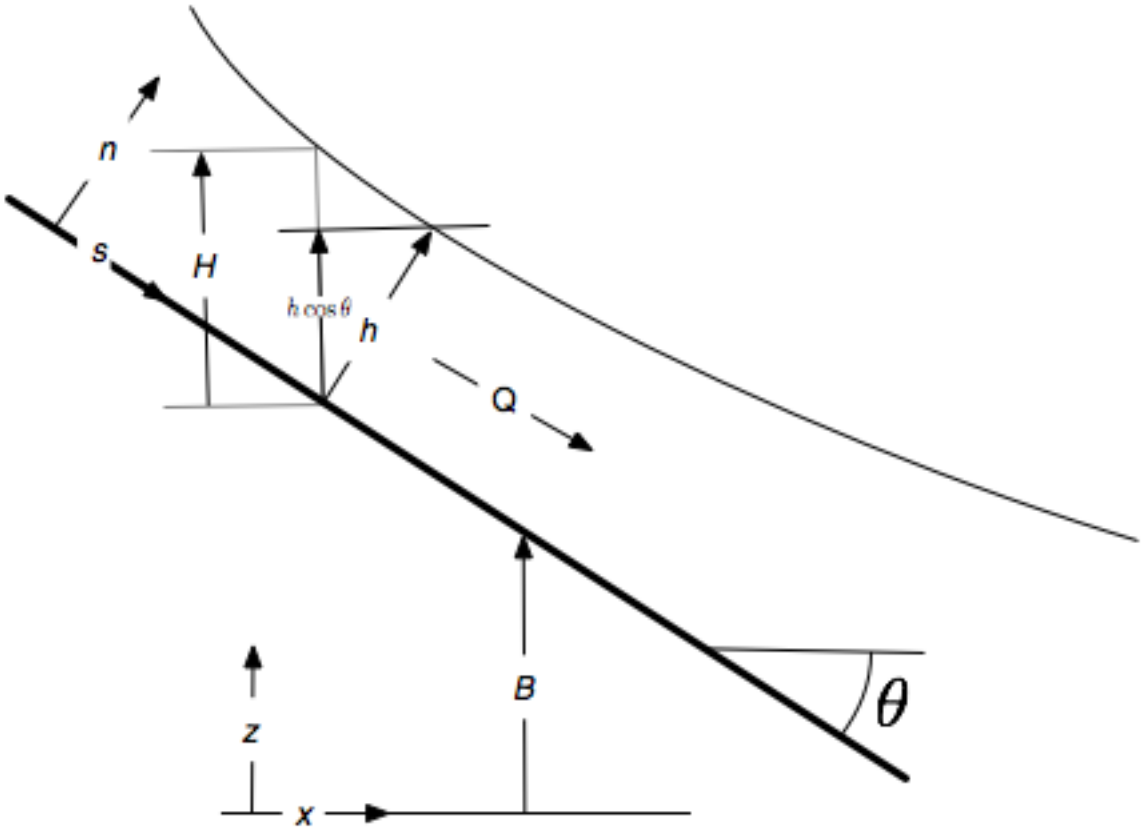


Figure 7.1: Groundwater flow on a slopt Boussinesq (1879)

beomes

$$Q = -kh\frac{\partial\left(h\cos\theta + B\right)}{\partial s} \tag{7.1}$$

$$Q = -kh\left(\cos\theta\frac{\partial h}{\partial s} + \sin\theta\right) \tag{7.2}$$

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

$$Q = -k^* H\left(\frac{\partial H}{\partial x} + \tan\theta\right) \tag{7.3}$$

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

$$k^* = k\cos^2\theta \tag{7.4}$$

This is clear from

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

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,

$$h_0 = \frac{Q}{k\sin\theta}, \quad H_0 = \frac{Q}{k^*\tan\theta} \tag{7.5}$$

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 (7.4). 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 (7.4).

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

$$Q = -kh\frac{\partial \phi}{\partial s}, \quad Q = -k^* H \frac{\partial \phi}{\partial x} \tag{7.6}$$

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 (7.6)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 (7.4). 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 (7.3) for uniform discharge , which can now be used in corrected form as follows (7.4):

$$\ln\frac{\frac{H}{H_0}-1}{\frac{H_1}{H_0}-1} + \frac{H}{H_0} - \frac{H_1}{H_0} = -\frac{B-B_1}{H_0} = -\frac{(x-x_1)\tan\theta}{H_0} \tag{7.7}$$

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

$$\ln\frac{\frac{h}{h_0}-1}{\frac{h}{h_1}-1} + \frac{h}{h_0} - \frac{h_1}{h_0} = -\frac{B-B_1}{h_0\cos\theta} = -\frac{(s-s_1)\sin\theta}{h_0\cos\theta} \tag{7.8}$$

and h_0 and H_0 the normal depths according to (7.5) 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 7.2. 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.

7.4.0.4 Comparion with analytical solution

The analytical solutions (7.8) and (7.7)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

$$x - x_1 = \frac{h_0}{\tan(-\theta)} \left\{ \ln \frac{\xi - 1}{\xi_1 - 1} + \xi - \xi_1 \right\} \tag{7.9}$$

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 $0 \leq \xi \leq 1$ in the second case $\infty \geq \xi \geq 1$. In the frist case $\xi < \xi_1$

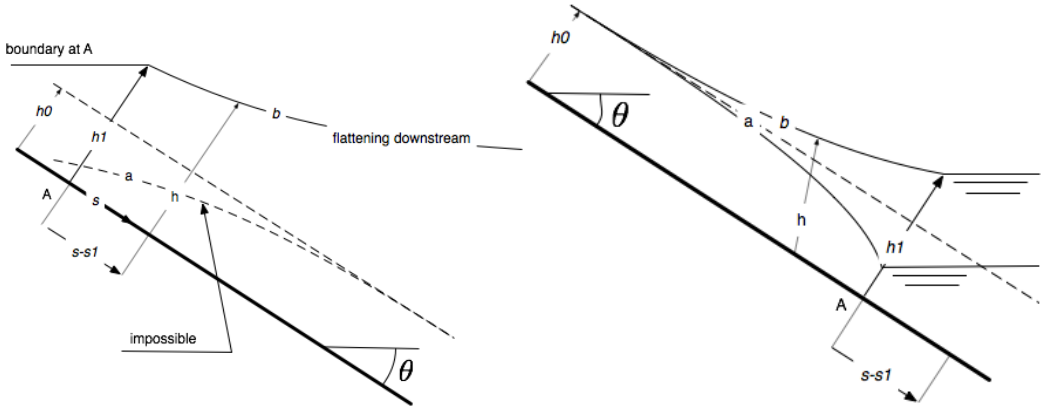


Figure 7.2: Boussinesq solution for uniform discharge. Left: downstream from point A. Right: Upstream of point A.

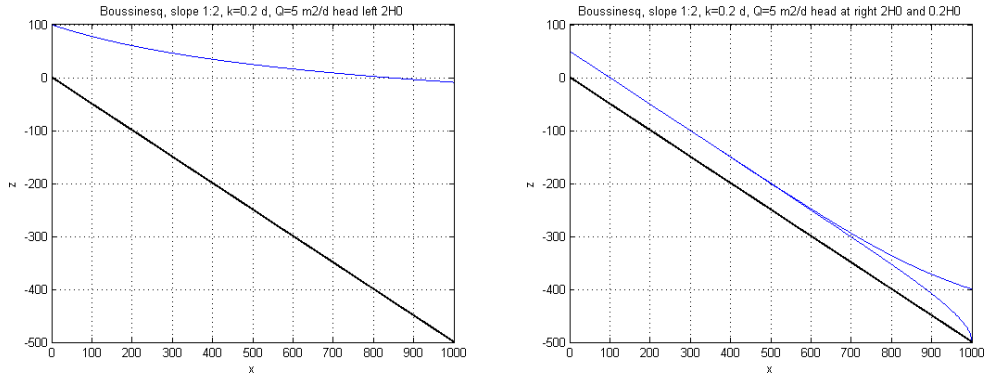


Figure 7.3: Numerical solution using $k = 0.2\,\mathrm{m/d}$, $\tan(slope)$=1/2, $Q = 5\,\mathrm{m^2/d}$, boundary $2 \times H_0$ and $0.2 \times H_0$, 1000 cells of 1 m, Picture left: Situation downstream of point where $h > h_0$. Picture right, head upstream of point where $h > h_0$ and where $h < h_0$.

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 (7.5).

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.

The analytical solution is shown in figure 7.5; the mfile which generated the figure is shown in figure 7.4. For easy comparison with the numerical results in figure 7.3, the same numerical values were used while the solution was superposed on the slope.

In the analytical solution given in figure 7.5we 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

$$\frac{s - s_1}{h_0} \tan(-\theta) = \ln \frac{\xi - 1}{\xi_1 - 1} + \xi - \xi_1 \tag{7.10}$$

which reveals over what distance the boundary has effect on the head (see figure 7.6).

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

```
%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, eta1=H1/H0
eta1A=2.0; etaA = logspace(0,log10(4),50);
eta1B=0.5; etaB = linspace(0.001,0.999,100);
sA=H0/(-slope)*(log((etaA-1)/(eta1A-1))+etaA-eta1A);
sB=H0/(-slope)*(log((etaB-1)/(eta1B-1))+etaB-eta1B);

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 7.4: Computation of analytical solution in Matlab

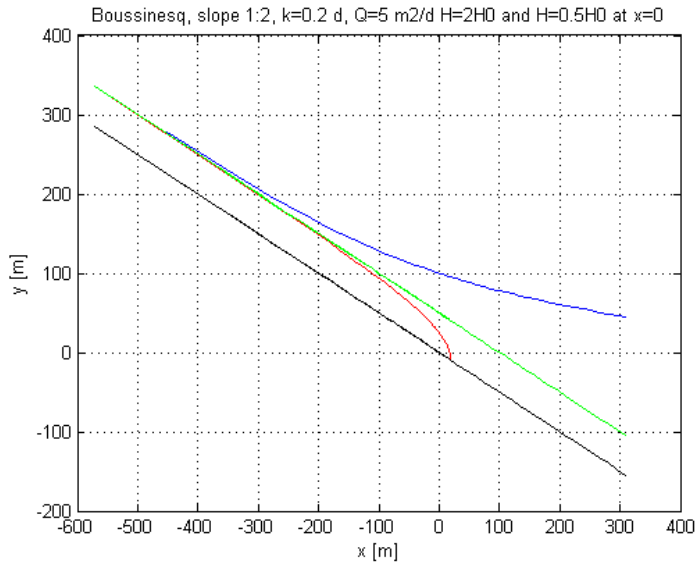


Figure 7.5: Analytical solution of Steward (2007), i.e. equation (7.8).

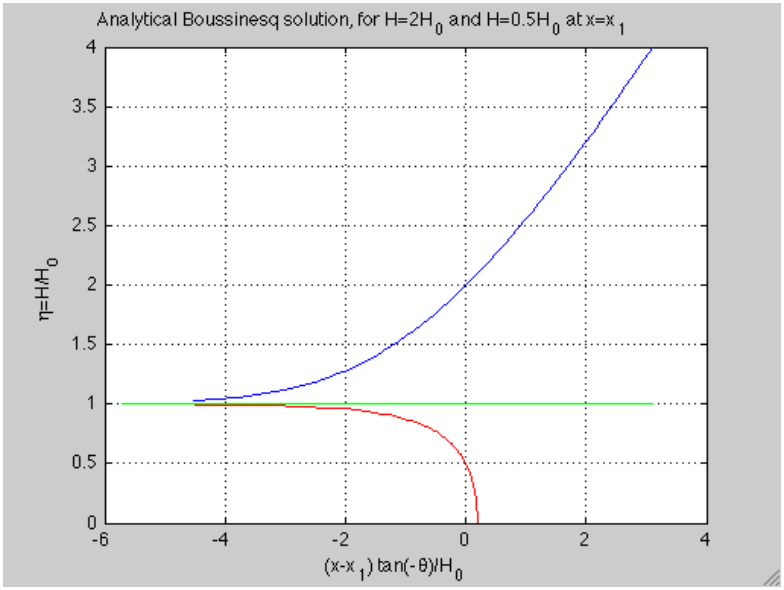


Figure 7.6: Analytical Boussinesq solution in in dimensionless form

head and aquifer top no longer change.

7.4.0.5 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_setup 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 7.7. The blue lines produced by fdm3 are completely covered by the red dots which are the plot of the results from mfd2k.

7.4.0.6 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.

7.5 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

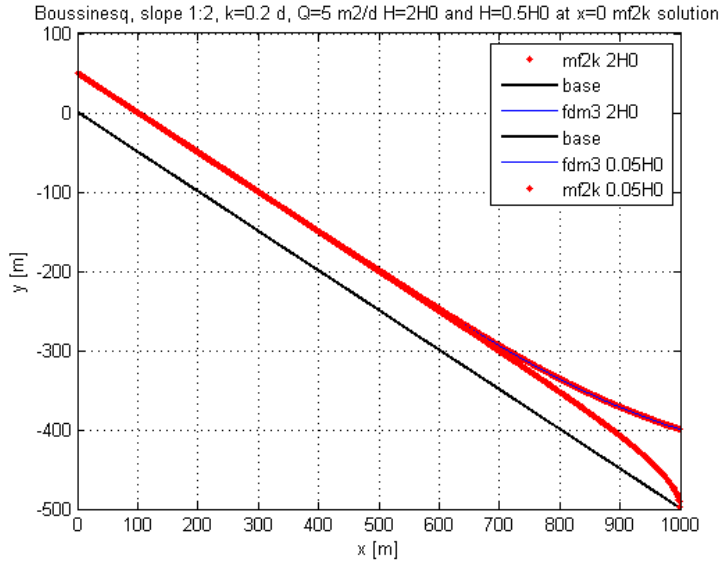


Figure 7.7: Head computed with mf2k through *mfLab* (red) and fdm3 (blue) for uniform discharge of $5\,\mathrm{m}^2/\mathrm{d}$ and head boundary condition at right-hand side of $2H_0$ and $0.05H_0$ respectively, with H0 the normal depth and $k = 0.2\,\mathrm{m/d}$.

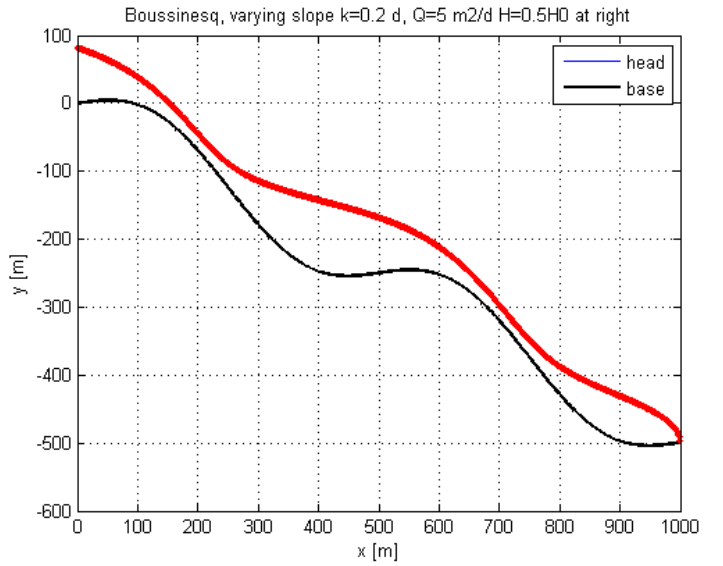


Figure 7.8: Contant groundwater discharge on a sloping base

thus prevent convergence and possible cause spurious uncertain en somtimes even wroing results.

7.5.1 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

7.5.2 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 $0 < \beta \leq 1$.

7.5.3 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 $d_0 = 2.5\,\mathrm{m}$, $d_2 = 5d_0$ 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 7.9 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

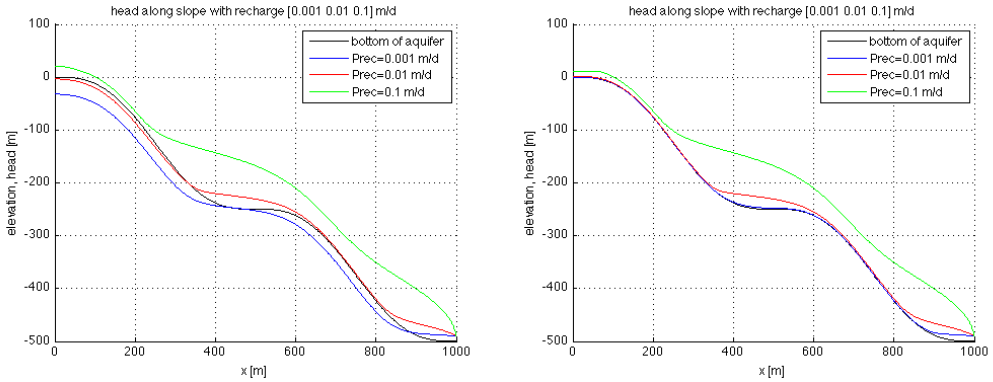


Figure 7.9: Water table above a sloping aquifer base with recharge (see text)

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 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.

Chapter 8

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.

8.1 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 8.1 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.

8.2 1D-Nonlinear

This example concerns one-dimensional transport with non-linear sorption. Both Langmuir and Freundlich sorption has been computed with *mfLab*. Figure 8.2 shows the development of the concentration curves over time with Langmuir sorption. Figure 8.3is 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.

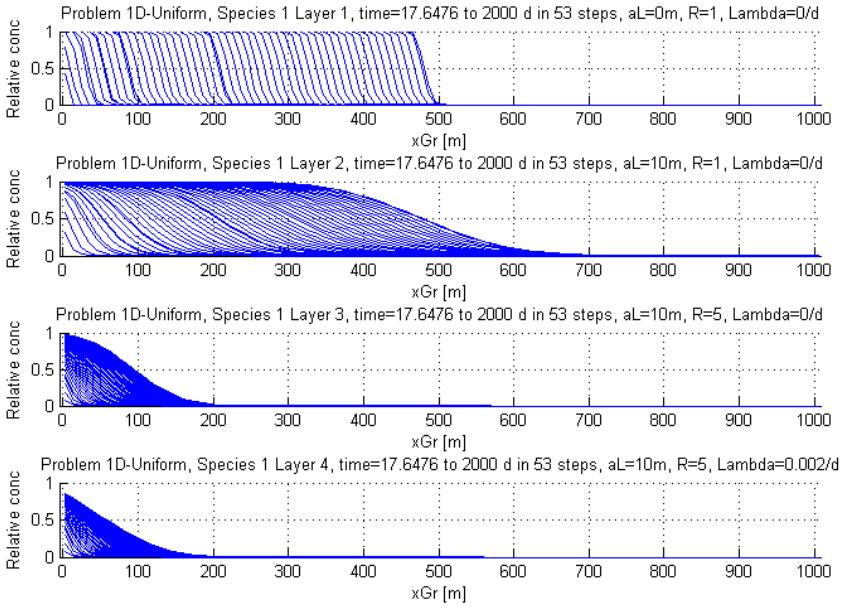


Figure 8.1: 1D-Linear: The four cases as described above. See *mf_adapt* in the example directory for numerical values

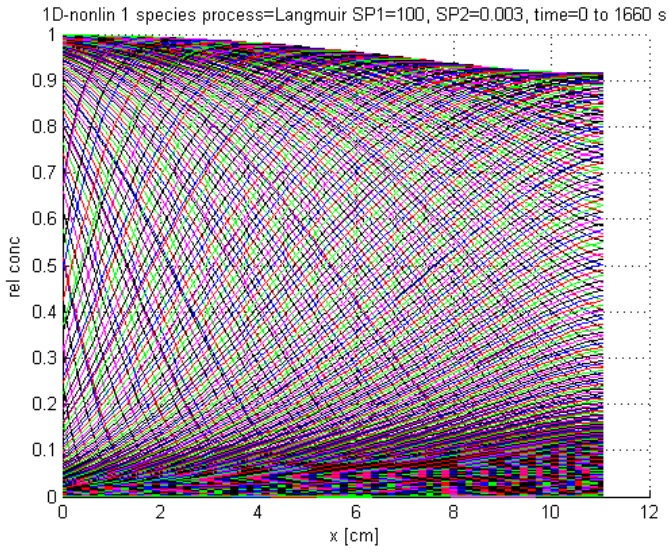


Figure 8.2: 1D nonlinear: Concentration curves at many times during the transport

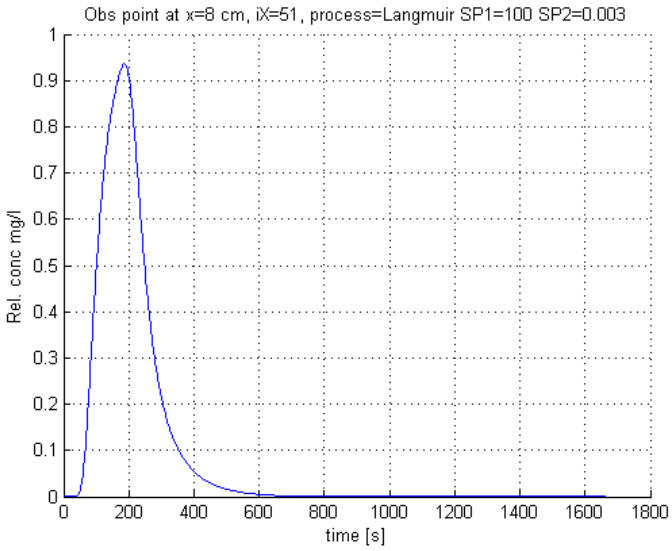


Figure 8.3: 1D-nonlinear: Breakthrough of the concentration at a fixed point (x=8 cm) for Langmuir nonlinear sorption

8.3 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 8.4 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 8.4 shows the spread of the contaminant after one year and 8.5 shows the concentration over time at various points of the grid.

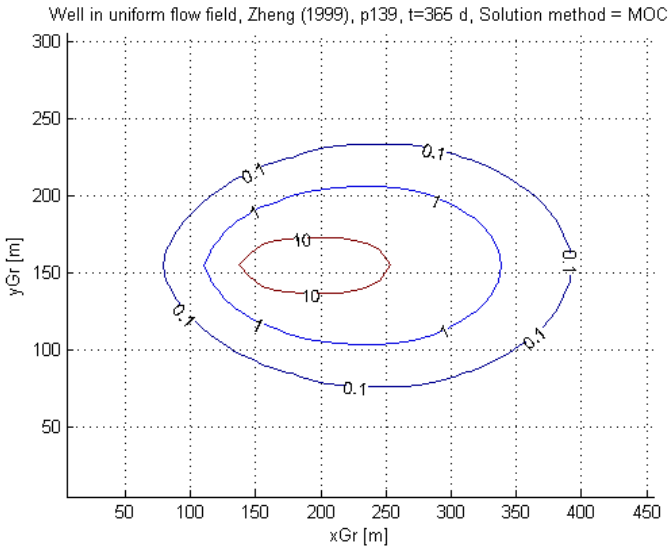


Figure 8.4: Concentration contours after 365 days of injection

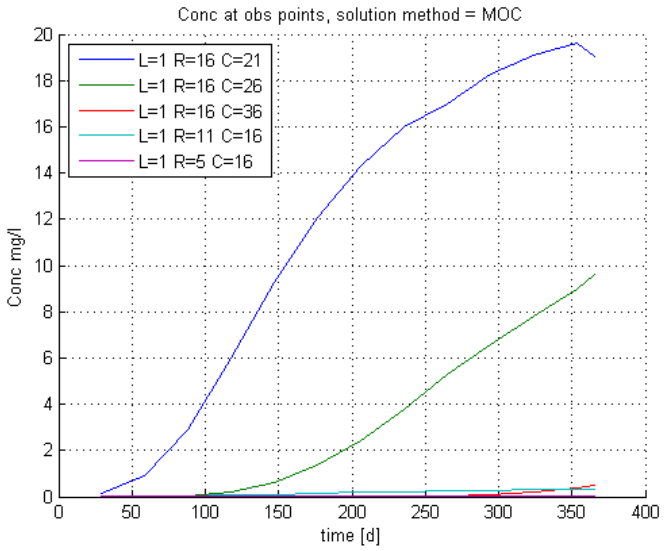


Figure 8.5: 2D-Uniform: Evolution of the concentration at different locations in the grid

8.4 2D-Diagonal

The example concerns two-dimensional flow in a diagonal flow field (see figure 8.6). 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.

8.5 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 flow system. Figure 8.7 and 8.8. 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.

8.6 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 8.9. Salt was entered in one of the upstream stand pipes and the electrical conductivity was measured continuously at downstream standpipes.

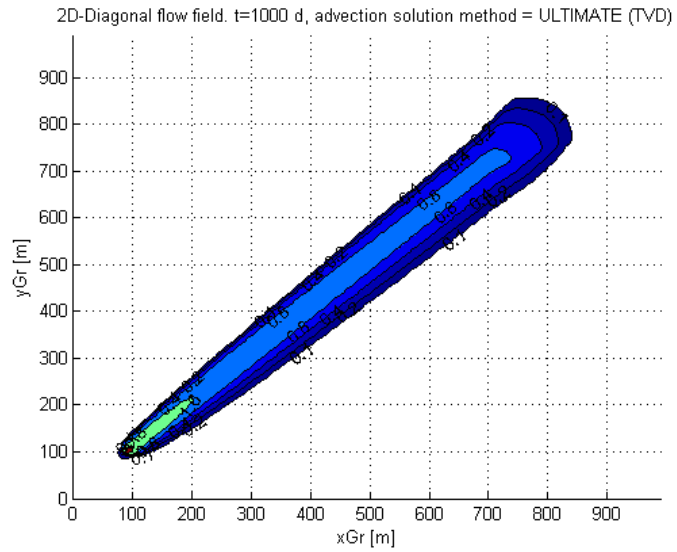


Figure 8.6: Diagonal flow field: Relative distribution of constituent after 1000 days injection in a uniform diagonal flow field

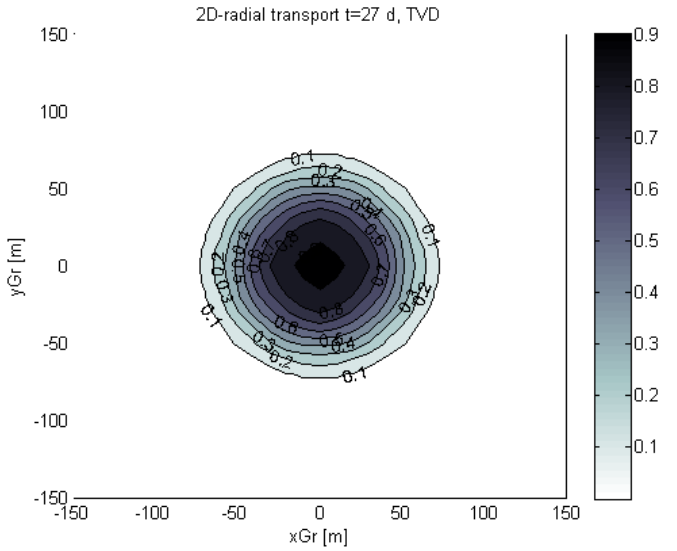


Figure 8.7: Injection with radial flow: distribution of constituent after 27 days of injection

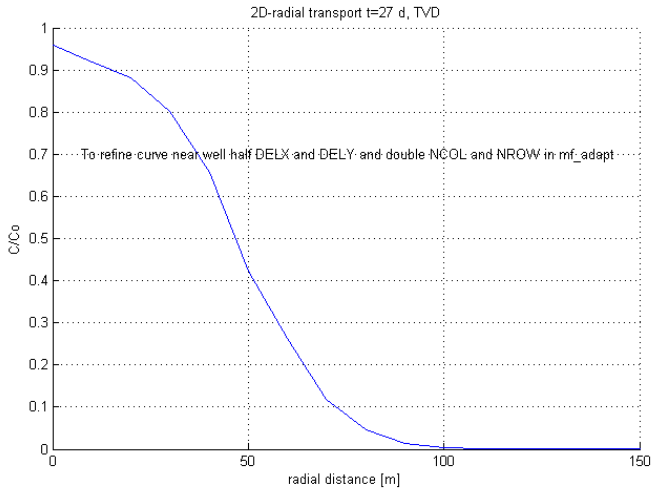


Figure 8.8: Injection with Radial flow: distribution of constituent after 27 days injection

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 8.12discharge 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 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

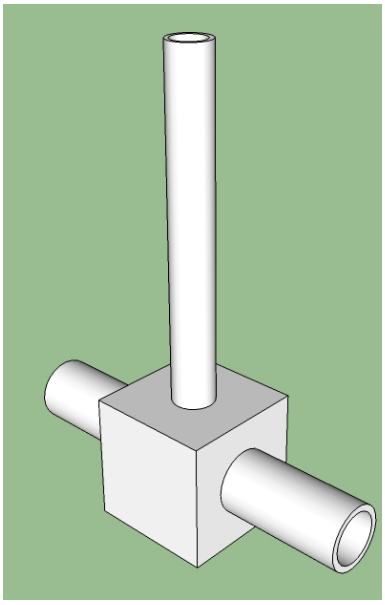


Figure 8.9: Drawing of gallery with connection box and standpipe for observations

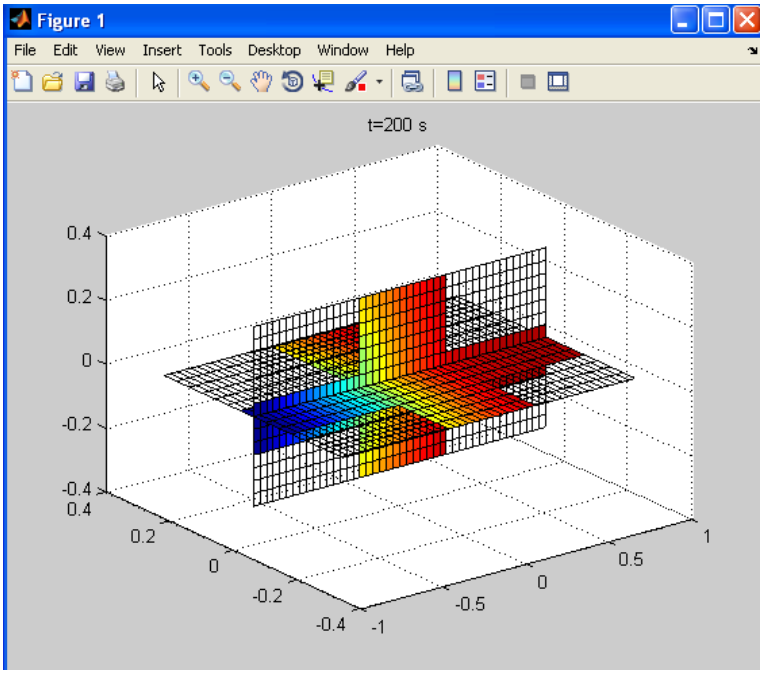


Figure 8.10: Salt dilution test: Salt distribution in the model after 200 seconds

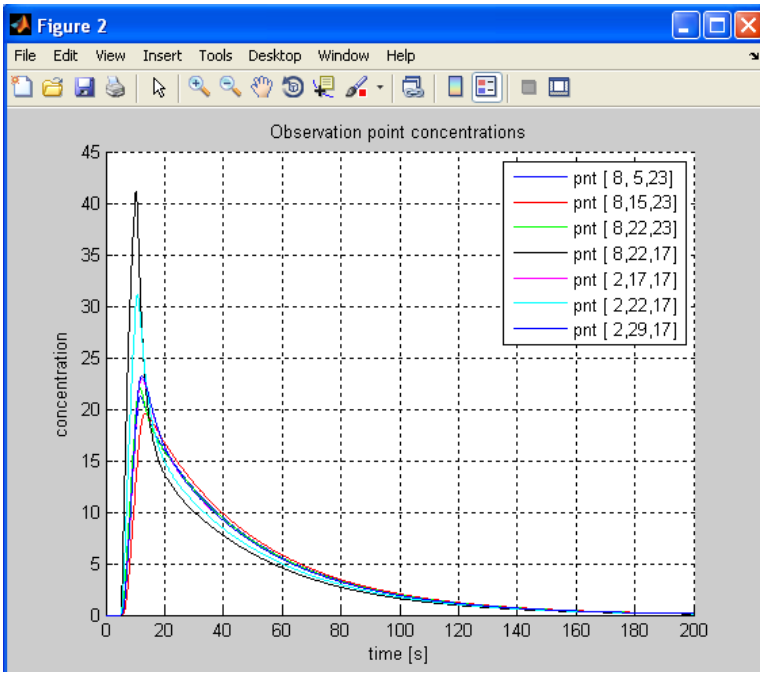


Figure 8.11: Salt dilution test, break-through curves for specific observation locations in the network

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.

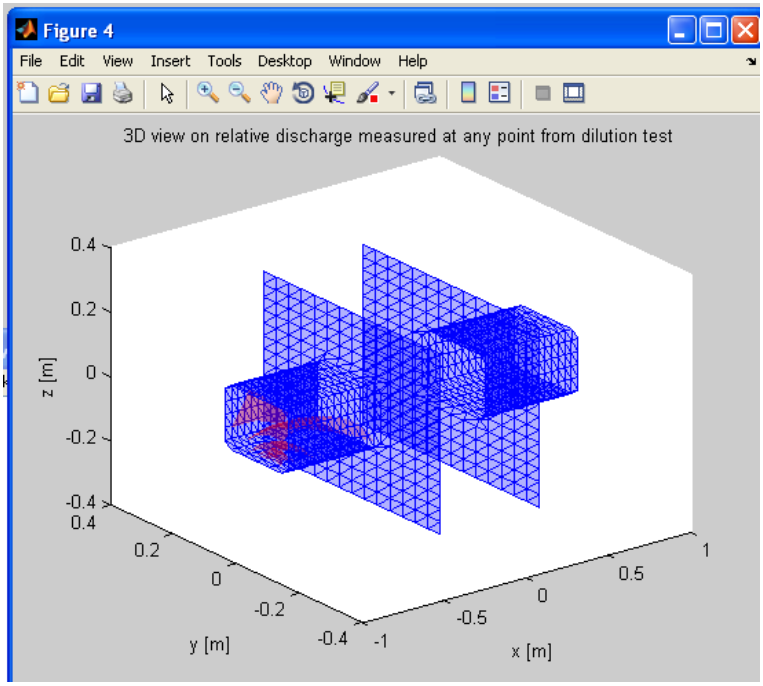


Figure 8.12: Salt dilution test, isoplanes showing in which part of the model the test would reveal a total flow less than the real total flow

Chapter 9

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.

9.1 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 9.1 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 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 9.1. 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

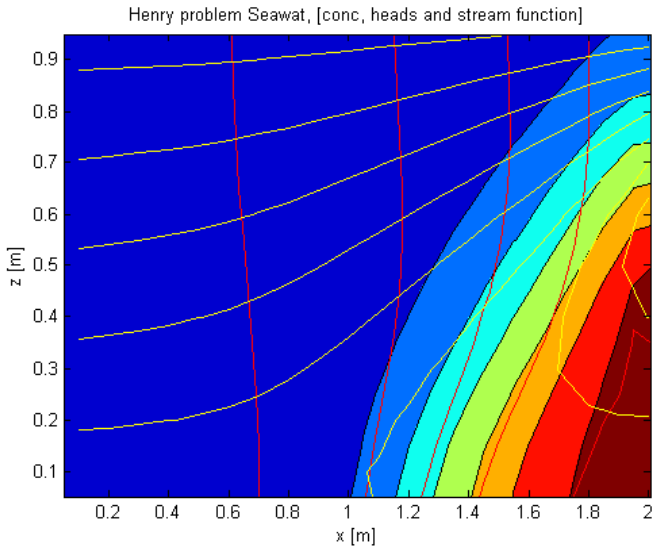


Figure 9.1: The Henry problem, freshwater heads (red), streamlines (yellow) and density as filled contours

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.

9.2 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 9.2. The problem is extensively documented in the *mf_adapt.m* file on the example directory.

9.3 Hydrocoin

The hydrocoin problem has been somewhat difficult as the GCG solver would not converge with the Cholesky decomposition method. However, it did

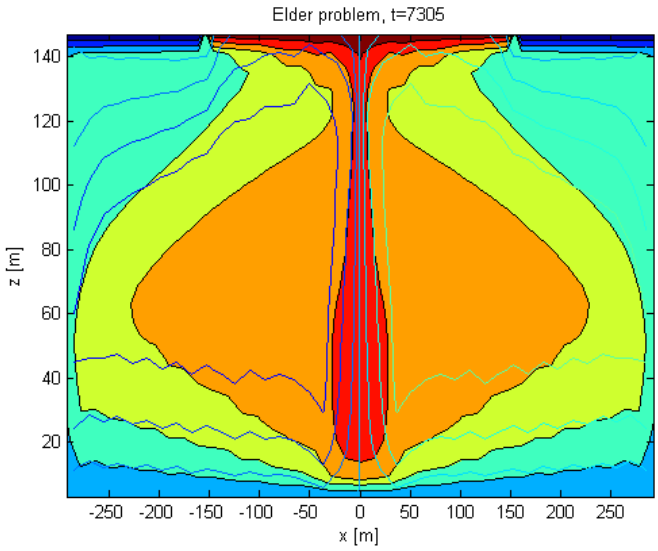


Figure 9.2: Result of Elder problem

converge with SSOR (see parameter PERCEL in the MT3D worksheet). It finally worked with good results that are shown in figure 9.3. 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.

9.4 Coastal flow

Figure 9.4shows 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

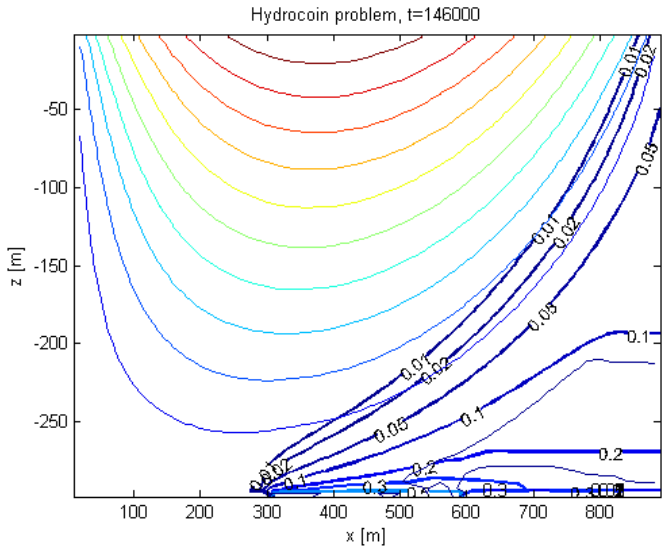


Figure 9.3: Results of hydrocoin example t=146000 d

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 vis-

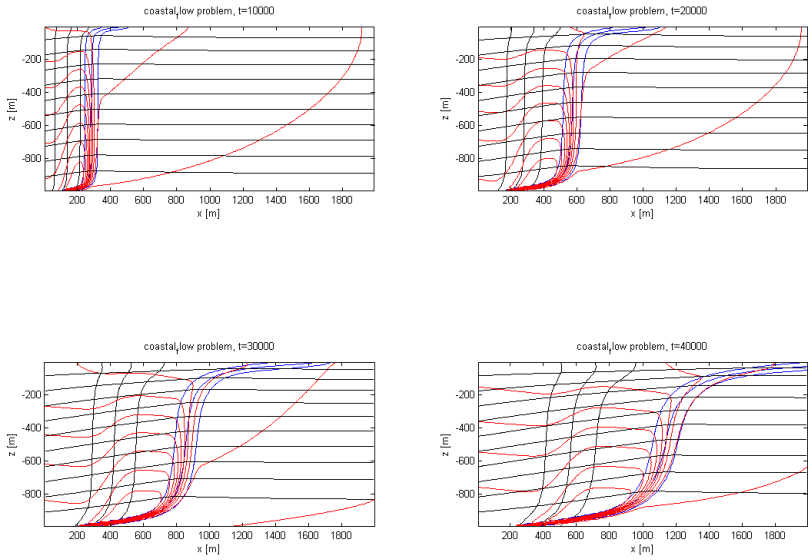


Figure 9.4: Coastal_flow problem presented by Langevin et al, 2008

cosity 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.

Chapter 10

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.

10.1 SWI example 1, rotating interface

The first example is a rotating interface as shown in figure 10.1.

The interface positions computed with the *mfLab* implementation is given in the second picture of 10.1, 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.

10.2 SWI example 2, rotating brackish zone

Figure 10.2from 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.

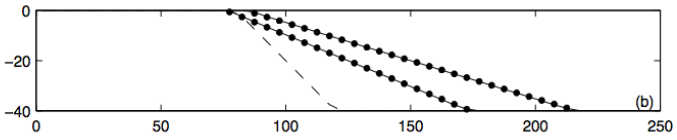
The first figure is from the SWI manual showing setup (a), comparison with SEAWAT (b) and comparison of the results of the stratified versus the non-stratified option of SWI. The second figure shows the results as computed via mfLab for the stratified option

The second figure of figure 10.2shows 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.

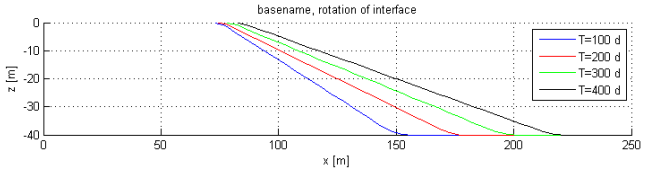
10.3 SWI example 3

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 10.3taken 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 10.3 shows the results computed through *mfLab*. See *mf_analyze* in the example directory for the methods used to visualize the results.

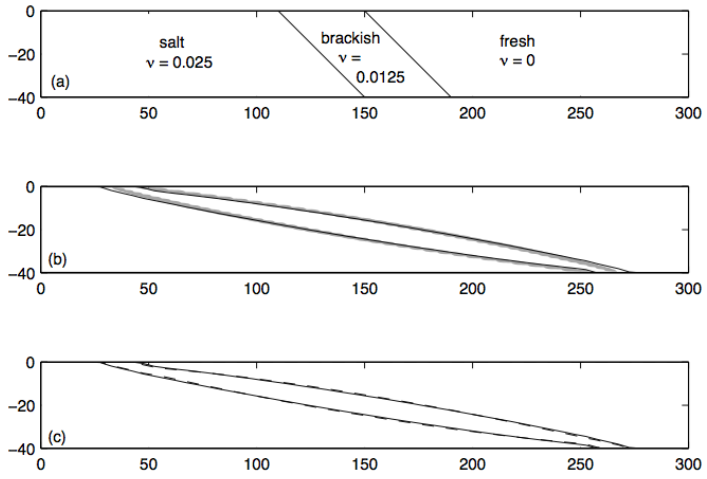


(a) from SWI manual

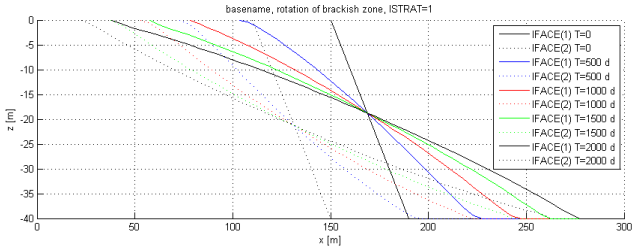


(b) *mfLab* results

Figure 10.1: Initial interface and position computed by SWI after 200 and 400 days

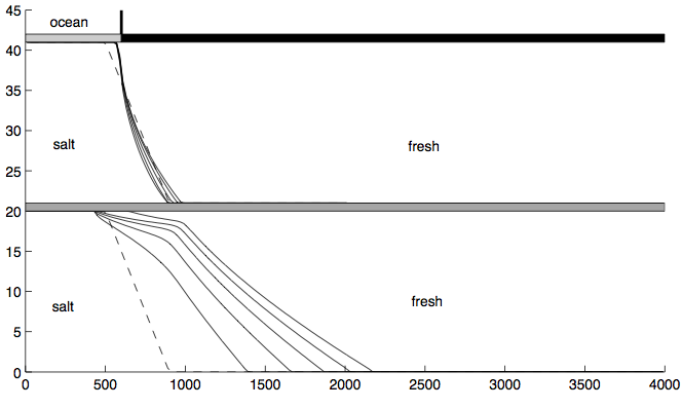


(a) SWI-manual

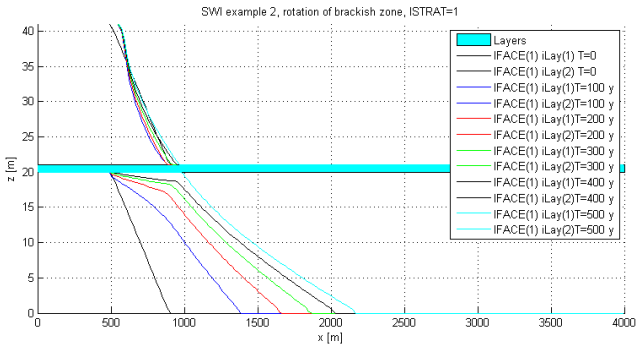


(b) mfLab

Figure 10.2: 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 10.3: Initial position and position of the interface at 100 year intervals as computed through *mfLab*

10.4 SWI example 4, coast with a well

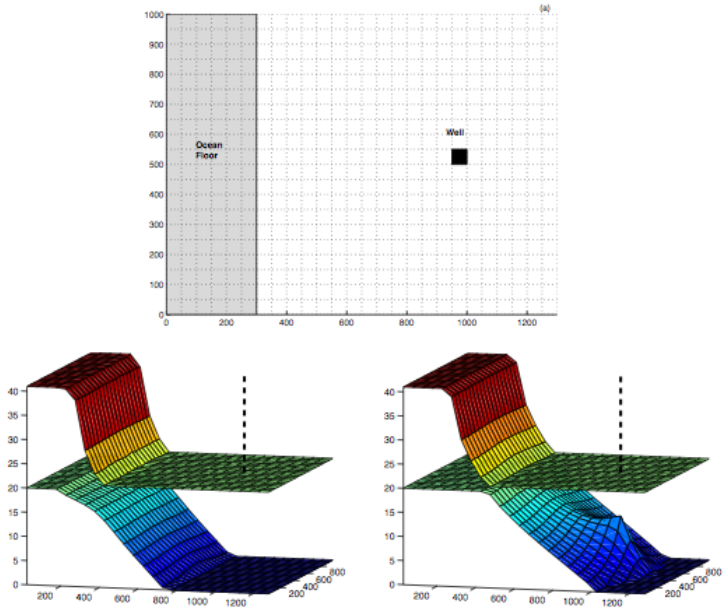
Example 4, figure 10.4, concerns two coastal aquifers with a confining bet in between where the top aquifer is in full contact with the ocean floor over a distance of 300 m. The situation is thus similar to that of example 3. However, there is a well inland extraction water. Due to the combined effect 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 10.4. For details and the used numbers see the description in the SWI manual and *mf_ adapt* in the example directory.

10.5 SWI example 5, square island with well

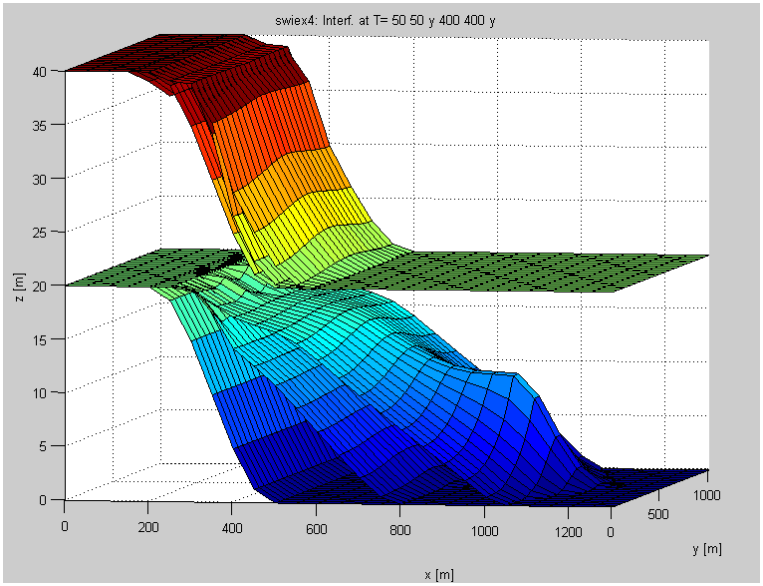
The fifth and last example in the SWI manual concerns a square island surrounded by ocean. Its setup is shown in figure 10.5. Recharge causes a fresh water lens to develop and maintain, while water is extracted somewhat artificially over part of the area of the island as indicated in the figure below 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 10.6 shows the results as computed through *mfLab*. The figure shows the contours of the elevation of both interfaces and in a cross section.



(a) SWI-manual: setup, initial interfaces and results of intrusion and extraction after 400 years



(b) *mfLab*

Figure 10.4: Interface movement and up-coning as computed through *mfLab*

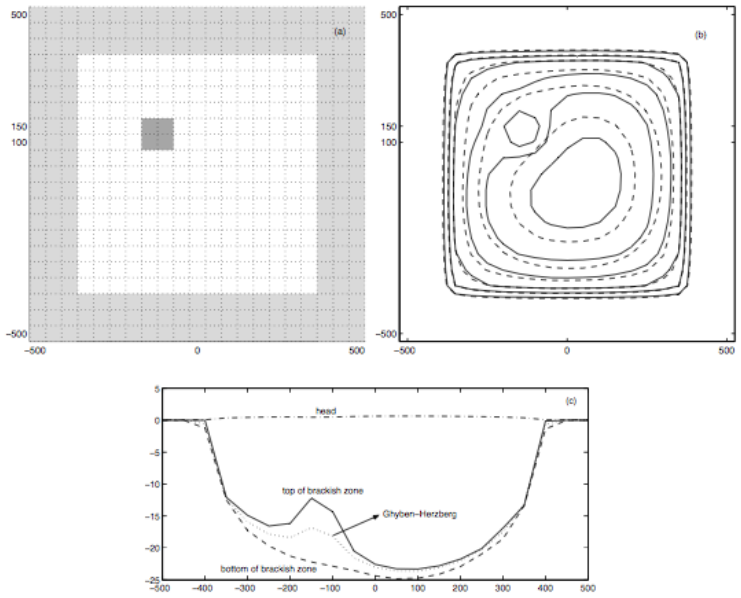
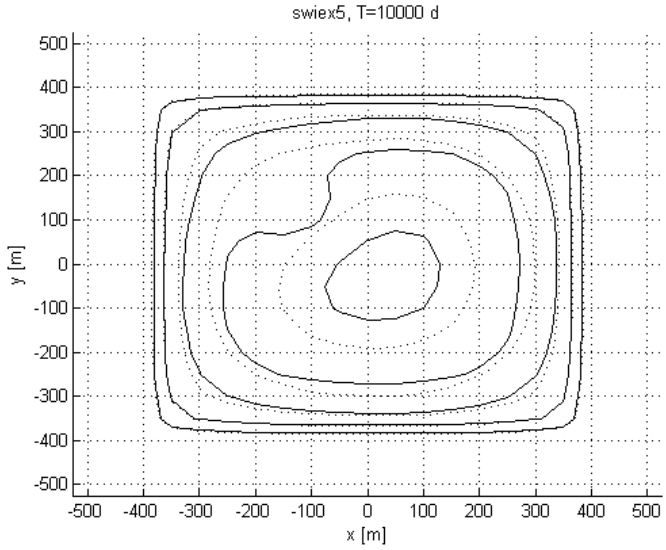
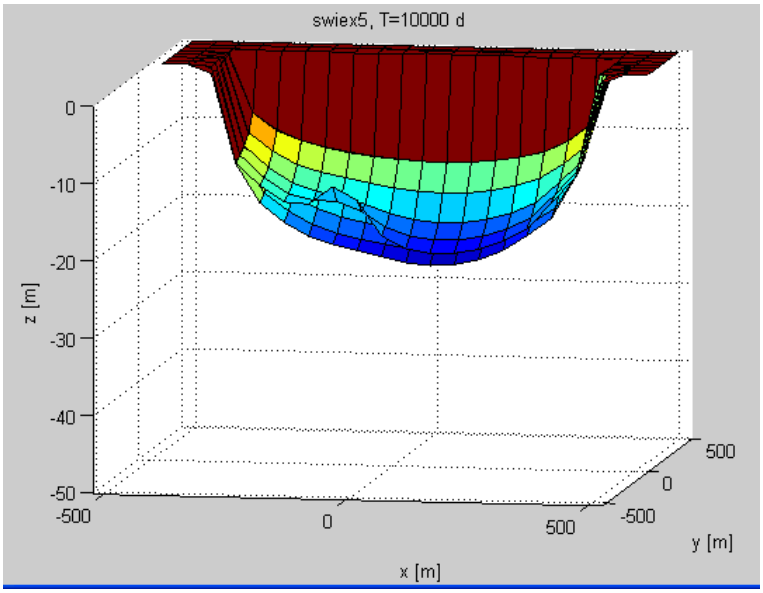


Figure 10.5: SWI example 5, problem from manual, square island with extraction



(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 10.6: SWI ex5: Computation through *mfLab* visualized as depth contours and 3D-cutoff

Chapter 11

examples/mf2005

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

Bibliography

[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.]

[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.