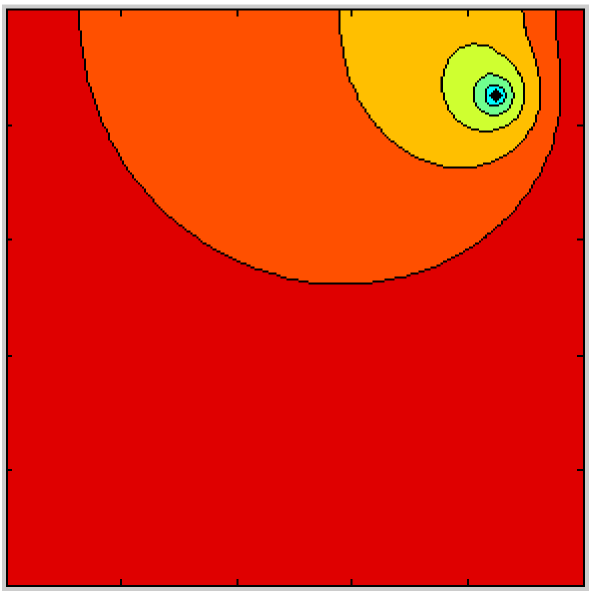
RESEARCH UNIT GROUNDWATER MODELING, GHENT UNIVERSITY, BELGIUM

MAxSym

A MATLAB Tool to Simulate Two-Dimensional Axi-Symmetric Groundwater Flow



Andy Louwyck

2011

Preface

MAxSym was developed during my PhD research at Ghent University under the mentoring of Prof. Dr. Luc C. Lebbe. The tool could be regarded as a further development of the AS2D code he developed during the eighties. The AS2D code is written in FORTRAN-77, runs under MS-DOS or UNIX, uses fixed format ASCII files for data exchange, and creates HP-GL/2 files as graphical output. Needless to say that integrating AS2D into a more interactive and user-friendly environment suggested itself during my research.

At first, I wrote some MATLAB functions to write and read the AS2D input and output files, and to invoke the AS2D executables using MATLAB function `dos`. Although this worked fine, I still decided to re-write the AS2D code, mainly because the latter does not support radial variation of hydraulic parameters. The iterative ADI solver was written in FORTRAN-95 and compiled to a MEX function, additional code was written in MATLAB. This new code was the core of the OGMA-RF program I developed together with my colleague Dr. A. Vandenbohede. OGMA-RF is a numerical, axi-symmetric model to simulate and analyze groundwater flow and solute transport provided with a user-friendly MATLAB GUI. The program was presented at the MODELCARE-2007 congress and a few studies applying OGMA-RF were published recently.

Several reasons have encouraged the development of MAxSym, however. The main reason was the hybrid numerical approach applied by Lebbe turned out to give inaccurate results when simulating drawdown within and near the well. Therefore, the finite-difference method was used instead, and the SIP solver was implemented additionally, because the iterative ADI solver was found to be inefficient in case of more complex models. Both ADI and SIP were written in ANSI C instead of FORTRAN, as the MathWorks suddenly decided to stop supporting the LAHEY compiler, which I used to compile FORTRAN code into MEX functions. Another reason to develop MAxSym was I wanted to take advantage of the object oriented programming capabilities provided with MATLAB releases R2008a and higher. It also seemed like a good idea to extend the model with features such as initial drawdowns, discharges, inactive and constant-head rings, and stress periods to cover a broader range of axi-symmetric models. As a consequence, the use of MAxSym is not limited to the simulation of constant-discharge pumping tests, but also includes simulation of variable-discharge pumping tests, slug tests, steady-state models and models considering areal infiltration.

The MAxSym code has been tested carefully and I'm grateful to Prof. Dr. M. Bakker and Dr. A. Vandenbohede for their help with the verification of MAxSym against MODFLOW and several analytical models. Dr. A. Vandenbohede is also thanked for reviewing the manual. Despite these efforts of testing and reviewing, it is still possible the MAxSym code or manual contain errors. Possible bugs can be reported by sending an email to andy.louwyck@gmail.com. Any other comments and suggestions will be received with great interest.

Andy Louwyck

Contents

List of Tables and Figures

Introduction

MAxSym is a MATLAB tool for the simulation of two-dimensional, non-uniform, axi-symmetric groundwater flow in saturated, porous media. The model is axi-symmetric, which means it is used to simulate flow towards or away from an extraction well, which is induced by extracting or injecting water from or into that well. The extraction or injection may be instantaneous, constant, or time-varying to allow simulation of slug tests and all kinds of pumping and injection tests. Unlike many analytical models, MAxSym can deal with wells of both small and large diameter and the well casing can be taken into account.

The model is two-dimensional because it considers both radial and vertical components of the flow. By allowing hydraulic parameters to vary in the vertical direction, it is possible, for instance, to simulate flow in multi-layer systems. Non-uniform flow may be simulated by varying hydraulic parameters in the radial direction, which is useful, for instance, to consider the effect of a finite-thickness skin. The aquifer system under consideration may be confined or unconfined. If the system is unconfined, MAxSym simulates saturated flow and takes into account the time-varying saturated thickness of the top layer. Flow in the unsaturated zone on top of the system is ignored, but it is possible to define areal infiltration in the top layer. Steady state as well as transient flow conditions may be considered, the former being useful to simulate permanent groundwater extractions, the latter being necessary for the simulation of aquifer tests.

The governing partial differential equation is solved using a finite-difference approach, which is outlined in detail in the first chapter. The approach is also discussed briefly by Louwyck et al. (in press). Readers who are familiar to finite-difference groundwater models may skip this chapter and jump to the next chapter explaining how to use MAxSym. The latter is followed by a large number of examples. Both steady and unsteady state model cases are given, and most of them are compared with the corresponding analytical solution described in the literature. Verification of the code against MODFLOW-2005 (Harbaugh, 2005) and several analytical solutions was also done by Louwyck et al. (in press). Accurate results are obtained when radial distance, vertical distance, and simulation time are discretized appropriately. Although any discretization scheme can be defined with MAxSym, the scheme proposed by Louwyck et al. (in press) is applied for all examples, as it is proven to be accurate.

Besides two-dimensional, axi-symmetric models, it also possible to construct simple, three-dimensional models through application of superposition. After explaining briefly the theory behind superposition, examples of steady and unsteady state, three-dimensional models for multiple wells in multi-layer systems are given. These models may also incorporate areal infiltration, and applying the method of images, it is even possible to include lateral bounds. Finally, a short introduction about inverse modeling and parameter identification is presented, and as an example, a synthetic pumping test and a synthetic slug test are interpreted using non-linear regression.

The design and implementation of MAxSym is object-oriented. In this way, developers may apply the inheritance principle to extend MAxSym without having to modify the original code. Therefore, a separate chapter is dedicated to MAxSym’s object-oriented design and a full description of all MAxSym classes and functions is given in the appendices. Additionally, the source code files are well documented to improve code readability.

It is assumed that the reader is familiar to axi-symmetric groundwater models and the MATLAB environment. Experience of finite-difference modeling is an advantage, though not required. Knowledge of the fundamentals of object-oriented programming is needed to get a good understanding of MAxSym's object-oriented structure, but not required for modelers who only use MAxSym without extending it. Familiarity with the Unified Modeling Language (UML, see www.uml.org) is necessary to fully understand the UML diagrams presented in the manual. These diagrams were created using the Community Edition of Visual Paradigm version 8.2 (see www.visual-paradigm.com), which supports the latest UML 2.x diagrams.

MAxSym License

MAxSym is free software for academic and research purposes only. Publications of any kind using MAxSym must refer to this manual and to Louwyck et al. (in press).

It is allowed to redistribute MAxSym under the terms of the "New BSD license", a certified open source license: http://www.opensource.org/trademark. Please contact andy.louwyck@gmail.com to use MAxSym for commercial purposes. All redistributions must retain the following copyright notice and disclaimer:

Installation of MAxSym

MAxSym is written in MATLAB and consequently, it can only be run in the MATLAB environment, a numerical computing environment developed and sold by the MathWorks (www.mathworks.com). MAxSym has an object-oriented design and its implementation uses MATLAB classes. Therefore, a MATLAB release must be used that supports the definition of classes (i.e. MATLAB R2008a or higher). The use of additional MATLAB toolboxes is not required to run MAxSym.

All MAxSym files must be stored in the MAxSym package folder, i.e. a folder with name “+MAxSym”. To run MAxsym, this folder must be a subfolder of the current folder or a folder belonging to the MATLAB search path. For example, when the MAxSym m-files are copied to the MAxSym package folder “C:\MATLABTools\+MAxSym\”, then the folder “C:\MATLABTools\” must be the current folder or added to the search path. Adding a folder to the search path is done using the function `addpath` or via menu “File > Set Path…”. See the MATLAB Help for more information about package folders and adding folders to the MATLAB search path.

MAxSym provides different solvers, which are used to solve the system of finite-difference equations. The solvers are coded in MATLAB, but to enhance performance, they were also written in ANSI C. The C routines, however, can only be used after being compiled to binary MATLAB executables, called mex-files. These are dynamically-linked subroutines that the MATLAB interpreter loads and executes as if they were built-in functions. More information about mex-files is found in the MATLAB Help. Here, it is only explained how to compile the MAxSym C routines to mex-files.

First, an appropriate C compiler must be installed. If a 32-bit release of MATLAB is used, then the LCC C compiler included with MATLAB may be used. For 64-bit releases of MATLAB, the MathWorks’ website provides a list of supported C compilers. MAxSym C routines use the `mwSize` type instead of integer types to represent size values. Hence, it should be possible to compile the routines in a 64-bit environment. However, at the moment of writing this manual, compilation of the C routines was only verified by the author for the windows’ 32-bit LCC compiler. The mex-files *.mexw32 in the MAxSym package folder were compiled under MATLAB R2011a (32bit). Code testing was done for MATLAB releases R2010b and R2011a (32bit).

Once an appropriate C compiler is installed, it must be located by the MATLAB environment using the command `mex` at the MATLAB prompt:

```
>> mex -setup
```

The most straightforward way is to let `mex` locate the installed compilers and then select the appropriate one from the list of identified compilers.

Finally, the `mex` function is used to compile the 4 MAxSym C routines that implement ADI and SIP for both confined and unconfined flow. Since the C source files are in the MAxSym package folder and the corresponding mex-files must also be compiled to this folder, the current folder must first be set to the package folder before executing the following commands:

```
>> mex mexADIconf.c

>> mex mexSIPconf.c
>> mex mexADIunconf.c
```

```
>> mex mexSIPunconf.c
```

When compilation was successfully, the mex-files are compiled to the MAxSym package folder. Compilation must only be done once, as long as the same release is used on the same platform. If a new release is used, then the source files must be re-compiled.

Users who are not able to compile the C routines are obliged to use the MATLAB routines, unless they run MAxSym under MATLAB R2011a (32bit). The MATLAB routines yield the same results, but they are several times slower than the mex routines. Therefore, MAxSym uses the mex-file solvers by default. Switching to the MATLAB solvers is done by setting the `mex` property of the `Solver` object to `false`, as will be explained further.

Finite-difference approach

MAxSym is developed to simulate transient, two-dimensional, non-uniform, axi-symmetric groundwater flow in saturated, porous media, which is treated by the following partial differential equation:

$$\frac{\partial}{\partial r}\left(K^r_{(z,r)}\frac{\partial s_{(z,r,t)}}{\partial r}\right)+\frac{K^r_{(z,r)}}{r}\frac{\partial s_{(z,r,t)}}{\partial r}+\frac{\partial}{\partial z}\left(K^z_{(z,r)}\frac{\partial s_{(z,r,t)}}{\partial z}\right)=S^s_{(z,r)}\frac{\partial s_{(z,r,t)}}{\partial t} \tag{1}$$

where *r* is the radial distance [L], *z* is the vertical distance [L], *t* is time [T], *s* is the drawdown [L], K^r is the radial component of hydraulic conductivity [L/T], K^z is the vertical component of hydraulic conductivity [L/T], and S^s is the specific elastic storage coefficient [L^{-1}]. If the right-hand side of differential equation [1] is zero, drawdown is not a function of time *t* and flow conditions are steady state. Otherwise, flow conditions are transient. If the third term on the left-hand side of [1] is zero, drawdown is not a function of vertical distance *z* and radial flow is strictly horizontal. Otherwise, flow is two-dimensional.

Drawdown *s(z,r,t)* is defined as the difference between the head *h(z,r,t)* [L] and the steady initial head $h_0(z,r)$ [L] before the test starts at *t = 0*:

$$s_{(z,r,t)} = h_{(z,r,t)} - h_{0(z,r)} \quad t \geq 0 \tag{2}$$

According to this definition, drawdown is positive if the head at time *t* is larger than the steady initial head, and negative if the head at time *t* is smaller.

To obtain a unique solution for partial differential equation [1], initial conditions for time *t = 0* and boundary conditions must be defined. These conditions depend on the problem under consideration. Initial and boundary conditions for pumping tests and slug tests, for instance, are presented by Louwyck et al. (in press). Many solution techniques exist to solve partial differential equation [1] subject to a given set of initial and boundary conditions. MAxSym applies the finite-difference method, which offers the ability to simulate the variation of the saturated thickness in unconfined top layers, and the possibility to include radial variation of hydraulic parameters.

The finite-difference method comes down to transforming partial differential equation [1] into a finite set of linear algebraic equations by replacing its partial derivatives by finite differences. The latter is done by performing a discretization of time *t*, radial distance *r*, and vertical distance *z*. The set of linear equations is solved numerically by applying an iterative routine. MAxSym has two solvers implemented: the iterative Alternating Direction Implicit method (ADI) and the Strongly Implicit Procedure (SIP).

Model discretization

Radial distance *r* is discretized by subdividing the groundwater reservoir into n_r coaxial rings. The width of the *j*-th ring is determined by inner radius $r_b(j)$ and outer radius $r_b(j+1)$. The *j*-th ring contains the *j*-th nodal circle with radius *r(j)* equal to the geometric mean of inner and outer radius:

$$r_{(j)} = \sqrt{r_{b(j)}\, r_{b(j+1)}} \quad j = 1,2,\ldots,n_r \tag{3}$$

Rings are numbered from inside to outside, and the axis of the coaxial rings is located at *r = 0*. Radius *r_b(1)* is the inner radius of the model grid, which is greater than 0, and radius *r_b(n_r+1)* is the outer grid radius.

If vertical flow is considered, vertical distance *z* is discretized by subdividing the groundwater reservoir into n_z layers. The thickness of layer *i* is *D(i)*, which is equal to the difference of top level *z_b(i)* and bottom level *z_b(i+1)* of the layer. The nodal circles are positioned at the center of each layer, at vertical distance *z(i)*:

$$z_{(i)} = \left(z_{b(i)} + z_{b(i+1)}\right)/2 \quad i = 1,2,\dots,n_z \qquad [4]$$

Layers are numbered from top to bottom, and the bottom of the groundwater reservoir *z_b(n_z+1)* corresponds to *z = 0*. A vertical cross section of the axi-symmetric grid is presented in Figure 1.

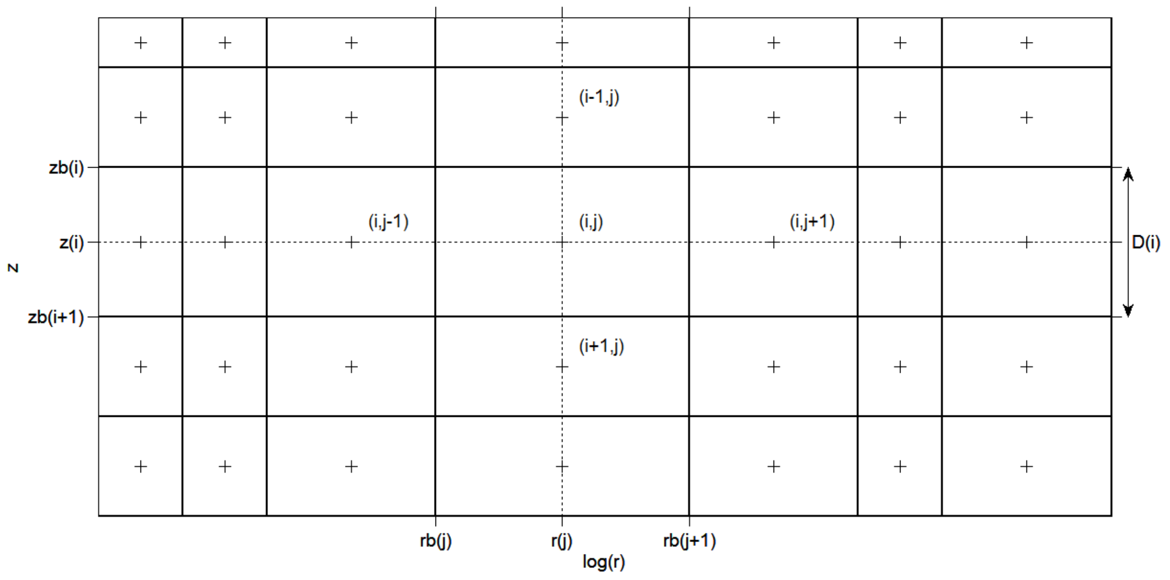


Figure 1. Cross section of MAxSym's axi-symmetric grid consisting of layers (rows), rings (columns), and nodal circles (crosses). Ring (i,j) and adjacent rings are indicated by indices. See text for explanation of variables. Note that the x-axis indicates logarithm of radial distance r.

If the simulation is transient, time *t* is subdivided into n_{dt} time steps, where the *k*-th time step *Δt(k)* starts at time *t(k)* and ends at time *t(k+1)*:

$$\Delta t_{(k)} = t_{(k+1)} - t_{(k)} \quad k = 1,2,\dots,n_{dt} \qquad [5]$$

The total number of simulation times *t(k)* is *n_t = n_{dt} + 1*. The initial time *t(1)* is equal to 0 and corresponds to the initial state of the model for which the initial conditions are stated.

To reduce the number of nodes and stress periods, it is common to define radii of nodal circles *r(j)* and simulation times *t(k)* recursively (e.g. Reilly and Harbaugh, 1993; Lebbe, 1999; Louwyck et al., in press):

$$r_{(j)} = r_{(j-1)}a_r \quad j = 2,3,\dots,n_r \qquad [6]$$

$$t_{(k)} = t_{(k-1)}a_t \quad k = 2,3,\dots,n_t \qquad [7]$$

where a_r and a_t are constants greater than 1. The recursive equations [6] and [7] come down to discretizing *r* and *t* in logarithmic space:

$$log\, r_{(j)} = log\, r_{(j-1)} + log\, a_r \qquad j = 2,3, \dots, n_r \tag{8}$$

$$log\, t_{(k)} = log\, t_{(k-1)} + log\, a_t \qquad k = 2,3, \dots, n_t \tag{9}$$

Applying recursive definition [6] for *r(j)* to definition [3], the inner and outer radii of ring *j* can also be expressed as a function of the expansion factor a_r:

$$r_{b(j)} = r_{(j)} {a_r}^{-1/2} \quad and \quad r_{b(j+1)} = r_{(j)} {a_r}^{1/2} \quad for\ j = 1,2, \dots, n_r \tag{10}$$

This explains why the geometric mean is used to position the nodal circles inside the rings:

$$log\, r_{(j)} = \left(log\, r_{b(j)} + log\, r_{b(j+1)}\right)/2 \tag{11}$$

which indicates the nodal circle is at the center of the ring if radial distance is transformed logarithmically.

The advantages of applying the recursive discretization schemes are discussed by Louwyck et al. (in press). The MATLAB function `logspace` may be used to discretize *r* and *t* in logarithmic space.

Once the axi-symmetric grid is defined, hydraulic parameters are assigned to each ring *(i,j)*, initial conditions at initial time *t(1)* are specified for each ring *(i,j)*, and boundary conditions are defined for each ring *(i,j)* and for each time step *k*. To facilitate the input of time-dependent boundary conditions, time steps are grouped into stress periods, as will be explained further below. The initial conditions are referred to as starting conditions in case of a steady state simulation, since there is no time dimension in such models. The input parameters and conditions are used to formulate a finite-difference equation for each ring *(i,j)* and each time step *k*, and the set of finite-difference equations is solved to obtain drawdown *s(i,j,k)* at each level *z(i)*, at each distance *r(j)*, and at each time *t(k)*.

Finite-difference equations

The finite-difference equations are formulated applying Darcy's law and the continuity principle. The former is used to calculate flow into and out of ring *(i,j)* at time *t(k)*, the latter states the net flow at time *t(k)* is equal to the net volume of water added to or removed from ring *(i,j)* during time step *Δt(k-1)* due to storage change and/or external sources or sinks defined for ring *(i,j)*.

Flow equations

To calculate radial flow between ring *(i,j)* and ring *(i,j+1)*, hydraulic conductivity is averaged first using the Thiem (1906) equation:

$$K_{(i,j+1)}^{reff} = ln(r_{(j+1)}/r_{(j)}) \left(\frac{ln(r_{b(j+2)}/r_{b(j+1)})}{2K_{(i,j+1)}^{r}} + \frac{ln(r_{b(j+1)}/r_{b(j)})}{2K_{(i,j)}^{r}} \right)^{-1}$$

$$with\ j = 1,2, \dots, n_r - 1\ and\ i = 1,2, \dots, n_z \tag{12}$$

where *K^r^(i,j)* is the radial component of hydraulic conductivity in ring *(i,j)*, and *K^reff^(i,j+1)* is the effective conductivity in layer i between nodes *j* and *j+1*. Radial discharge Q^r [L³/T] between ring *(i,j)* and ring *(i,j+1)* at time *t(k)* is calculated according to the Thiem equation:

$$Q^{r}_{(i,j+1,k)} = 2\pi K^{reff}_{(i,j+1)} D_{(i)} \frac{s_{(i,j+1,k)} - s_{(i,j,k)}}{ln(r_{(j+1)}/r_{(j)})}$$

$$with\ j = 1,2,\dots,n_r - 1;\ i = 1,2,\dots,n_z;\ k = 1,2,\dots,n_t \quad [13]$$

If radial resistance *C^r^(i,j+1)* [T] between ring *(i,j)* and ring *(i,j+1)* is defined, radial discharge Q^r is defined as:

$$Q^{r}_{(i,j+1,k)} = 2\pi r_{b(j+1)} D_{(i)} \frac{s_{(i,j+1,k)} - s_{(i,j,k)}}{C^{r}_{(i,j+1)}}$$

$$with\ j = 1,2,\dots,n_r - 1;\ i = 1,2,\dots,n_z;\ k = 1,2,\dots,n_t \quad [14]$$

Theoretically, it is assumed the radial distance between the nodes is infinitesimal small in the latter case. In practice, the distance must be negligibly small to apply flow equation [14].

The constant term in radial flow equations [13] and [14] is the radial conductance Q^{rc} [L²/T], so:

$$Q^{r}_{(i,j+1,k)} = Q^{rc}_{(i,j+1)}\left(s_{(i,j+1,k)} - s_{(i,j,k)}\right)$$

$$with\ j = 1,2,\dots,n_r - 1;\ i = 1,2,\dots,n_z;\ k = 1,2,\dots,n_t \quad [15]$$

To calculate vertical flow between ring *(i,j)* and ring *(i+1,j)*, vertical resistance C^z [T] is derived first using the harmonic mean of vertical conductivities:

$$C^{z}_{(i+1,j)} = \frac{1}{2}\left(\frac{D_{(i)}}{K^{z}_{(i,j)}} + \frac{D_{(i+1)}}{K^{z}_{(i+1,j)}}\right)$$

$$with\ i = 1,2,\dots,n_z - 1\ and\ j = 1,2,\dots,n_r \quad [16]$$

where *K^z^(i,j)* is the vertical component of hydraulic conductivity for ring *(i, j)*. Note that MAxSym allows the modeler to define vertical resistances between rings directly. Vertical discharge Q^z [L³/T] between ring *(i,j)* and ring *(i+1,j)* at time *t(k)* is defined as:

$$Q^{z}_{(i+1,j,k)} = \pi\left(r^{2}_{b(j+1)} - r^{2}_{b(j)}\right) \frac{s_{(i+1,j,k)} - s_{(i,j,k)}}{C^{z}_{(i+1,j)}}$$

$$with\ i = 1,2,\dots,n_z - 1;\ j = 1,2,\dots,n_r;\ k = 1,2,\dots,n_t \quad [17]$$

Vertical flow is calculated only if the model grid has more than one layer. The constant term in vertical flow equation [17] is called the vertical conductance Q^{zc} [L²/T]:

$$Q^{z}_{(i+1,j,k)} = Q^{zc}_{(i+1,j)}\left(s_{(i+1,j,k)} - s_{(i,j,k)}\right)$$

$$with\ i = 1,2,\dots,n_z - 1;\ j = 1,2,\dots,n_r;\ k = 1,2,\dots,n_t \quad [18]$$

The volume of water per unit of time Q^S [L³/T] due to storage change in ring *(i,j)* for time step *Δt(k)* is approximated by:

$$Q^s_{(i,j,k)} = \pi\left(r^2_{b(j+1)} - r^2_{b(j)}\right) S^s_{(i,j)} D_{(i)} \frac{s_{(i,j,k+1)} - s_{(i,j,k)}}{\Delta t_{(k)}}$$

$$with\ i = 1,2, \dots, n_z;\ j = 1,2, \dots, n_r;\ k = 1,2, \dots, n_t - 1 \qquad [19]$$

where *S^s(i,j)* is the specific elastic storage coefficient for ring *(i,j)*. Storage change is considered only if the simulation is transient. The constant, time-independent term in storage change equation [19] is indicated as Q^{ssc} [L²]:

$$Q^s_{(i,j,k)} = Q^{ssc}_{(i,j)} \frac{s_{(i,j,k+1)} - s_{(i,j,k)}}{\Delta t_{(k)}}$$

$$with\ i = 1,2, \dots, n_z;\ j = 1,2, \dots, n_r;\ k = 1,2, \dots, n_t - 1 \qquad [20]$$

Initial conditions

Initial drawdown *s(i,j,1)* at initial time *t(1) = 0* is defined as:

$$s_{(i,j,1)} = h_{(i,j,1)} - h_0$$

$$with\ i = 1,2, \dots, n_z\ and\ j = 1,2, \dots, n_r \qquad [21]$$

where *h(i,j,1)* is the head in ring *(i,j)* at the start of the test, and h_0 is the steady head in ring *(i,j)* before the test starts. The latter is the same for all rings and equal to the total saturated thickness D_{tot} of the groundwater reservoir or:

$$h_0 = D_{tot} = \sum_{i=1}^{n_l} D_{(i)} \qquad [22]$$

If initial drawdown *s(i,j,1)* differs from zero, an instantaneous head change is realized in ring *(i,j)* at the start of the test. A positive initial drawdown corresponds to an instantaneous head rise, whereas a negative initial drawdown indicates an instantaneous head fall.

Initial drawdown *s(i,j,1)* is defined by the modeler and MAxSym calculates radial and vertical flow components *Q^r(i,j,1)* and *Q^z(i,j,1)* accordingly, although they are not required to solve the system of equations. Note that storage change *Q^S(i,j,1)* is derived from initial drawdown *s(i,j,1)* and drawdown *s(i,j,2)* at the end of the first time step.

Boundary conditions

Grid boundaries

The boundaries of the axi-symmetric grid are impervious, which is expressed as:

$$Q^r_{(i,1,k)} = Q^r_{(i,n_r+1,k)} = Q^z_{(1,j,k)} = Q^z_{(n_z+1,j,k)} = 0$$

$$with\ i = 1,2, \dots, n_z;\ j = 1,2, \dots, n_r;\ k = 1,2, \dots, n_t \qquad [23]$$

If a groundwater reservoir of infinite lateral extent is assumed, the outer model boundary must be placed at large distance to ensure the no-flow conditions have a negligibly small influence on calculated drawdown.

The modeler may define additional impervious boundaries by setting rings inactive. Inactive rings are skipped during the iterative process of solving the system of equations. Hence, setting ring *(i,j)* inactive is equivalent to the following boundary conditions:

$$Q^r_{(i,j,k)} = Q^r_{(i,j+1,k)} = Q^z_{(i,j,k)} = Q^z_{(i+1,j,k)} = 0 \qquad [24]$$

Inactive rings are not part of the model domain, consequently, parameters and other boundary conditions defined for these rings are irrelevant, as is drawdown and storage change. However, since MAxSym is matrix oriented, inactive rings still require an input value, which is ignored during the calculation process. Resulting output matrices also contain a value for inactive rings: drawdown *s* is set `NaN` and storage change Q^s is set `0`. Note that the impervious grid boundaries may also be considered as inactive rings at positions *(0,j)*, *(n_z+1,j)*, *(i,0)*, and *(i,n_r+1)*.

Ring *(i,j)* may also have a constant head *H* [L] defined, which is equivalent to the following boundary condition:

$$s_{(i,j,k)} = H_{(i,j)} - h_0 \qquad [25]$$

Constant heads are specified by setting initial drawdowns. Because drawdown in a constant-head ring does not change during simulation, storage change Q^s always is zero, and specific storage coefficient $S^s(i,j)$ is ignored. Radial and vertical conductivity input, however, are relevant, because MAxSym needs to calculate flow between the constant-head ring and the adjacent rings. When solving the system of equations, equations that correspond to constant-head rings are skipped, but the constant-head terms in the equations of variable-head rings are not ignored.

Summarizing, the model grid consists of three types of rings:

- inactive, no-flow rings;
- active, constant-head rings;
- active, variable-head rings.

The first two type of rings are used to define the no-flow and the constant-head boundaries of the model.

Water table

The impervious grid boundaries state the groundwater reservoir is bounded below and above by an impervious layer. It is, however, possible to simulate flow in an unconfined system by modifying the flow equations for the rings in the top layer. The effect of a water table at the top of the groundwater reservoir is simulated by taking into account the time-varying saturated thickness of the top layer and by defining the specific yield S^y [-].

If the top layer is defined phreatic, the radial flow Q^r in this layer is defined as:

$$Q^r_{(1,j+1,k)} = 2\pi K^{reff}_{(1,j+1)} \left(D_{(1)} + \frac{s_{(1,j+1,k)} + s_{(1,j,k)}}{2} \right) \frac{s_{(1,j+1,k)} - s_{(1j,k)}}{ln(r_{(j+1)}/r_{(j)})}$$

$$with\ j = 1,2, \dots, n_r - 1\ and\ k = 1,2, \dots, n_t \qquad [26]$$

or, if radial resistance C^r is given:

$$Q^r_{(1,j+1,k)} = 2\pi r_{b(j+1)} \left(D_{(1)} + \frac{s_{(1,j+1,k)} + s_{(1,j,k)}}{2} \right) \frac{s_{(1,j+1,k)} - s_{(1,j,k)}}{C^r_{(1,j+1)}}$$

$$with\ j = 1,2, \dots, n_r - 1\ and\ k = 1,2, \dots, n_t \qquad [27]$$

Drawdown at the ring face with radius *r_b(j+1)* is calculated as the arithmetic mean of the drawdowns calculated at the adjacent nodal circles with radius *r(j)* and *r(j+1)*, and the resulting mean drawdown is used to correct the thickness *D(1)* of the top layer at the ring face. Using radial conductance Q^{rc}, the two equations [26] and [27] are summarized as:

$$Q^r_{(1,j+1,k)} = Q^{rc}_{(1,j+1)} \left(1 + \frac{s_{(1,j+1,k)} + s_{(1,j,k)}}{2D_{(1)}} \right) \left(s_{(1,j+1,k)} - s_{(1j,k)} \right)$$

$$with\ j = 1,2, \dots, n_r - 1\ and\ k = 1,2, \dots, n_t \qquad [28]$$

Concerning vertical flow Q^z, the vertical resistance between the top layers must also be corrected for the time-varying saturated thickness:

$$C^z_{(2,j,k)} = \frac{D_{(1)} + s_{(1,j,k)}}{2K^z_{(1,j)}} + \frac{D_{(2)}}{2K^z_{(2,j)}}$$

$$with\ j = 1,2, \dots, n_r\ and\ k = 1,2, \dots, n_t \qquad [29]$$

Here, the thickness of the top layer is considered at the position of the nodal circles, hence, drawdown at the nodal circles is used to correct the thickness. Using vertical conductance Q^{zc}, vertical flow between the top layers is written as:

$$Q^z_{(2,j,k)} = \left(\frac{1}{Q^{zc}_{(2,j)}} + \frac{s_{(1,j,k)}}{2\pi \left(r^2_{b(j+1)} - r^2_{b(j)} \right) K^z_{(1,j)}} \right)^{-1} \left(s_{(2,j,k)} - s_{(1,j,k)} \right)$$

$$with\ j = 1,2, \dots, n_r\ and\ k = 1,2, \dots, n_t \qquad [30]$$

Note that C^z is not corrected if it has been defined by the modeller.

Finally, the storage change Q^s is corrected for the time-varying thickness and specific yield is taken into account:

$$Q^s_{(1,j,k)} = \pi \left(r^2_{b(j+1)} - r^2_{b(j)} \right) \left[S^s_{(1,j)} \left(D_{(1)} + s_{(1,j,k)} \right) + S^y_{(j)} \right] \frac{s_{(1,j,k+1)} - s_{(1,j,k)}}{\Delta t_{(k)}}$$

$$with\ j = 1,2, \dots, n_r\ and\ k = 1,2, \dots, n_t - 1 \qquad [31]$$

and the following time-independent term Q^{syc} [L²] related to specific yield is introduced:

$$Q^{syc}_{(j)} = \pi\left(r^2_{b(j+1)} - r^2_{b(j)}\right) S^y_{(j)} \qquad j = 1,2,\ldots,n_r \qquad [32]$$

Using this term and term Q^{ssc}, the storage change equation is summarized as:

$$Q^s_{(1,j,k)} = \left[Q^{ssc}_{(1,j)}\left(1 + \frac{s_{(1,j,k)}}{D_{(1)}}\right) + Q^{syc}_{(j)}\right] \frac{s_{(1,j,k+1)} - s_{(1,j,k)}}{\Delta t_{(k)}}$$

$$with\ j = 1,2,\ldots,n_r\ and\ k = 1,2,\ldots,n_t - 1 \qquad [33]$$

At this moment, MAxSym only allows the top layer to be phreatic. Hence, specific yield is defined for the top layer only. Correction of the flow equations is done during the iterative process of solving the system of equations for each time step. Because of the non-linearity of the corrected flow equations, the process will need more iterations to converge to a solution. After each iteration, it is verified if the corrected thickness is still greater than zero for all rings. If not, the calculation process is aborted.

Stresses

An instantaneous head change $s_0(i,j,k)$ [L] at a certain time *t(k)* can be defined for any active ring *(i,j)*, including constant head rings. Defining an instantaneous head change comes down to the following boundary condition:

$$s_{(i,j,k)} = h_{(i,j,k)} + s_{0(i,j,k)} - h_0 \qquad [34]$$

where $h(i,j,k) - h_0$ is the drawdown calculated at time *t(k)*. So, the instantaneous head change is simply added to the calculated drawdown, where a positive head change corresponds to an instantaneous head rise and a negative head change to an instantaneous head fall. Specifying initial head changes for constant-head rings allows the modeler to define time-varying specified heads.

Another time dependent stress is discharge *Q(i,j,k)* [L^3/T], which can be defined for any variable-head ring *(i,j)*. Discharge *Q(i,j,k)* is defined for time step *k*, which means a volume of water equal to *Q(i,j,k) x Δt(k)* is added to or removed from ring *(i,j)* between times *t(k)* and *t(k+1)*. Water is added to the ring if discharge is negative and removed from the ring if discharge is positive. Mathematically, *Q(i,j,k)* is an extra term in the finite-difference equation for ring *(i,j)* and time step *k*.

To make the input of time dependent stresses easier, time steps are grouped into stress periods. During a stress period, discharges are constant for all time steps which are part of the stress period. Instantaneous head changes are defined at the beginning of a stress period, and therefore, they are also referred to as initial drawdowns. A stress period consisting of *m* time steps starting with time step *k*, starts at time *t(k)* and ends at time *t(k+m)*. If the stress period defines a discharge *Q* for ring *(i,j)*, then:

$$Q_{(i,j,k)} = Q_{(i,j,k+1)} = \ldots = Q_{(i,j,k+m-1)} = Q \qquad [35]$$

If the stress period defines an initial drawdown s_0 for ring *(i,j)*, then:

$$s_{0(i,j,k)} = s_0 \quad and \quad s_{0(i,j,k+1)} = s_{0(i,j,k+2)} = \ldots = s_{0(i,j,k+m-1)} = 0 \qquad [36]$$

Initial drawdowns for the first stress period hold for initial time *t(1)*, hence, they define the initial conditions:

$$s_{0(i,j,1)} = s_{(i,j,1)} = h_{(i,j,0)} - h_0 \qquad [37]$$

Initial drawdown and discharge stresses are used to define head specified and discharge specified wells respectively, as will be explained in the next paragraph. Areal infiltration into the top layer is defined using discharges. If the recharge flux for ring *(1,j)* is *N(j,k)* during time step *k*, the corresponding discharge is defined as:

$$Q_{(1,j,k)} = N_{(j,k)}\pi\left(r_{b(j+1)}^2 - r_{b(j)}^2\right) \qquad [38]$$

Recall that *N* is negative if water is added to the groundwater reservoir.

Well

Most axi-symmetric models consider a well at the center of the model grid. Analytically, the well face coincides with the inner model boundary and drawdown at the face of the well screen is assumed equal to the change in head *H(t)* within the well:

$$s_{(z,r_w,t)} = H_{(t)} \quad t \geq 0,\ z_1 \leq z \leq z_2 \qquad [39]$$

where r_w is the radius of the well screen, which is situated between levels z_1 and z_2. When $z_1 = 0$ and $z_2 = D_{tot}$, the well is fully penetrating. The initial condition at the well face is expressed as:

$$s_{(z,r_w,0)} = \begin{cases} H_0 & z_1 \leq z \leq z_2 \\ 0 & elsewhere \end{cases} \qquad [40]$$

where H_0 is the initial head change within the well at $t = 0$. The boundary conditions at the well face hold for $t > 0$:

$$2\pi r_w \int_{z_1}^{z_2} K_{(z,r_w)}^r \frac{\partial s_{(z,r_w,t)}}{\partial r} dz = -Q_{(t)} - \pi r_w^2 \frac{\partial s_{(z,r_w,t)}}{\partial t} \quad z_1 \leq z \leq z_2$$

$$\frac{\partial s_{(z,r_w,t)}}{\partial r} = 0 \quad z < z_1\ or\ z > z_2$$

[41]

The first condition means that the rate of flow of water through the well screen is equal to the sum of the pumping rate *Q(t)* [m³/d] and the rate of decrease or increase in volume of water within the well. Elsewhere, the well face is impermeable which is expressed by the last condition.

In the axi-symmetric grid, the well is represented by the inner rings *(i,1)*, hence, $r_b(2)$ is set equal to well radius r_w and drawdown *s(i,1,k)* corresponds to the drawdown within the well. The inner grid radius $r_b(1)$ must be greater than 0 to avoid a nodal circle with zero radius, for which drawdown is infinitely large. Moreover, drawdown in the first ring must approximate drawdown at the well face, so it is recommended to define a first ring of small width to ensure *r(1)* is close to $r_b(2) = r_w$.

The finite-difference approximation of initial condition [40] at time step *k = 1* is written as:

$$s_{(i,1,1)} = s_{0(i,1,1)} = \begin{cases} H_o & i = p \\ 0 & otherwise \end{cases} \qquad [42]$$

where *p* is the index of the layer in which the well screen is situated. The initial condition is defined by defining initial drawdown $s_0(p,1,1)$. The boundary conditions at the well face hold for $k > 1$ and are reformulated as:

$$Q^r_{(i,2,k+1)} = \begin{cases} -Q_{(i,1,k)} - \pi r^2_{b(2)} \dfrac{s_{(i,1,k+1)} - s_{(i,1,k)}}{\Delta t_{(k)}} & i = p \\ 0 & i \neq p \end{cases} \qquad [43]$$

In layers where no screen is present, the well face is impermeable, which is obtained by setting rings *(i,1)* inactive if $i \neq p$. Radial flow through the outer face of ring *(p,1)* represents the flow through the well screen and considers well discharge and wellbore storage. Well discharge is assigned to the discharge *Q(p,1,k)* of the first ring *(p,1)*, and wellbore storage is taken into account by setting the specific storage coefficient:

$$S^s_{(p,1)} = \frac{\pi r^2_{b(2)}}{\pi\left(r^2_{b(2)} - r^2_{b(1)}\right)D_{(p)}} \qquad [44]$$

Substituting expression [44] into equation [19] for the storage change Q^s indeed results to the wellbore storage term. The radial conductivity $K^r(p,1)$ is set equal to $K^r(p,2)$.

If the stressed layer *p* is discretized into n_p sub-layers of thickness $D(p)/n_p$, the right-hand side of equation [43] must be divided by n_p for each of the n_p sub-layers, and the vertical resistance C^z between the first rings in these sub-layers must be sufficiently small to ensure all rings calculate the same drawdown. Defining a large value for the vertical resistances may be done to implement multi-level wells extracting or injecting water at different levels separated by packers.

If a slug test is simulated, $s_0(p,1,1)$ differs from zero, while *Q(p,1,k)* is zero for all time steps *k*. In case of a pumping test, $s_0(p,1,1)$ is zero, and *Q(p,1,k)* is positive if water is extracted during time step *k* and negative if water is injected. It is allowed to define a nonzero value for both s_0 and *Q* to simulate the combined effect of pumping or injection and slugging. Instantaneous head changes $s_0(p,1,k)$ may also be defined for $k > 1$.

It is justified to neglect wellbore storage if the well radius is small and no instantaneous head changes are realized within the well. In this case, it is convenient to set $S^s(p,1)$ equal to $S^S(p,2)$ and to let radius *r(1)* of the first nodal circle coincide with well radius r_w.

System of equations

When all parameters, initial conditions, and boundary conditions have been defined for each ring *(i,j)* and each time step *k*, a system of $n_z \times n_r \times n_{dt}$ finite-difference equations for the $n_z \times n_r \times n_{dt}$ unknown drawdowns *s(i,j,k+1)* is defined:

$$Q^r_{(i,j+1,k+1)} - Q^r_{(i,j,k+1)} + Q^z_{(i,j,k+1)} - Q^z_{(i+1,j,k+1)} - Q^s_{(i,j,k)} - Q_{(i,j,k)} = 0 \qquad [45]$$

The equations fulfill the law of conservation of mass and calculate the volumetric budget for each ring *(i,j)* at the end of time step *k*. In case of a steady state simulation, the Q^s term is zero, and if the model is one-dimensional, the Q^z terms are zero. The equations are considered for active, variable-head rings only: the equation for an inactive ring has all terms zero anyway, and in case of a constant-head ring, Q^s and *Q* are zero, but the sum of flow terms differs from zero, which means a constant-head ring acts as a source or sink.

The n_z x n_r equations [45] for time step *k* contain drawdowns only corresponding to the start and the end of the time step. Consequently, the whole system of equations can be solved sequentially for each time step *k* by rearranging terms to solve for the unknown drawdowns at the end of the time step. The drawdowns at the beginning of the time step are known and were calculated during the previous time step *k-1*. First, the set of equations is solved for the first time step, where the known drawdowns are the initial drawdowns *s(i,j,1)* defined by the modeler, and the unknown drawdowns are the drawdowns *s(i,j,2)* at the end of the time step. Then the set of equations for the second time step is solved to calculate the unknown drawdowns *s(i,j,3)* at the end of the time step, where the known drawdowns *s(i,j,2)* were calculated during the previous time step. This is repeated for all time steps, until all drawdowns *s(i,j,k+1)* are calculated at the end of the iterative process.

Afterwards, it is good practice to verify the volumetric budgets [45] for all rings *(i,j)* and all time steps *k*, which should be close to zero according to the law of mass conservation. The total volumetric budget for the entire model at the end of time step *k* is calculated by taking the sum of the volumetric budgets [45] for all variable-head rings during time step *k*. This sum should also be close to zero. MAxSym also calculates the volumetric budget for constant-head rings to allow calculation of the total amount of water added to or removed from the model by the constant-head rings. This amount of water may be verified against storage change or other sources or sinks.

If an initial drawdown $s_0(i,j,k)$ is defined for ring *(i,j)* at time *t(k)*, the volumetric budget for time step *k-1* is calculated using drawdown *s(i,j,k)* at the end of the time step, whereas the volumetric budget for time step *k* is calculated using drawdown $s(i,j,k) + s_0(i,j,k)$ at the start of the time step. During the iterative calculation process, drawdown *s(i,j,k)* is unknown during time step *k-1* and calculated at the end of this time step, while drawdown $s(i,j,k) + s_0(i,j,k)$ is the known drawdown at the start of time step *k*. In this way, the head change is instantaneous.

If the groundwater reservoir is assumed to have infinite lateral extent, it is also good practice to verify the boundary condition at infinity, which states drawdown at infinity is constant and zero throughout the simulation. Numerically, the infinite lateral extent is approximated by considering a large number of rings to let the no-flow conditions at the outer model boundary have an negligibly small influence on calculated drawdown. This is true if drawdown in the outer rings and radial discharges $Q^r(i,nr,k)$ are close to zero.

Once all drawdowns *s(i,j,k)* are calculated accurately, drawdown *s(i,r,t)* in layer *i* at any radial distance *r* and at any time *t* is computed using bilinear interpolation, which will be discussed in a next section. First, it is explained how the set of n_z x n_r equations is solved iteratively during time step *k*. Two solvers are provided with MAxSym: the iterative Alternating Direction Implicit method (ADI) and the Strongly Implicit Procedure (SIP).

Solvers

Rearranging terms, the finite difference equation for ring *(i,j)* and time step *k* is written as:

$$B_{(i,j)}s_{(i-1,j,k+1)} + D_{(i,j)}s_{(i,j-1,k+1)} + E_{(i,j,k)}s_{(i,j,k+1)} + F_{(i,j)}s_{(i,j+1,k+1)} + H_{(i,j)}s_{(i+1,j,k+1)} = g_{(i,j,k)}$$

[46]

where *B*, *D*, *E*, *F*, and *H* are known coefficients for the unknown drawdowns at the end of time step *k*, and where the right-hand side of equation [46] is the sum of all known terms, including the term containing the known drawdown *s(i,j,k)* from previous time step. Coefficients *B*, *D*, *F*, and *H* are time independent, because they only contain conductance terms related to the radial or vertical flow equations, whereas coefficient *E* and term *g* are time dependent:

$$\begin{aligned}
B_{(i,j)} &= Q^{zc}_{(i,j)}\\
D_{(i,j)} &= Q^{rc}_{(i,j)}\\
F_{(i,j)} &= Q^{rc}_{(i,j+1)}\\
H_{(i,j)} &= Q^{zc}_{(i+1,j)}\\
E_{(i,j,k)} &= -Q^{rzc}_{(i,j)} - Q^{ssc}_{(i,j)}/\Delta t_{(k)}\\
g_{(i,j,k)} &= Q_{(i,j,k)} - Q^{ssc}_{(i,j)}\left(s_{(i,j,k)} + s_{0(i,j,k)}\right)/\Delta t_{(k)}
\end{aligned}$$

[47]

The sum of conductances Q^{rzc} [L²/T] for ring *(i,j)* is defined as:

$$Q^{rzc}_{(i,j)} = Q^{rc}_{(i,j)} + Q^{rc}_{(i,j+1)} + Q^{zc}_{(i,j)} + Q^{zc}_{(i+1,j)} \qquad [48]$$

Recall that some coefficients for the top layer and the layer below must be updated during the calculation process if the top layer is phreatic:

$$\begin{aligned}
B^m_{(2,j,k)} &= \left(\frac{1}{Q^{zc}_{(2,j)}} + \frac{s^{m-1}_{(1,j,k+1)}}{2\pi\left(r^2_{b(j+1)} - r^2_{b(j)}\right)K^z_{(1,j)}}\right)^{-1}\\
D^m_{(1,j,k)} &= Q^{rc}_{(1,j)}\left(1 + \frac{s^{m-1}_{(1,j,k+1)} + s^{m-1}_{(1,j-1,k+1)}}{2D_{(1)}}\right)\\
F^m_{(1,j,k)} &= Q^{rc}_{(1,j+1)}\left(1 + \frac{s^{m-1}_{(1,j+1,k+1)} + s^{m-1}_{(1,j,k+1)}}{2D_{(1)}}\right)\\
H^m_{(1,j,k)} &= \left(\frac{1}{Q^{zc}_{(2,j)}} + \frac{s^{m-1}_{(1,j,k+1)}}{2\pi\left(r^2_{b(j+1)} - r^2_{b(j)}\right)K^z_{(1,j)}}\right)^{-1}\\
E^m_{(1,j,k)} &= -B^m_{(1,j,k)} - D^m_{(1,j,k)} - F^m_{(1,j,k)} - H^m_{(1,j,k)} - \left[Q^{ssc}_{(1,j)}\left(1 + \frac{s^{m-1}_{(1,j,k+1)}}{D_{(1)}}\right) + Q^{syc}_{(j)}\right]\Big/\Delta t_{(k)}\\
g^m_{(1,j,k)} &= Q_{(i,j,k)} - \left[Q^{ssc}_{(1,j)}\left(1 + \frac{s^{m-1}_{(1,j,k+1)}}{D_{(1)}}\right) + Q^{syc}_{(j)}\right]\left(s_{(i,j,k)} + s_{0(i,j,k)}\right)\Big/\Delta t_{(k)}
\end{aligned}$$

[49]

where index *m* refers to the iteration number for the given time step *k*.

The rearranged equations can be summarized in matrix form as:

$$A\bar{s} = \bar{g} \qquad [50]$$

where *A* is an *(n_z x n_r) x (n_z x n_r)* matrix containing the known coefficients, $\bar{s}$ is an *(n_z x n_r) x 1* column vector containing the unknown drawdowns, and $\bar{g}$ is an *(n_z x n_r) x 1* column vector containing the known right-hand side terms. In this form, the matrix system is layer-oriented. The ring-oriented formulation is obtained by swapping coefficients *B* and *D* and coefficients *F* and *H* in equation [46]:

$$B_{(i,j)}s_{(i,j-1,k+1)} + D_{(i,j)}s_{(i-1,j,k+1)} + E_{(i,j,k)}s_{(i,j,k+1)} + F_{(i,j)}s_{(i+1,j,k+1)} + H_{(i,j)}s_{(i,j+1,k+1)} = g_{(i,j,k)}$$

[51]

with:

$$\begin{aligned} B_{(i,j)} &= Q^{rc}_{(i,j)} \\ D_{(i,j)} &= Q^{zc}_{(i,j)} \\ F_{(i,j)} &= Q^{zc}_{(i+1,j)} \\ H_{(i,j)} &= Q^{rc}_{(i,j+1)} \\ E_{(i,j,k)} &= -Q^{rzc}_{(i,j)} - Q^{ssc}_{(i,j)}/\Delta t_{(k)} \\ g_{(i,j,k)} &= Q_{(i,j,k)} - Q^{ssc}_{(i,j)}\left(s_{(i,j,k)} + s_{0(i,j,k)}\right)/\Delta t_{(k)} \end{aligned}$$

[52]

In case of a phreatic top layer, some coefficients for the top layer and the layer below must be updated in the same way as for the layer-oriented formulation (see equations [49]).

For both formulations, coefficient matrix *A* is symmetric and sparse, where the nonzero elements are located on just five diagonals. The coefficients *D*, *E*, and *F* are always located on the three adjacent main diagonals, the coefficients *B* and *H* are located on 2 other diagonals, where the number of rows and columns respectively between those diagonals and the main diagonal depends on the formulation (Figure 2).

If the model is one-dimensional, it has one layer only with variable-head rings defined, and the layer-oriented formulation will produce a tridiagonal matrix *A*, since all coefficients *B* and *H* are zero. This is the case for one-layer models and models with one variable-head layer bounded above and/or below by a constant-head layer. The tridiagonal matrix system can be solved directly applying the Thomas algorithm if the system is linear. The latter condition is fulfilled if the variable-head layer has a constant saturated thickness. In case of an unconfined one-layer model, matrix *A* is also tridiagonal, but the system is not linear, because the coefficients contain unknown drawdown variables. Hence, the matrix system is solved iteratively. The matrix system is also solved iteratively when *A* has 5 nonzero diagonals. This is the case for all two-dimensional models having two or more layers with variable-head rings defined.

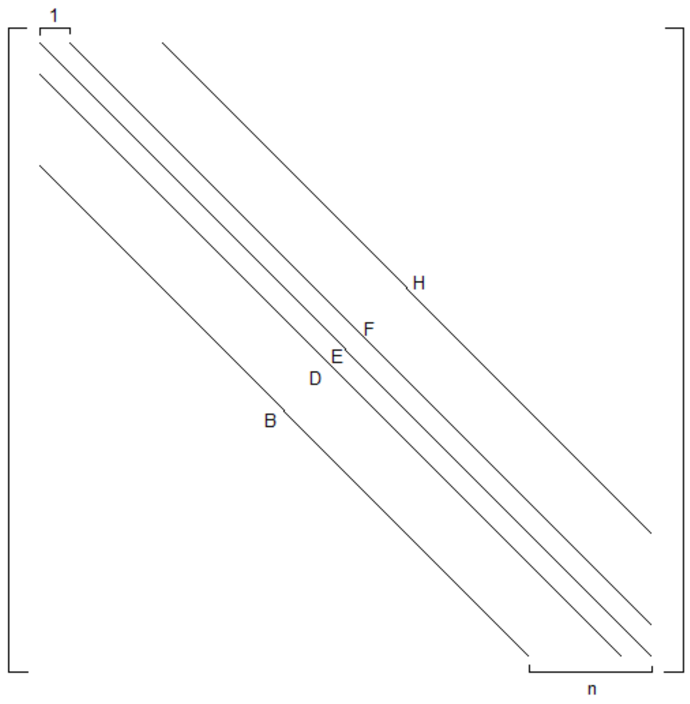


Figure 2. Structure of coefficient matrix A showing nonzero diagonals B, D, E, F, and H. Horizontal spacing is indicated between nonzero diagonals and expressed in matrix columns: 1 refers to adjacent diagonals, n = n_r if A is layer-oriented, and n = n_z if A is ring-oriented.

The two iterative solvers implemented with MAxSym are ADI and SIP. They do not yield an exact solution, but give an approximate solution that satisfies the following criterion for convergence [L]:

$$\left|s_{(i,j,k+1)}^{m} - s_{(i,j,k+1)}^{m-1}\right| \leq \delta \qquad \forall i,j \tag{53}$$

where index *m* refers to the iteration number for the given time step *k*. The value for *δ* is given by the modeler and must be sufficiently small to obtain accurate results. If a reasonable value for *δ* is chosen and the number of equations is not too large, convergence is reached relatively fast. Still, MAxSym requires the modeler to define a maximum number of iterations to prevent endless iterating of the solver in case of non-convergence. It is also good practice to verify the maximum number of iterations against the actual number of iterations the solver needed to solve the system. If both numbers are the same, the criterion for convergence was not satisfied and one should consider re-running the model defining a larger maximum number of iterations.

The general idea behind the presented solver algorithms is to decompose matrix *A* into a lower triangular matrix *L* and an upper triangular matrix *U*:

$$LU\bar{s} = \bar{g} \tag{54}$$

The lower triangular matrix *L* is a square matrix with all entries above the main diagonal zero, whereas upper triangular matrix *U* has all entries below the main diagonal zero. Substituting $U\bar{s}$ for column vector $\bar{v}$, the following system is obtained:

$$\begin{cases} L\bar{v} = \bar{g} \\ U\bar{s} = \bar{v} \end{cases} \tag{55}$$

which is easily solved by applying the technique of forward and backward substitution. The forward substitution is performed recursively:

$$\begin{cases} v_1 = g_1/l_{11} \\ v_2 = (g_2 - l_{21}v_1)/l_{22} \\ v_3 = (g_3 - l_{31}v_1 - l_{32}v_2)/l_{33} \\ \qquad\qquad \vdots \end{cases}$$

[56]

where l_{ij} is the entry on row *i* and column *j* in matrix *L*. The backward substitution is also performed recursively:

$$\begin{cases} s_p = v_p/u_{pp} \\ s_{p-1} = \left(v_{p-1} - u_{p-1,p}s_p\right)/u_{p-1,p-1} \\ s_{p-2} = \left(v_{p-1} - u_{p-2,p}s_p - u_{p-2,p-1}s_{p-1}\right)/u_{p-2,p-2} \\ \qquad\qquad \vdots \end{cases}$$

[57]

where $p = n_z \times n_r$ and u_{ij} is the entry on row *i* and column *j* in matrix *U*. Note that the linear index of drawdown *s* depends on the formulation of matrix *A*, and it corresponds to ring *(i,j)* as follows:

$$\begin{cases} s_{i_l} = s_{(i,j)} \Leftrightarrow i_l = (i-1)n_r + j \quad (layer - oriented) \\ s_{i_r} = s_{(i,j)} \Leftrightarrow i_r = (j-1)n_l + i \quad (row - oriented) \end{cases}$$

[58]

Because drawdowns are represented by a column vector, the linear indices refer to the rows of matrix *A*. Hence, coefficients B_i, D_i, E_i, F_i, and H_i corresponding to drawdown s_i are located on the *i*-th row of matrix *A*.

The decomposition could be performed by applying the standard LU decomposition algorithm, which gives a result for any invertible matrix. However, this algorithm disregards the fact matrix *A* is sparse, whereas the presented algorithms take advantage of the sparseness of *A*, and consequently, solve the matrix system faster, more efficiently, and in most cases more accurately.

Thomas algorithm

Performing elementary row operations on tridiagonal matrix *A*, it can be shown upper triangular matrix *U* is defined recursively as (see, for instance, Wang and Anderson, 1982):

$$\begin{cases} u_{ii} = 1 \\ u_{i,i+1} = F_i/(E_i - D_i u_{i-1,i}) \\ u_{i,i+j} = 0 \qquad\qquad j > 1 \end{cases}$$

[59]

where $u_{12} = F_1/E_1$, and *i* is the linear index corresponding to ring *(i,j)* as defined above. So, all entries along the main diagonal of *U* are 1, the entries along the diagonal just above the main diagonal are nonzero, and all other elements are zero. The corresponding vector $\bar{v}$ is also defined recursively:

$$v_i = (g_i - D_i v_{i-1})/(E_i - D_i u_{i-1,i}) \quad [60]$$

starting with $v_1 = g_1/E_1$. After U and $\bar{v}$ are obtained through forward substitution, backward substitution is applied to calculate drawdown vector $\bar{s}$ recursively as:

$$s_i = v_i - u_{i,i+1} s_{i+1} \quad [61]$$

starting with $s_p = v_p$ where $p = n_z \times n_r$.

Not only this algorithm is a fast way to solve one-dimensional systems directly, it is also the basis for the iterative ADI method.

Iterative Alternating Direction Implicit method

The iterative ADI method is explained in, for instance, Wang and Anderson (1982) and Lebbe (1999).

If it is assumed drawdown corresponding to coefficients *B* and *H* is known, a tridiagonal matrix system is obtained:

$$D_i s_{i-1}^m + E_i s_i^m + F_i s_{i+1}^m = g_i - B_i s_{i-n}^{m-1} - H_i s_{i+n}^{m-1} = g_i' \quad [62]$$

where index *i* is the linear index referring to ring *(i,j)*, variable *n* is n_l or n_r depending on the matrix orientation (see Figure 2), and index *m* indicates the iteration number. Hence, drawdown corresponding to coefficients *B* and *H* is adopted from the previous iteration; during the first iteration, drawdown from the previous time step is taken.

The Thomas algorithm is applied to solve the system for each iteration and convergence is obtained by alternating the direction: the first iteration solves the layer-oriented formulation of the matrix system, the second iteration solves the ring-oriented formulation, and so on, until the criterion for convergence is satisfied or the maximum number of iterations is reached. As already mentioned before, some of the coefficients *B*, *D*, *E*, *F*, and *H* are updated at the beginning of each iteration if the top layer is unconfined, and initial drawdowns must be taken into account at the beginning of the first iteration.

The iterative ADI method is efficient when applied to relatively small models simulating confined flow and having few sources and sinks defined. Running more complex models, however, it is recommended to apply the SIP method, especially when areal infiltration is defined or constant-head rings are used.

Strongly Implicit Procedure

The SIP method is developed by Stone (1968). The MAxSym implementation is adopted mainly from McDonald and Harbaugh (1988).

The SIP method seeks to find a matrix *Z* such that *A+Z* can be factored easily into matrices *L* and *U*, where *A+Z* is close to *A*, both *L* and *U* have just four nonzero diagonals, and all entries along the main diagonal of *U* are 1. The new system of equations is written in matrix form as:

$$(A + Z)\bar{s} = \bar{g} + Z\bar{s} \quad [63]$$

The iterative approach allows to replace the drawdown vector on the right-hand side by the drawdown vector from the preceding iteration. Substituting $\bar{s}^{m-1}$ into the right-hand side and subtracting $(A+Z)\bar{s}^{m-1}$ from each side of the equation yields the system of equations to solve using LU factorization:

$$(A+Z)(\bar{s}^m - \bar{s}^{m-1}) = \bar{g} - A\bar{s}^{m-1} \quad \Leftrightarrow \quad LU(\bar{s}^m - \bar{s}^{m-1}) = \bar{R} \qquad [64]$$

where vector $\bar{R}$ is the right-hand side of the equation equal to $\bar{g} - A\bar{s}^{m-1}$. In order to meet the conditions specified for *A+Z*, *L*, and *U*, matrix *A+Z* must contain two additional nonzero diagonals *C* and *G*, as shown in Figure 3. The required structure of lower and upper triangular matrices *L* and *U* is shown in Figure 4.

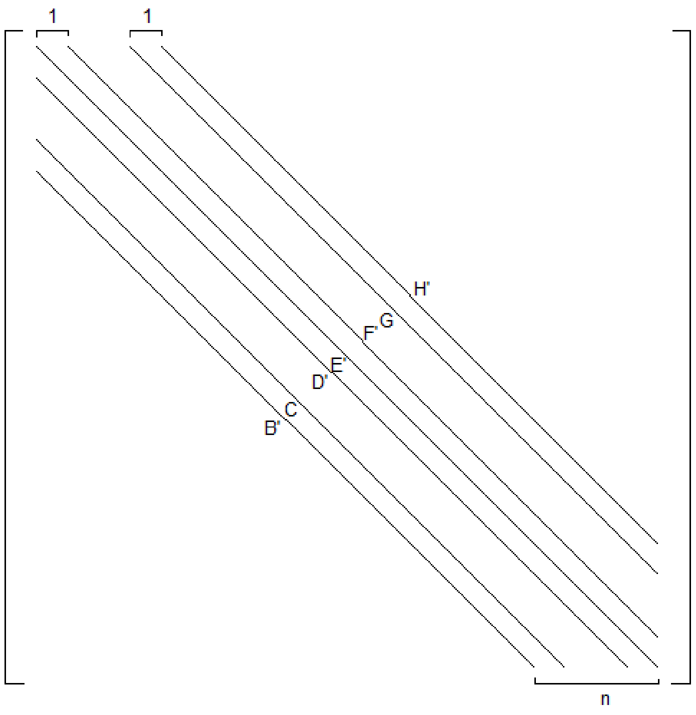


Figure 3. Structure of modified coefficient matrix A+Z showing nonzero diagonals. Horizontal spacing is indicated between nonzero diagonals and expressed in matrix columns: 1 refers to adjacent diagonals, n = n_r if A is layer-oriented, and n = n_z if A is ring-oriented. Matrix A+Z has nonzero diagonal terms on the same positions as A, which are indicated by the same symbols, and two additional nonzero diagonals C and G. Note that elements along the corresponding diagonals have been modified, which is indicated by a quote: B', D', E', F', and H'.

The following recursive expressions are used to factorize *A+Z* into *L* and *U*, starting with *i = 1*:

$$\begin{cases} b_i = B_i/(1+\omega e_{i-n}) \\ c_i = D_i/(1+\omega f_{i-1}) \\ C_i = e_{i-n} b_i \\ G_i = f_{i-1} c_i \\ d_i = E_i + \omega(C_i + G_i) - b_i f_{i-n} - c_i e_{i-1} \\ e_i = (F_i - \omega C_i)/d_i \\ f_i = (H_i - \omega G_i)/d_i \end{cases} \qquad [65]$$

where *B*, *D*, *E*, *F*, and *H* are elements from the original coefficient matrix *A*; *C* and *G* are elements belonging to the additional nonzero diagonals of matrix *A+Z*; *b*, *c*, and *d* are elements from matrix *L*; *e* and *f* belong to matrix *U*. Recall *n* depends on the orientation of *A* (see Figure 2). Multiplier *ω* is a damping factor between 0 and 1, and will be discussed further below.

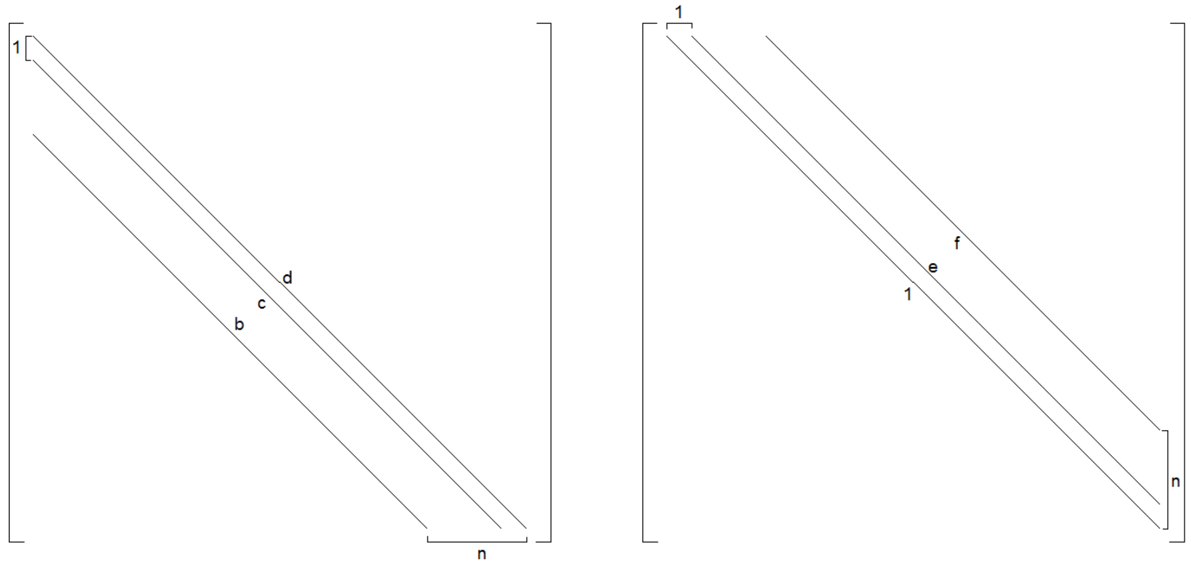


Figure 4. Structure of triangular matrix L (left) and U (right) showing nonzero diagonals. Spacing between nonzero diagonals is expressed in matrix columns or rows: 1 refers to adjacent diagonals, n = n_r if A is layer-oriented, and n = n_l if A is ring-oriented. The nonzero diagonals of L are b, c, and d. The main diagonal of U only contains elements equal to 1, the two other nonzero diagonals are e and f.

Once matrix *A+Z* has been decomposed into triangular matrices *L* and *U*, it is straightforward to perform forward and backward substitution. The forward substitution starts with *i = 1* and is expressed as:

$$v_i = (\alpha R_i - b_i v_{i-n} - c_i v_{i-1})/d_i \qquad [66]$$

where accelerator *α* is an additional relaxation parameter between 0 and 1. Mostly, *α* is assigned a value of 1, but a smaller value can be assigned if problems with convergence are encountered. Term R_i is the *i*-th element of the right-hand side vector $\bar{R}$.

The backward substitution starts with $i = n_z \times n_r$ and yields a value for all unknown drawdowns:

$$s_i^m = s_i^{m-1} + v_i - e_i v_{i+1} - f_i v_{i+n} \qquad [67]$$

Apparently, the number of iterations needed to converge to a solution is reduced if the equations are solved in two different orders on alternate iterations. Therefore, the alternating direction technique is adopted from the ADI method: the first iteration solves the layer-oriented formulation of the matrix system, the second iteration solves the ring-oriented formulation, and so on, until the criterion for convergence is satisfied or the maximum number of iterations is reached. Recall that some of the coefficients *B*, *D*, *E*, *F*, and *H* are updated at the beginning of each iteration if the top layer is unconfined, and initial drawdowns must be taken into account at the beginning of the first iteration.

Relaxation parameter *ω* is also updated at the beginning of each iteration according to:

$$\omega_j = 1 - \sigma^{(j-1)/(n_{parm}-1)} \qquad j = 1,2,\dots,n_{parm} \qquad [68]$$

where index *j* is 1 at the start of the first iteration, 2 at the start of the second iteration, and so on, until *j* is n_{parm}, after which *j* is reset to 1. The total number of iteration parameters n_{parm} is given by the modeler. In most cases, 5 iteration parameters are sufficient. Variable *σ* is the so called iteration parameter seed and it may be specified by the modeler. If not, a value for *σ* is derived from the flow conductances by determining the following terms ρ_1 and ρ_2 for each ring *(i,j)*:

$$\rho_{1(i,j)} = \frac{max(Q^{zc}_{(i,j)}, Q^{zc}_{(i+1,j)})}{min(Q^{rc}_{(i,j)}, Q^{rc}_{(i,j+1)})}$$

$$\rho_{2(i,j)} = \frac{max(Q^{rc}_{(i,j)}, Q^{rc}_{(i,j+1)})}{min(Q^{zc}_{(i,j)}, Q^{zc}_{(i+1,j)})}$$

[69]

where functions *max(x₁,x₂)* and *min(x₁,x₂)* return respectively maximum and minimum of x_1 and x_2. The seed *σ* is computed as:

$$\sigma = \frac{1}{n_r n_z}\sum_{i,j} min\left(\frac{\pi^2}{2n_r^2(1+\rho_{1(i,j)})}, \frac{\pi^2}{2n_Z^2(1+\rho_{2(i,j)})}\right)$$

[70]

The above procedure to derive a iteration parameter seed is adopted from McDonald and Harbaugh (1988). There is, however, little understanding of why some sequence of iteration parameters ω_j performs better than another, because the process is essentially empirical. Anyway, the final solution should not be influenced by the choice of relaxation parameters as they are intended only to fasten the rate of convergence.

In the author's experience, the SIP method is generally faster and more efficient than the ADI method. Running complex models considering many sources and sinks, the SIP method still yields accurate results, whereas the ADI method often fails to reach convergence or to produce accurate results. Therefore, it is recommended to use SIP as default solver.

Interpolation

Drawdown *s(i,r,t)* in layer *i* at any radial distance *r* and at any time *t* is computed using bilinear interpolation. Drawdown is assumed to vary linearly as a function of *log(r)* between two nodal circles *j* and *j+1* according to the Thiem (1906) equation:

$$s_{(i,r,k)} = s_{(i,j,k)} + \frac{s_{(i,j+1,k)} - s_{(i,j,k)}}{log(r_{(j+1)}/r_{(j)})} log(r/r_{(j)}) \qquad r_{(j)} \le r \le r_{(j+1)}$$

[71]

where *s(i,r,k)* is drawdown in layer *i* at time *t(k)* at an arbitrary distance *r* between nodal circles *j* and *j+1*. Drawdown is also assumed to vary linearly as a function of *log(t)* between two times *t(k)* and *t(k+1)* according to the Cooper-Jacob approximation (Cooper and Jacob, 1946):

$$s_{(i,j,t)} = s_{(i,j,k)} + \frac{s_{(i,j,k+1)} - s_{(i,j,k)}}{log(t_{(k+1)}/t_{(k)})} log(t/t_{(k)}) \qquad t_{(k)} \leq t \leq t_{(k+1)}$$

[72]

where *s(i,j,t)* is drawdown in layer *i* at distance *r(j)* at an arbitrary time *t* within time step *k*. Within layer *i*, drawdown is assumed not to vary in the vertical direction according to the Dupuit (1863) approximation.

To calculate drawdown *s(i,r,t)* in layer *i* at any radial distance *r* and at any time *t*, bilinear interpolation is performed in two steps: first, *s(i,r,k)* and *s(i,r,k+1)* are determined applying linear interpolation in the dimension of *r*, after which *s(i,r,t)* is derived from *s(i,r,k)* and *s(i,r,k+1)* applying linear interpolation in the dimension of *t*. With MATLAB, function `interp1` is provided to perform linear interpolation in one dimension, and function `interp2` is available for bilinear interpolation.

Care should be taken when determining drawdown within the distance of the well radius. Since it is assumed drawdown within the well is equal to drawdown at the well face, interpolation is not allowed, and drawdown at an arbitrary distance within the well radius is defined as:

$$s_{(i,r,k)} = s_{(i,1,k)} \qquad r \leq r_{b(2)}$$ [73]

where the outer radius r_b*(2)* of the first ring is equal to the well radius r_w. If wellbore storage is neglected and the well radius corresponds to radius *r(1)* of the first nodal circle, the following holds:

$$s_{(i,r,k)} = s_{(i,1,k)} \qquad r \leq r_{(1)}$$ [74]

Constant-head rings are no obstacle when interpolation is applied, and neither are inactive rings, because drawdown corresponding to the latter is assigned a `NaN` value. However, the modeler should be aware of the extent of constant-head and inactive rings. When applying the above equation to interpolate drawdown in dimension *r*, constant heads will occur only at the radial distances at which they are defined, whereas inactive rings will extend to the adjacent nodal circles.

When instantaneous head changes are defined, they must be subtracted from the corresponding drawdown if interpolated drawdown is within the preceding time step. Consider an initial drawdown s_0*(i,j,k)* defined at the start of time step *k > 1*, drawdown at an arbitrary time *t* between *t(k-1)* and *t(k)* is interpolated as:

$$s_{(i,j,t)} = s_{(i,j,k-1)} + \frac{s_{(i,j,k)} - s_{0(i,j,k)} - s_{(i,j,k-1)}}{log(t_{(k+1)}/t_{(k)})} log(t/t_{(k)}) \qquad t_{(k-1)} \leq t \leq t_{(k)}$$

[75]

whereas drawdown at an arbitrary time *t* between *t(k)* and *t(k+1)* is interpolated as explained above.

Summary

MAxSym solves the partial differential equation that describes transient, two-dimensional, non-uniform, axi-symmetric groundwater flow in saturated, porous media by applying the finite-difference method. Steady-state flow and one-dimensional flow in a single layer are treated as special cases of the generic transient, two-dimensional case. The partial differential equation is linear, which means it assumed the groundwater reservoir is confined or has constant-head boundaries. However, unconfined conditions may also be considered, in which case the effect of a water table is accounted for by defining a specific yield parameter.

The finite-difference method requires a discretization of space and time dimensions. Therefore, an axi-symmetric grid consisting of layers and rings is constructed, and time steps are specified, which are grouped into stress periods to facilitate the input of time-dependent stresses. The grid boundaries are defined using constant-head and inactive rings. After the model discretization is performed, hydraulic parameters are assigned to each ring in the grid. Initial and time-dependent boundary conditions are stated by specifying initial drawdown and discharge for each ring and for each stress period. Stresses are used to implement head and discharge specified wells and areal infiltration.

The model input is used to formulate a system of finite-difference equations applying Darcy's law and the continuity principle. If the system is linear and one-dimensional, it can be solved directly using the Thomas algorithm. Otherwise, the system must be solved iteratively using one of the two solvers provided with MAxSym: the iterative Alternating Direction Implicit method (ADI) or the Strongly Implicit Procedure (SIP). It is recommended to apply the SIP method to solve large models containing many sources and sinks. After each model run, it is good practice to verify the number of iterations the solver needed to solve the system. Additionally, the volumetric budget for each cell and for the entire model must be verified.

MAxSym only calculates drawdown at the nodal circles of the rings in the axi-symmetric grid and at the end of each time step. Bilinear interpolation is applied to calculate drawdown at an arbitrary distance and at an arbitrary time. This is required, for instance, when calculated drawdown is used to construct simple, three-dimensional models applying the principle of superposition, or to analyze aquifer tests applying an inverse model. Examples of which are given in a next chapter presenting many Modeling examples.

Modelers' guide

The steps involved in building and running a MAxSym model are:

1. defining stress periods and time steps;
2. defining the axi-symmetric grid;
3. defining parameters;
4. defining stresses;
5. defining the solver;
6. running the model.

These steps are reflected in MAxSym's object-oriented structure, which is presented in Figure 5. The `MAxSym` package contains 6 classes, from which class `Model` is the central class or mediator. The other 5 classes are connected to class `Model` through a composition relationship, which means that they are part of class `Model`. The multiplicity indicates the number of class instances that participate in the composition: one `Model` object contains one `Grid` object, one `TimeSteps` object, one `Parameters` object, one `Solver` object and at least one `Stresses` object. Note that all MAxSym classes are `handle` classes, which means that MATLAB does not make a copy of the original object when it is assigned to multiple variables or passed to functions. See MATLAB Help to get more information about `handle` classes and to know the differences from value classes.

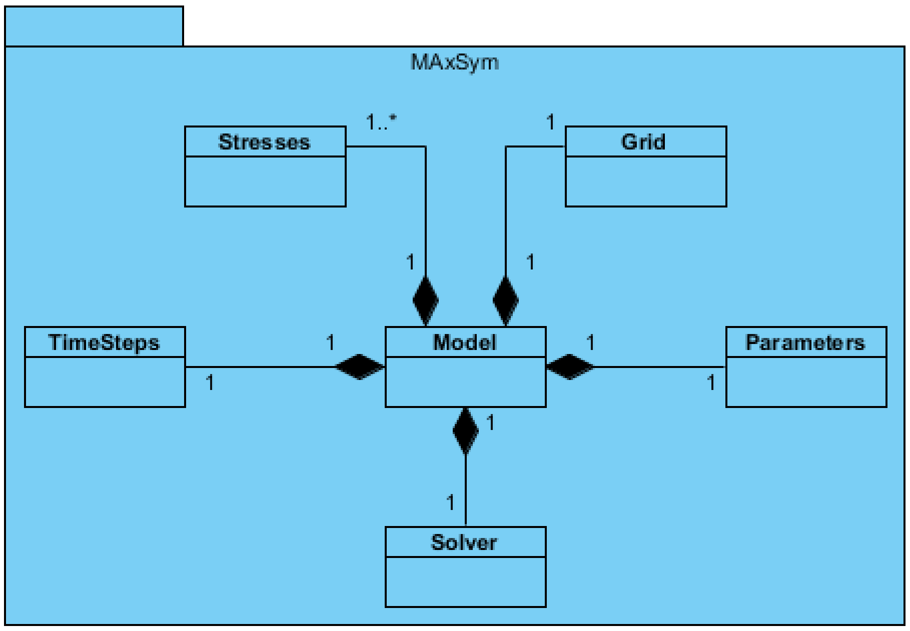


Figure 5. UML class diagram presenting MAxSym's object-oriented structure.

Figure 6 is a sequence diagram showing the interactions between modeler, class `Model` and the other MAxSym classes during model setup. Since class `Model` contains all other classes, the model setup always starts with creating a `Model` object by calling the class constructor. Then the other MAxSym classes are instantiated one by one following the steps outlined above:

1. the modeler creates a `TimeSteps` object by calling the `settime` method of the `Model` object;

2. the modeler creates a `Grid` object by calling the `setgrid` method of the `Model` object;
3. the `Model` object creates a `Parameters` object to allow the modeler to set the hydraulic parameters in the `Parameters` object;
4. the `Model` object creates the required number of `Stresses` objects to allow the modeler to set the stresses in the `Stresses` objects;
5. the modeler creates a `Solver` object by calling the `setsolver` method of the `Model` object;
6. the modeler runs the model by calling the `run` method of the `Model` object.

Note that the order in which the steps are performed is not as strict as indicated here, and only three steps require precedent steps to be fulfilled: steps 3 and 4 can only take place after steps 1 and 2 were accomplished, and step 6 can only be done if the previous 5 steps were performed.

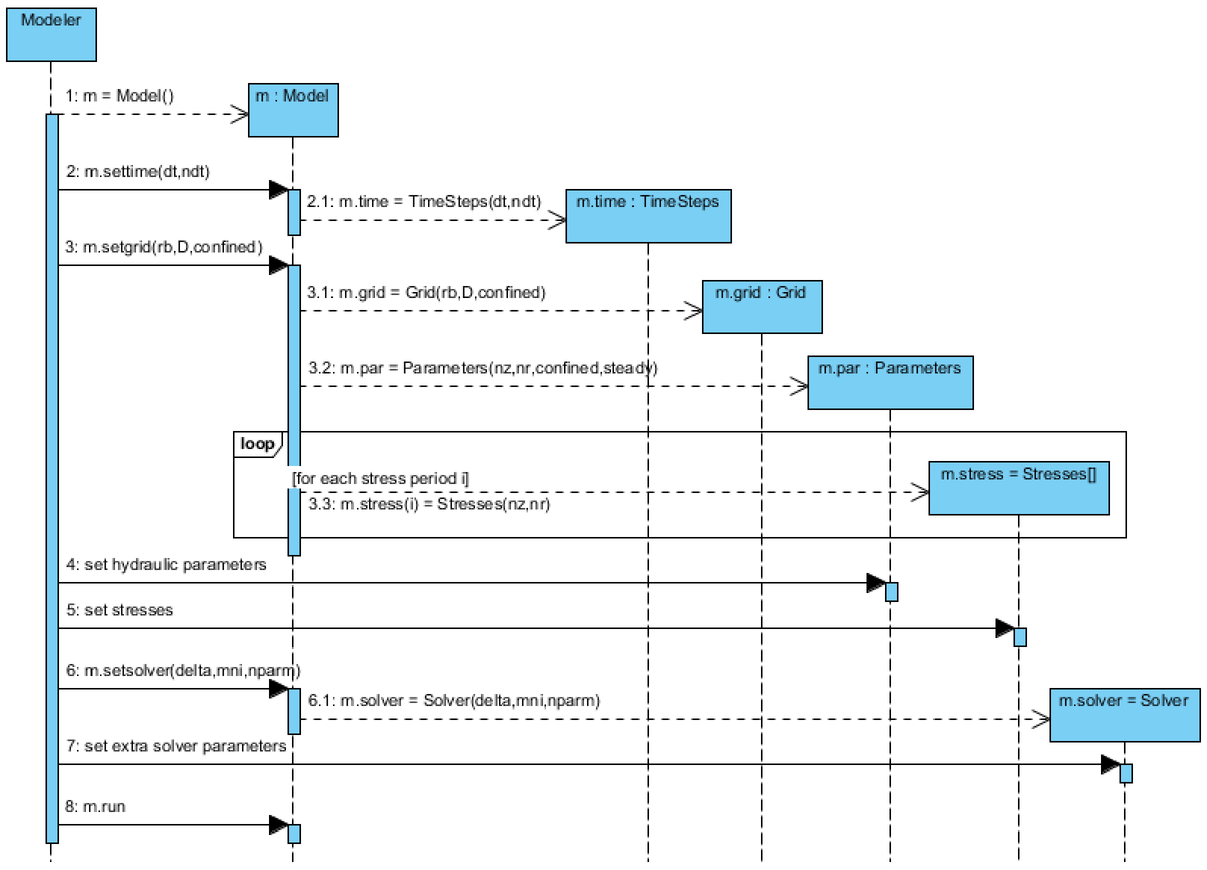

Modeler
1: m = Model()
m : Model
2: m.settime(dt,ndt)
2.1: m.time = TimeSteps(dt,ndt)
m.time : TimeSteps
3: m.setgrid(rb,D,confined)
3.1: m.grid = Grid(rb,D,confined)
m.grid : Grid
3.2: m.par = Parameters(nz,nr,confined,steady)
m.par : Parameters
loop
[for each stress period i]
3.3: m.stress(i) = Stresses(nz,nr)
m.stress = Stresses[]
4: set hydraulic parameters
5: set stresses
6: m.setsolver(delta,mni,nparm)
6.1: m.solver = Solver(delta,mni,nparm)
m.solver = Solver
7: set extra solver parameters
8: m.run

Figure 6. UML sequence diagram showing the interactions between modeler and MAxSym classes during model setup.

The following paragraphs will discuss in more detail the six steps listed above by giving an overview of the public properties and methods of the MAxSym classes. Note that MAxSym source files are well documented, and hence, it is possible to use the standard help functions `lookfor` and `help` provided with MATLAB.

Figure 6 indicates the modeler interacts with the `Model` object only, so the most efficient way to get help on using MAxSym is to invoke the `help` function for class `Model`:

```
>> help MAxSym.Model
  class to build and run axi-symmetric model MAxSym

  1.  create MAxSym.Model object: m = MAxSym.Model;
  2.  define stress periods and time steps: m.settime(dt,ndt);
  3.  define model grid: m.setgrid(rb,D,confined);
  4.  define hydraulic parameters:
      - radial conductivity: m.par.kr = ...;
      - radial resistance: m.par.cr = ...;
      - vertical conductivity: m.par.kz = ...;
      - vertical resistance: m.par.cz = ...;
      - specific elastic storage coefficient: m.par.ss = ...;
      - specific yield: m.par.sy = ...;
  5.  define inactive and/or constant head rings if any:
      - inactive rings: m.par.inactive = ...;
      - constant head rings: m.par.constant = ...;
  6.  define stresses for each stress period k:
      - discharge: m.stress(k).q = ...;
      - initial head change: m.stress(k).s0 = ...;
  7.  define solver: m.setsolver(delta,mni,nparm);
  8.  run model: m.run;
  9.  check flow and volumetric budget terms:
      - radial flow terms: m.qr
      - vertical flow terms: m.qz
      - storage change terms: m.qs
      - volumetric budget terms: m.bud
      - total volumetric budget: m.totbud
  10. get drawdown matrix: m.s
  11. apply linear interpolation to get drawdown at arbitrary distances and times:
      - linear interpolation in dimension of log(r): s = m.interp1r(ilay,r,it,rw);
      - linear interpolation in dimension of log(t): s = m.interp1t(ilay,ir,t);
      - bilinear interpolation in dimensions of log(r) and log(t): s = m.interp2(ilay,r,t,rw);
```

A detailed overview of the six steps discussed above is displayed, which will become clear at the end of this chapter. Properties and methods are hyperlinked and clicking on a hyperlink displays the corresponding help information. For instance, clicking on the hyperlink for method `interp1t` displays a full description of its signature:

```
  interpolate drawdown in dimension of log(t)

  syntax: s = obj.interp1t(ilay,ir,t)
   ilay is vector with indices of considered layers
    if ilay is empty, all layers are considered or ilay = 1:obj.grid.nz
   ir is vector with indices of considered rings
    if ir is empty, all rings are considered or ir = 1:obj.grid.nr
   t is vector with times at which drawdown is interpolated
    times must be between zero and obj.time.t(end)
    for times t smaller than obj.time.t(2), drawdown s is not interpolated,
     but equal to initial drawdown or: if t < obj.time.t(2), then s = obj.stress(1).s0
   s is the interpolated drawdown array
    size(s) = [length(ilay), length(ir), length(t)]

   remark: method interp1t can only be used if the model is transient
```

Note that `obj` refers to the `Model` object. The same information is displayed when the following command is evaluated:

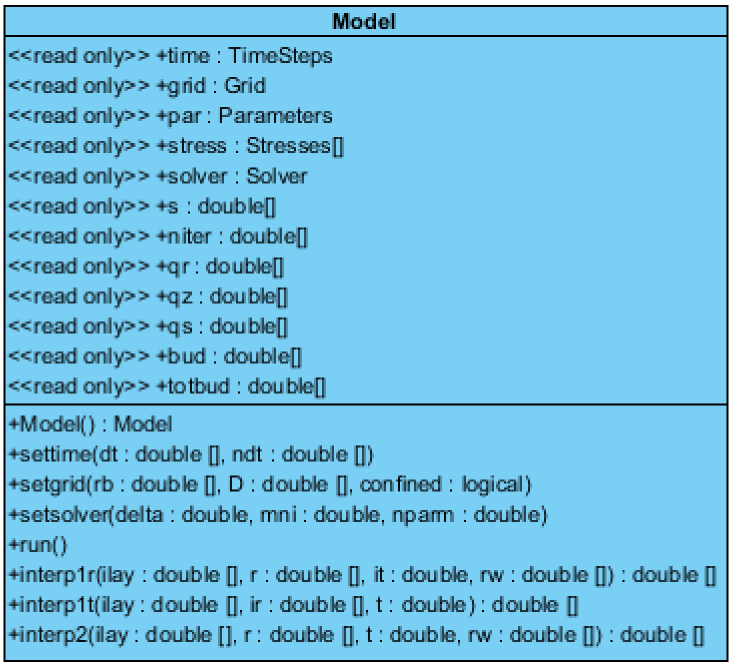
```
>> help MAxSym.Model.interp1t
```

All other MAxSym methods and properties are provided with a similar description.

Class Model

Figure 7 indicates the public methods and properties for class `Model`. All indicated properties are read only, meaning that they cannot be modified. The input objects of which a `Model` object is composed, correspond to properties `time`, `grid`, `par`, `stress`, and `solver`. These properties will be discussed in the next paragraphs. The other properties are model output: property `s` contains the drawdown matrix, property `niter` indicates the number of iterations, and properties `qr`, `qz`, `qs`, `bud`, and `totbud` are flow and budget terms. These properties will be discussed in the paragraph explaining how to run a MAxSym model.

Some of the presented methods have already been mentioned: constructor `Model` is called to create a `Model` object, `settime` is called to define time steps, `setgrid` is called to define the model grid, `setsolver` is called to define the solver, and `run` is called to run the model. Additionally, class `Model` is also provided with three interpolation methods: `interp1r`, `interp1t`, and `interp2`. These methods will also be discussed in more detail in the next paragraphs.



```
Model
<<read only>> +time : TimeSteps
<<read only>> +grid : Grid
<<read only>> +par : Parameters
<<read only>> +stress : Stresses[]
<<read only>> +solver : Solver
<<read only>> +s : double[]
<<read only>> +niter : double[]
<<read only>> +qr : double[]
<<read only>> +qz : double[]
<<read only>> +qs : double[]
<<read only>> +bud : double[]
<<read only>> +totbud : double[]
+Model() : Model
+settime(dt : double [], ndt : double [])
+setgrid(rb : double [], D : double [], confined : logical)
+setsolver(delta : double, mni : double, nparm : double)
+run()
+interp1r(ilay : double [], r : double [], it : double, rw : double []) : double []
+interp1t(ilay : double [], ir : double [], t : double) : double []
+interp2(ilay : double [], r : double [], t : double, rw : double []) : double []
```

Figure 7. UML class diagram presenting public properties and methods for class `Model`. Data types followed by `[]` indicate arrays, otherwise a scalar value is required.

The model setup starts with creating a `Model` object. This is done by invoking the `Model` class constructor, which returns a new `Model` object:

```
>> m = MAxSym.Model;
```

The returned object is also a `handle`, which means a reference to the object created is stored in variable `m`. Other variable names may be chosen, as long as they are valid MATLAB names. Note that the constructor name must be prefixed by the package name `MAxSym` if the package has not been imported (See MATLAB function `import`).

Class TimeSteps

The simulation time is discretized into time steps. When transient stresses are considered, the time steps are grouped into stress periods, and stresses can only change at the beginning of a stress period.

Figure 8 indicates the public methods and properties for class `TimeSteps`. All indicated properties are read only, meaning that they cannot be modified. The only public method is the `TimeSteps` constructor, which is invoked when the modeler calls the `settime` method of the `Model` object. Both methods require the same input arguments: the first argument is a `double` vector with time steps, the second argument indicates the number of time steps in each stress period and is required only if the model has more than one stress period.

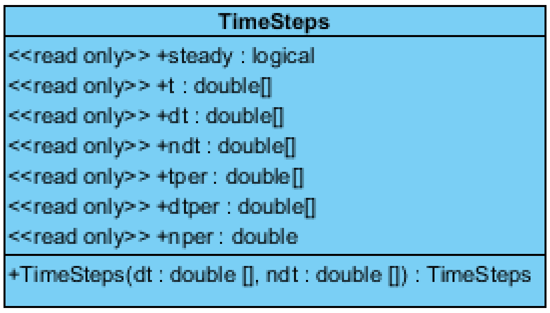

TimeSteps
<<read only>> +steady : logical
<<read only>> +t : double[]
<<read only>> +dt : double[]
<<read only>> +ndt : double[]
<<read only>> +tper : double[]
<<read only>> +dtper : double[]
<<read only>> +nper : double
+TimeSteps(dt : double [], ndt : double []) : TimeSteps

Figure 8. UML class diagram presenting public properties and methods for class `TimeSteps`. Data types followed by `[]` indicate arrays, otherwise a scalar value is required.

To create a `TimeSteps` object, a distinction must be made between steady and unsteady state simulations.

The following statement creates a `TimeSteps` object when the simulation is **steady state**:

```
>> m.settime(0);
```

The created object is stored in property `time` of the `Model` object:

```
>> m.time
```

In case of a steady state simulation, property `steady` of the `TimeStep` object is set `true`. The contents of the other properties is irrelevant, except for property `nper`, which must be `1`, since it indicates the number of stress periods, and hence, the number of `Stresses` objects.

When the simulation is **transient**, time steps need to be defined first. If time is discretized in logarithmic space, then the MATLAB function `logspace` may be used:

```
>> p1 = logspace(log10(t1),log10(t2),n1+1);
```

where `p1` is a vector with simulation times for stress period 1, `t1` is the starting time of period 1, `t2` is the ending time of period 1, and `n1` is the number of time steps in period 1. The subsequent stress periods are defined similarly:

```
>> p2 = logspace(log10(t2),log10(t3),n2+1);
>> p3 = logspace(log10(t3),log10(t4),n3+1);

>> ...;
```

where `p2` is a vector with simulation times for stress period 2, `t2` is the starting time of period 2, `t3` is the ending time of period 2, `n2` is the number of time steps in period 2, etc. Once all simulation times are determined for each stress period, time steps `dt` are calculated as follow:

```
>> t = [p1, p2(2:end), p3(2:end), ...];

>> dt = diff(t);
```

Finally, the following statement creates the `TimeSteps` object:

```
>> m.settime(dt,[n1,n2,n3,…]);
```

If only one time step is considered, the second input argument is not required. The properties of the created `TimeSteps` object can be verified by getting the `time` property of the `Model` object. Table 1 gives an overview of these properties.

Table 1. Read only properties for the `TimeSteps` object in case of a transient simulation.

Property	**Description**	**Contents**
`m.time.steady`	simulation is steady state?	`false`
`m.time.t`	simulation times [T]	`t - t1`
`m.time.dt`	time steps [T]	`dt`
`m.time.ndt`	number of time steps for each stress period	`[n1,n2,n3,…]`
`m.time.tper`	stress period times [T]	`[t1,t2,t3,t4,…] - t1`
`m.time.dtper`	stress period lengths [T]	`diff([t1,t2,t3,t4,…])`
`m.time.nper`	number of stress periods	`length([n1,n2,n3,…])`

Note that simulation times and stress period times are rescaled to zero and hence, they must be augmented by `t1` when used to process calculated drawdowns as a function of time. The time unit [T] may be chosen arbitrarily, but must be maintained when defining other time dependent parameters such as discharges and conductivities.

Class Grid

The aquifer is discretized into rings (radially) and layers (vertically). The former are characterized by the radii of inner and outer ring boundaries, the latter by the layer thicknesses. Rings are numbered from inner to outer model boundary, layers from top to bottom.

Figure 9 indicates the public methods and properties for class `Grid`. All indicated properties are read only, meaning that they cannot be modified. The only public method is the `Grid` constructor, which is invoked when the modeler calls the `setgrid` method of the `Model` object. Both methods require the same input arguments: the first argument is a `double` vector with the radii of the ring boundaries, the second argument is a vector with the layer thicknesses, the third argument is a `logical` indicating whether the aquifer system is confined or unconfined.

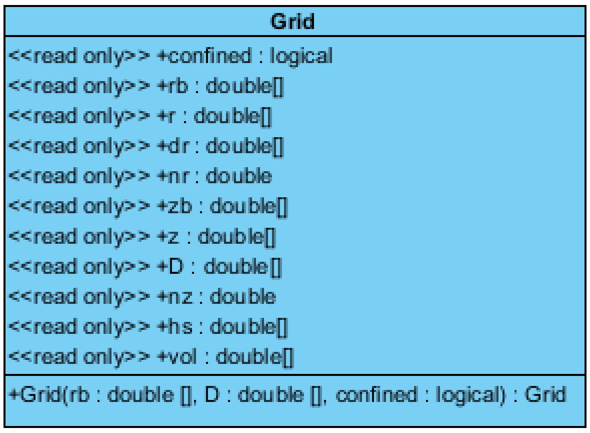

Grid
<<read only>> +confined : logical
<<read only>> +rb : double[]
<<read only>> +r : double[]
<<read only>> +dr : double[]
<<read only>> +nr : double
<<read only>> +zb : double[]
<<read only>> +z : double[]
<<read only>> +D : double[]
<<read only>> +nz : double
<<read only>> +hs : double[]
<<read only>> +vol : double[]
+Grid(rb : double [], D : double [], confined : logical) : Grid

Figure 9. UML class diagram presenting public properties and methods for class `Grid`. Data types followed by `[]` indicate arrays, otherwise a scalar value is required.

The first argument is obtained by discretizing radial distance in logarithmic space using the function `logspace`:

```
>> rb = logspace(log10(rw),log10(rout),nr);
```

where `rb` is a vector with the radii of the ring boundaries, `rw` is the well radius, `rout` is the outer grid radius, and `nr` is the number of rings. To obtain the first ring that represents the well, we need to add the inner model radius to `rb`:

```
>> ar = rb(2)/rb(1);

>> rb = [rw/ar, rb];
```

where `ar` is the expansion factor.

Then, vertical distance is discretized by defining a vector `D` with `nz` layer thicknesses `D1`, `D2`, `D3`, … :

```
>> D = [D1; D2; D3; …];
```

where `nz` is the total number of layers counted from top to bottom.

Finally, the `Grid` object is created by calling the “`setgrid`” method of the `Model` object with `rb` and `D` as input arguments:

```
>> m.setgrid(rb, D, confined);
```

The third input argument `confined` is `true` if the aquifer system is confined and `false` if the system is unconfined. All arguments are required.

The `Grid` object properties can be verified by getting property `grid` of the `Model` object:

```
>> m.grid
```

After instantiation, all read only properties are set. Table 2 gives an overview.

Table 2. Read only properties for the `Grid` object.

Property	Description	Contents
`m.grid.confined`	aquifer system is confined?	`true` if confined `false` if unconfined
`m.grid.rb`	radius [L] of ring boundaries	`rb`
`m.grid.r`	radius [L] of nodal circles	`sqrt(rb(1:end-1).*rb(2:end))`
`m.grid.dr`	width [L] of rings	`diff(rb)`
`m.grid.nr`	number of rings	`nr`
`m.grid.zb`	level [L] of layer boundaries	`flipud(cumsum([0; flipud(D)]))`
`m.grid.z`	level [L] of layer middles	`(zb(1:end-1)+zb(2:end))/2`
`m.grid.D`	thickness [L] of layers	`D`
`m.grid.nz`	number of layers	`nz`
`m.grid.hs`	horizontal surface area [L^2] of rings	`pi*(rb(2:end).^2-rb(1:end-1).^2)`
`m.grid.vol`	volume [L^3] of rings	`D*hs`

Note that the lower grid boundary has level equal to `0` and the upper grid boundary has level equal to `sum(D)`. The length unit [L] may be chosen arbitrarily, but must be maintained when defining other length dependent parameters such as discharges, conductivities and storage coefficients.

Class Parameters

Once the `TimeSteps` and `Grid` objects have been created, a `Parameters` object is instantiated automatically by the `Model` object, enabling the modeler to define the required hydraulic parameters by setting the properties of the `Parameters` object.

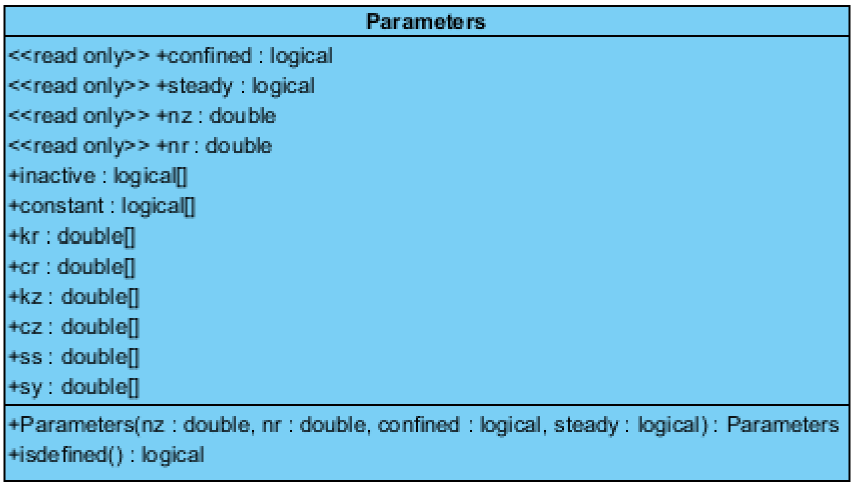

Parameters
<<read only>> +confined : logical
<<read only>> +steady : logical
<<read only>> +nz : double
<<read only>> +nr : double
+inactive : logical[]
+constant : logical[]
+kr : double[]
+cr : double[]
+kz : double[]
+cz : double[]
+ss : double[]
+sy : double[]
+Parameters(nz : double, nr : double, confined : logical, steady : logical) : Parameters
+isdefined() : logical

Figure 10. UML class diagram presenting public properties and methods for class `Parameters`. Data types followed by `[]` indicate arrays, otherwise a scalar value is required.

Figure 10 gives an overview of all public properties and methods for class `Parameters`. The read only properties cannot be modified and their contents is set by the constructor. The input arguments of the latter are adopted from the `TimeSteps` object and the `Grid` object: `nz` and `nr` are the number of layers and rings respectively, `confined` indicates whether the aquifer system is confined or unconfined, and `steady` indicates whether the simulation is steady or unsteady state. These properties are necessary to check the parameter matrix dimensions and to determine which parameters are required and which are not allowed. If the grid only contains one layer, for example, setting the vertical conductivity is not allowed. If the simulation is steady state, setting the specific storage coefficient is not allowed, and setting the specific yield is only required if the simulation is transient and the system is unconfined.

The `Parameters` object is stored in property `par` of the `Model` object:

```
>> m.par
```

The public properties available in the `Parameters` object are listed in Table 3. The method `isdefined` is called by the `Model` object to verify if all required properties are set.

Table 3. Public properties for the `Parameters` object.

Property	**Description**	**Size**	**Remarks**
`m.par.inactive`	indicates inactive rings	`nz x nr`	`true` for inactive rings set only if any inactive rings
`m.par.constant`	indicates constant-head rings	`nz x nr`	`true` for constant-head rings set only if any constant-head rings
`m.par.kr`	radial conductivity [L/T]	`nz x nr`	only values ≥ 0 allowed `NaN` values are ignored
`m.par.cr`	radial resistance [T]	`nz x (nr-1)`	overwrites `kr` values `NaN` and values ≤ 0 are ignored only required if `kr` is not set
`m.par.kz`	vertical conductivity [L/T]	`nz x nr`	only values ≥ 0 allowed `NaN` values are ignored only required if `nz` > 1
`m.par.cz`	vertical resistance [T]	`(nz-1) x nr`	overwrites `kz` values `NaN` and values ≤ 0 are ignored only required if `nz` > 1 and `kz` is not set
`m.par.ss`	specific storage [1/L]	`nz x nr`	only values ≥ 0 allowed `NaN` values are ignored only required if transient
`m.par.sy`	specific yield [-]	`1 x nr`	only values ≥ 0 allowed `NaN` values are ignored only required if transient and unconfined

Hydraulic conductivity, specific elastic storage coefficient and specific yield are well known parameters and do not need further explanation. The other properties, however, require some additional definitions:

- An inactive ring is an impermeable ring that is not part of the flow domain. It could also be defined by setting its hydraulic parameters equal to zero, but it is more straightforward to use the `inactive` property. Rings for which the `inactive` matrix entry is `true` are skipped during the iterative solving of the finite-difference equations. Inactive rings are used, for example, to implement the well casing.

- A constant-head ring is a ring in which the hydraulic head is kept constant during one stress period. The values of the constant head are defined by setting the initial head change property in the `Stresses` objects. In this way, the constant heads can be transient. Conductivity values (or resistances) need to be defined in constant-head rings, storage parameters are ignored since there is no storage change when the head remains constant. Constant-head rings are useful to define, for example, a top layer with a constant head.

- Radial resistance between to distances `r1` and `r2`, with `r2` > `r1`, is defined as `(r2-r1)/Kr`, where `Kr` is the radial component of the hydraulic conductivity. Theoretically, the distance between `r1` and `r2` should be infinitesimal small; in practice, this distance should be negligible. Radial resistances are useful, for example, to define an infinitesimal small well skin. Radial resistances are defined at the ring boundaries, and therefore, the size of the `cr` matrix must be `nz x (nr-1)`.

- Vertical resistance between two levels `z1` and `z2`, with `z2` > `z1`, is defined as `(z2-z1)/Kz`, where `Kz` is the vertical component of the hydraulic conductivity. Vertical resistances are useful to define, for example, (semi-)confining beds. In this case, the vertical resistance is defined as `b/Kz`, where `b` is the thickness of the confining bed. Vertical resistances are defined at the layer boundaries, hence, the size of the `cz` matrix must be `(nz-1) x nr`.

A final remark is about the size of the parameter matrices. Singleton dimensions are allowed for all matrices to simplify the user input. If the required size of a parameter matrix is `m x n`, then the input may also be an `m x 1` column vector, a `1 x n` row vector or a `1 x 1` scalar. In this way, parameter values do not have to be repeated when layers are homogeneous and setting a column vector with the values for each layer is sufficient in this case.

Class Stresses

Once the `TimeSteps` and `Grid` objects have been created, the `Model` object creates a `Stresses` object for each stress period defined in the `TimeSteps` object. Two kinds of stresses may be defined in a `Stresses` object: discharges and initial head changes. Discharges are used to simulate extraction and injection, and they are also useful to simulate recharge in the top layer of an unconfined system. Initial head changes are used to simulate slug tests, and as already explained, they may also be applied to manipulate the constant heads.

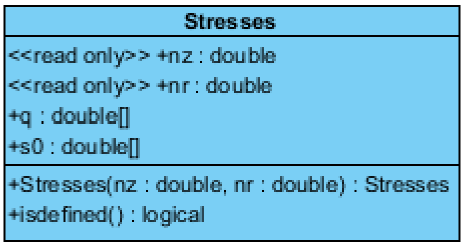

Stresses
<<read only>> +nz : double
<<read only>> +nr : double
+q : double[]
+s0 : double[]
+Stresses(nz : double, nr : double) : Stresses
+isdefined() : logical

Figure 11. UML class diagram presenting public properties and methods for class `Stresses`. Data types followed by `[]` indicate arrays, otherwise a scalar value is required.

Figure 11 gives an overview of all public properties and methods for class `Stresses`. The read only properties cannot be modified and their contents is set by the constructor. Input arguments `nz` and nr, the number of layers and rings respectively, are adopted from the Grid object. These properties are necessary to check the stress matrix dimensions.

The Stresses objects are stored in property stress of the Model object. As already mentioned, a Stresses object is created for each stress period and the Stresses object for the p-th stress period is:

```
    >> m.stress(p)
```

The public properties available in the Stresses objects are listed in Table 4. The method isdefined is called by the Model object to verify if any stress property is set. Note that it is not required each stress period has stresses defined. The recovery phase of a pumping test is an example of a stress period with no stresses defined. The entire simulation, however, should have at least one stress period with stresses defined. In case of a steady state simulation with no constant heads defined, the total discharge must be equal to the total recharge according to the law of mass conservation. These requirements are verified by the Model object before running the model.

Table 4. Public properties for the Stresses object corresponding to the p-th stress period.

Property	**Description**	**Size**	**Remarks**
m.stress(p).q	discharge [L^3/T]	nz x nr	q > 0 means extraction q < 0 means injection NaN values are not allowed
m.stress(p).s0	initial head change [L]	nz x nr	s0 > 0 means initial head rise s0 < 0 means initial head fall NaN values are not allowed

Defining a circular recharge area in the top layer is done by multiplying the recharge flux N [L/T] by the horizontal surface area of the k rings inside the recharge area:

```
    >> m.stress(p).q = zeros(nz,nr);

    >> m.stress(p).q(1,1:k) = -N * m.grid.hs(1:k);
```

The first statement initializes the q matrix for the p-th stress period by setting all entries zero, which corresponds to the default matrix. The second statement replaces the zero values by the recharge values [L^3/T] for the first k rings in the top layer. Note that the recharge values have a negative sign. Additionally, a well may be defined in the i-th layer which extracts water with a pumping rate Q [L^3/T] during the p-th stress period:

```
    >> m.stress(p).q(i,1) = Q;
```

If the simulation is steady state, then p is 1 and Q must be equal to sum(N*m.grid.hs(1:k)) if there are no constant-head rings defined.

The initial drawdown in the aquifer is assumed zero, which implies that all heads are equal. The s0 property allows to define an initial head change that differs from zero. For example, if a slug test is performed on a well in the i-th layer and the initial head change in the well is H0, than s0 is set as follows:

```
>> m.stress(1).s0 = zeros(nz,nr);

>> m.stress(1).s0(i,1) = H0;
```

The first statement initializes the initial head change matrix for the first stress period by setting all entries zero, which corresponds to the default matrix. The second statement replaces the zero value in the first ring of the `i`-th layer by `H0`. Initial head change `H0` is positive when water was poured into the well, and negative otherwise.

Initial head changes may also be defined for the subsequent stress periods, i.e. for stress periods `p` with `p` > 1. In this case, the initial head change matrix for stress period `p` is added to the drawdown matrix calculated at the end of the previous stress period `p-1`. This could be useful to define constant heads that change each stress period. Suppose a constant head is situated in ring `j` and layer `i`, then first, the constant head matrix is defined:

```
>> m.par.constant = false(nz,nr);

>> m.par.constant(i,j) = true;
```

The first statement initializes the constant head matrix, the second statement indicates that ring `(i,j)` has a constant head. Suppose the initial head in the aquifer is `h0` and the constant head in ring `(i,j)` is equal to `h(p)` during stress period `p`, then the initial head change matrix for the first stress period is defined as follows:

```
>> m.stress(1).s0 = zeros(nz,nr);

>> m.stress(1).s0(i,j) = h(1)-h0;
```

The initial head change matrix for the `p`-th stress period with `p` > 2 is defined as:

```
>> m.stress(p).s0 = zeros(nz,nr);

>> m.stress(p).s0(i,j) = h(p)-h(p-1);
```

In each definition, the first statement initializes the initial head change matrix, and the second statement sets the initial head change for ring `(i,j)`.

As is the case for the parameter matrices set in the `Parameters` object, the stress matrices may contain singleton dimensions. However, in most cases, the use of sparse matrices to define stresses will be more convenient and more effective in saving memory. See MATLAB help to get more information about sparse matrices.

Class Solver

MAxSym is provided with two solvers: the iterative Alternating Direction Implicit (ADI) method described in, for instance, Wang and Anderson (1982) and Lebbe (1999), and the Strongly Implicit Procedure (SIP) developed by Stone (1968). The SIP method is also provided with MODFLOW-2005 (Harbaugh, 2005) and the algorithm is described exhaustively by McDonald and Harbaugh (1988).

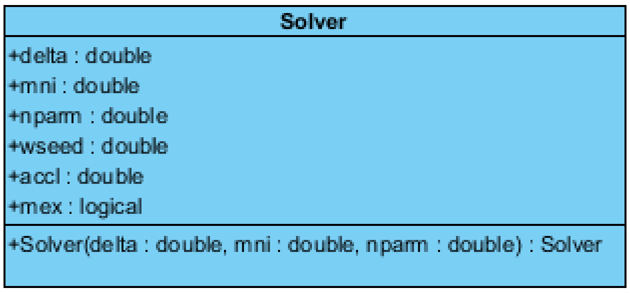


Figure 12. UML class diagram presenting public properties and methods for class `Solver`.

Figure 12 indicates the public methods and properties for class `Solver`. The only public method is the `Solver` constructor, which is invoked when the modeler calls the `setsolver` method of the `Model` object:

```
>> m.setsolver(delta,mni,nparm);
```

Both methods require the same input arguments: the first argument `delta` is the criterion for convergence, the second argument `mni` is the maximum number of iterations allowed for one time step, and the third argument `nparm` indicates the number of SIP iteration parameters. The latter is optional and if it is not given or smaller than 1, then ADI will be applied, otherwise, SIP will be used. The created `Solver` object is stored in the `Model` object's `solver` property:

```
>> m.solver
```

The modeler is able to set additional `Solver` properties. Table 5 gives an overview of all public properties.

Table 5. Public properties for the `Solver` object.

Property	**Description**	**Remarks**
`m.solver.delta`	criterion for convergence [L]	always required
`m.solver.mni`	maximum number of iterations	always required
`m.solver.nparm`	number of SIP iteration parameters	required for SIP
`m.solver.wseed`	seed for calculation of SIP iteration parameters	optional, default is calculated from the conductances
`m.solver.accl`	SIP accelerator	optional, default is `1`
`m.solver.mex`	call mex-file?	optional, default is `true`

The criterion for convergence and the maximum number of iterations is required for both ADI and SIP, the number of iteration parameters is required for SIP. In most cases, 5 iteration parameters are sufficient. Obviously, the seed for calculating the iteration parameters is used by the SIP solver only. If not set, the seed is derived from the conductances, which are calculated from the conductivities, the ring surfaces, and the distances between the nodal circles. The accelerator is also used by the SIP solver only. Its value must be between 0 and 1, and if not set, the default value of 1 is used.

Finally, the `Solver` class has a property `mex` to choose between the mex and the MATLAB routines. As already explained in the section about installing MAxSym, the mex files must be compiled before they can be used. Both routines give the same results, but the mex routines are much faster in solving the equations, and therefore, they are the default choice. It is also recommended to use the SIP solver, since it is more implicit than the ADI solver. As a consequence, SIP is faster and more accurate. The ADI solver should even be avoided when running models with aquifer recharge or constant-head rings, since it may give inaccurate results.

Run Model

Once all input is assigned to the `Model` object properties, the model is finally run by invoking the `run` method:

```
>> m.run
```

Before calling the appropriate solver routine, the `Model` object first checks if all required input is given. Then it calculates the flow constants, which are passed to the solver routine. The latter calculates drawdown for each ring and each time step. The calculated drawdown matrix is stored in property `s` of the `Model` object. After the model run is completed, it is good practice to verify the volumetric budget terms for each variable-head ring and at the end of each time step.

Table 6 gives an overview of the properties in class `Model` containing output results. Note that these properties are read only. Flow and budget term properties `qr`, `qz`, `qs`, `bud`, and `totbud` are also dependent, meaning that their contents is calculated whenever it is requested.

Table 6. Read only properties storing output results for the `Model` object.

Property	**Description**	**Size** [1]	**Dependent**
`m.s`	drawdown [L]	`nz x nr x nt`	no
`m.niter`	number of iterations	`(nt-1) x 1` if transient `1 x 1` if steady state	no
`m.qr`	radial flow terms [L^3/T]	`nz x (nr+1) x nt`	yes
`m.qz`	vertical flow terms [L^3/T]	`(nz+1) x nr x nt` empty if `nz` = 1	yes
`m.qs`	storage change terms [L^3/T]	`nz x nr x (nt-1)` empty if steady state	yes
`m.bud`	volumetric budget terms [L^3/T]	`nz x nr x nt`	yes
`m.totbud`	total volumetric budget [L^3/T]	`(nt-1) x 2` column 1: variable-head rings column 2: constant-head rings	yes

[1] `nt` is the number of simulation times or `nt = length(m.time.t)`, if steady state, `nt` is 1.

The definitions for radial flow terms `qr`, vertical flow terms `qz`, and storage change terms `qs` are given in previous chapter. The radial and vertical flow terms at the grid boundaries are zero. The volumetric budget terms `bud` are the sum of all these terms and the discharges `q` set in the `Stresses` objects. Hence, they indicate the net flow for each ring. According to the law of mass conservation, the budget terms should be close to zero for each ring, except for the constant-head rings, which act as sources or sinks.

The total volumetric budget for the entire model at the end of each time step is given by `totbud`. This property has two columns: the first column contains the total budget for all variable-head rings, the second column contains the total budget for all constant-head rings. The entries in the first column must also be close to zero. If there are no constant-head rings defined, the first column is calcutated alternatively as:

```
>> squeeze(sum(sum(m.bud,1),2))
```

which results in an `(nt-1) x 1` vector with the total volumetric budget for each time step.

Using standard plotting functions provided with MATLAB, it is straightforward to visualize results. For instance, a time-drawdown graph may be created using the function `semilogx`:

```
>> figure

>> semilogx(m.time.t,squeeze(m.s(i,j,:))')
```

The first statement creates a new figure, the second statement plots the drawdown in ring `(i,j)` as a function of the logarithm of time. The radial distance of the plotted drawdown is equal to `m.grid.r(j)`. The second argument passed to `semilogx` is transposed to allow a vector of ring indices for variable `j`.

Simularly, a distance-drawdown graph showing drawdown in layer `i` as a function of the logarithm of radial distance is plotted as follows:

```
>> figure

>> semilogx(m.grid.r,squeeze(m.s(i,:,k)))
```

The simulation time of the plotted drawdown is equal to `m.time.t(k)`, where variable `k` may be a vector of time indices.

Interpolate drawdown

MAxSym calculates drawdown at the nodal circles of the axi-symmetric grid and at the end of each time step. Consequently, drawdown must be interpolated to get results at arbitrary distances and at arbitrary times. It is assumed drawdown varies linearly as a function of *log(r)* between two nodal circles and as a function of *log(t)* between two simulation times, whereas drawdown within a layer is assumed not to vary in the vertical direction *z*.

Class `Model` is provided with three methods to perform (bi)linear interpolation:

- `s = m.interp1r(i,r,k,rw)`: interpolates drawdown in the dimension of *log(r)*
- `s = m.interp1t(i,j,t)`: interpolates drawdown in the dimension of *log(t)*
- `s = m.interp2(i,r,t,rw)`: interpolates drawdown in the dimensions of *log(r)* and *log(t)*

The methods use MATLAB functions `interp1` and `interp2` and have similar names. Technically, method `interp2` is said to overload function `interp2`. The signature of the methods is similar: the first argument refers to the layer dimension, the second argument refers to the dimension of radial distance, and the third dimension refers to the dimension of time. In this way, the order of the input arguments is the same as the order of the dimensions of drawdown and other MAxSym arrays. Depending on the dimension(s) in which drawdown is interpolated, the input arguments are indices or 'real' values. The interpolated drawdown array returned by the methods always has a size that corresponds to the length of the input arguments.

interp1r

Drawdown in layers `i` at arbitrary distances `r` and at simulation times `k` is calculated as:

```
>> s = m.interp1r(i,r,k);
```

where `size(s)` is `[length(i),length(r),length(k)]`. This statement is equivalent to the following statements using function `interp1`:

```
>> si = permute(m.s(i,:,k),[2 1 3]);
>> s = interp1(log10(m.grid.r),si,log10(r));
>> s = permute(s,[2 1 3]);
```

If all layers and/or all time steps must be considered, method `interp1r` allows the modeler to pass an empty array:

```
>> s = m.interp1r([],r,[]);
```

which is equivalent to:

```
>> s = m.interp1r(1:m.grid.nz,r,1:length(m.time.t));
```

All input arguments must be vectors. If `r` is an array of distances, interpolation is done as follows:

```
>> s = reshape(m.interp1r(i,r(:),k),[length(i),size(r),length(k)]);
```

to preserve the size of `r` in output `s`.

Method `interp1r` also accepts an optional argument to account for the well radius `rw`:

```
>> s = m.interp1r(i,r,k,rw);
```

In this case, drawdown at distances `r <= rw` is set equal to the drawdown in the first ring `s(i,1,k)`. If `rw` is not given, the radius of the first nodal circle is taken or `rw = m.grid.r(1)`. It is also possible to pass a well radius for each layer i. In this case `length(rw)` is equal to `length(i)`.

If the model is steady state, the input argument with simulation time indices is omitted:

```
>> s = m.interp1r(i,r);
```

Or when the well radius is given:

```
>> s = m.interp1r(i,r,rw);
```

Finally, the following command displays help on using method `interp1r`:

```
>> help MAxSym.Model.interp1r
```

interp1t

Drawdown in layers `i` at nodal circles `j` and at arbitrary times `t` is calculated as:

```
>> s = m.interp1t(i,j,t);
```

where `size(s)` is `[length(i),length(j),length(t)]`. This statement is equivalent to the following statements using function `interp1`:

```
>> si = permute(m.s(i,j,2:end),[3 1 2]);
>> s = interp1(log10(m.grid.t(2:end)),si,log10(t));
>> s = permute(s,[2 3 1]);
```

where all times `t` are greater than or equal to `m.grid.t(2)`.

If all layers and/or all nodal circles must be considered, method `interp1t` allows the modeler to pass an empty array:

```
>> s = m.interp1t([],[],t);
```

which is equivalent to:

```
>> s = m.interp1t(1:m.grid.nz,1:m.grid.nr,t);
```

All input arguments must be vectors. If `t` is an array of times, interpolation is done as follows:

```
>> s = reshape(m.interp1t(i,j,t(:)),[length(i),length(j),size(t)]);
```

to preserve the size of `t` in output `s`.

Drawdown at times `t < m.time.t(2)` is set equal to the initial drawdown `s(i,j,1)`.

It is useless to invoke method `interp1t` in case of a steady state model.

Finally, the following command displays help on using method `interp1t`:

```
>> help MAxSym.Model.interp1t
```

interp2

Drawdown in layers `i` at arbitrary distances `r` and at arbitrary times `t` is calculated as:

```
>> s = m.interp2(i,r,t);
```

where `size(s)` is `[length(i),length(r),length(t)]`. This statement is equivalent to the following statements using function `interp2`:

```
>> [ri,ti] = deal(log10(m.grid.r),log10(m.time.t(2:end)))
>> si = permute(m.s(i,:,2:end),[2 3 1]);
>> for n = 1:length(i)
       s(:,:,n) = interp2(ti,ri,si,log10(t(:)'),log10(r(:)));
   end
>> s = permute(s,[3 1 2]);
```

where all times `t` are greater than or equal to `m.grid.t(2)`.

If all layers must be considered, method `interp2` allows the modeler to pass an empty array:

```
>> s = m.interp2([],r,t);
```

which is equivalent to:

```
>> s = m.interp2(1:m.grid.nz,r,t);
```

All input arguments must be vectors. If `r` and `t` are arrays, interpolation is done as follows:

```
>> s = reshape(m.interp2(i,r(:),t(:)),[length(i),size(r),size(t)]);
```

to preserve the size of `r` and `t` in output `s`.

Method `interp2` also accepts an optional argument to account for the well radius `rw`:

```
>> s = m.interp2(i,r,t,rw);
```

In this case, drawdown at distances `r <= rw` is set equal to the drawdown in the first ring `s(i,1,t)`. If `rw` is not given, the radius of the first nodal circle is taken or `rw = m.grid.r(1)`. It is also possible to pass a well radius for each layer i. In this case `length(rw)` is equal to `length(i)`.

Drawdown at times `t < m.time.t(2)` is set equal to the initial drawdown `s(i,r,1)`.

It is useless to invoke method `interp2` in case of a steady state model.

Finally, the following command displays help on using method `interp2`:

```
>> help MAxSym.Model.interp2
```

Summary

The different steps explained in the previous paragraphs are summarized briefly.

1. Create a Model object.

   ```
   >> m = MAxSym.model;
   ```

2. Define time steps and stress periods by creating a `TimeSteps` object stored in `m.time`.

   If steady state:

   ```
   >> m.settime(0);
   ```

   If transient:

   ```
   >> p1 = logspace(log10(t1),log10(t2),n1+1);

   >> p2 = logspace(log10(t2),log10(t3),n2+1);

   >> p3 = logspace(log10(t3),log10(t4),n3+1);

   >> ...;

   >> t = [p1, p2(2:end), p3(2:end), ...];

   >> dt = diff(t);

   >> m.settime(dt,[n1,n2,n3,…]);
   ```

   where the first stress period `p1` has `n1` time steps between `t1` and `t2`, the second stress period `p2` has `n2` time steps between `t2` and `t2`, etc. The first input argument for `settime` contains the time steps, the second argument indicates the number of time steps for each stress period.

3. Define the axi-symmetric grid by creating a `Grid` object stored in `m.grid`.

   First, the rings are defined:

   ```
   >> rb = logspace(log10(rw),log10(rout),nr);
   
   >> ar = rb(2)/rb(1);
   
   >> rb = [rw/ar, rb];
   ```

   where `rb` is a vector with the radii of the ring boundaries, `rw` is the well radius, `rout` is the outer grid radius, `nr` is the number of rings, and `ar` is the expansion factor.

   Then, the `nz` layers with thickness `D1`, `D2`, `D3`, … are defined:

   ```
   >> D = [D1; D2; D3; …];
   ```

   Finally, the `Grid` object is created:

   ```
   >> m.setgrid(rb,D,confined);
   ```

   The third input argument `confined` is `true` if the aquifer system is confined and `false` if the system is unconfined.

4. Define hydraulic parameters by setting properties in the `Parameters` object `m.par`.

   The following parameters may be set:

   `m.par.inactive`: inactive rings, `nz x nr` matrix, optional
   `m.par.constant`: constant head rings, `nz x nr` matrix, optional
   `m.par.kr`: radial conductivity, `nz x nr` matrix, required if `cr` is not set
   `m.par.cr`: radial resistance, `nz x (nr-1)` matrix, required if `kr` is not set
   `m.par.kz`: vertical conductivity, `nz x nr` matrix, required if `nz` > 1 and `cz` is not set
   `m.par.cz`: vertical resistance, `(nz-1) x nr` matrix, required if `nz` > 1 and `kz` is not set
   `m.par.ss`: specific elastic storage coefficient, `nz x nr` matrix, required if transient
   `m.par.sy`: specific yield, `1 x nr` vector, required if transient and unconfined

5. Define stresses for each stress period by setting properties in the `Stresses` objects `m.stress`.

   The following stresses may be defined for stress period `p`:

   `m.stress(p).q`: discharge, `q` > 0 means extraction, `nz x nr` matrix, optional
   `m.stress(p).s0`: initial head change, `s0` > 0 means head rise, `nz x nr` matrix, optional

6. Define the solver by creating a `Solver` object stored in `m.solver`.

   If ADI is chosen:

   ```
   >> m.setsolver(delta,mni);
   ```

If SIP is chosen:

```
>> m.setsolver(delta,mni,nparm);
```

The following properties can be modified after instantiation:

`m.solver.delta`: criterion for convergence, required
`m.solver.mni`: maximum number of iterations, required
`m.solver.nparm`: number of SIP iteration parameters, required for SIP
`m.solver.wseed`: seed for calculation of SIP iteration parameters, optional
`m.solver.accl`: SIP accelerator, optional, default is `1`
`m.solver.mex`: call mex routine, optional, default is `true`

7. Run the model.

```
>> m.run;
```

The model output is set in the following properties, with `nt` the number of simulation times:

`m.s`: drawdown, `nz x nr x nt` array
`m.niter`: number of iterations, `(nt-1) x 1` vector if transient, scalar if steady state
`m.qr`: radial flow terms, `nz x (nr+1) x nt` array
`m.qz`: vertical flow terms, `(nz+1) x nr x nt` array if `nz` > 1, empty if `nz` = 1
`m.qs`: storage change terms, `nz x nr x (nt-1)` array if transient, empty if steady state
`m.bud`: volumetric budget terms, `nz x nr x (nt-1)` array
`m.totbud`: total volumetric budget, `(nt-1) x 2` array
`m.totbud(:,1)` for variable-head rings, `m.totbud(:,2)` for constant-head rings

8. Interpolate drawdown.

```
>> s = m.interp1r(i,r,k,rw);
```

Linear interpolation of drawdown in the dimension of *log(r)*, where `i` is a vector with layer indices, `r` is a vector with arbitrary distances, `k` is a vector with simulation time indices, and `rw` (optional) is the well radius.

```
>> s = m.interp1t(i,j,t);
```

Linear interpolation of drawdown in the dimension of *log(t)*, where `i` is a vector with layer indices, `j` is a vector with ring indices, and `t` is a vector with arbitrary times.

```
>> s = m.interp2(i,r,t,rw);
```

Bilinear interpolation of drawdown in the dimensions of *log(r)* and *log(t)*, where `i` is a vector with layer indices, `r` is a vector with arbitrary distances, `t` is a vector with arbitrary times, and `rw` (optional) is the well radius.

Modeling examples

The steps explained in previous section are demonstrated by a large number of examples. At the same time, these examples illustrate the many applications of MAxSym. If an analytical solution exists to the problem at hand, it is also presented to compare the MAxSym results with. In this way, the MAxSym code is verified throughout the examples. Verification of the MAxSym code against MODFLOW-2005 (Harbaugh, 2005) and several (semi) analytical models including TTim (Bakker, 2010) was already done by Louwyck et al. (in press). The examples for which no analytical solution exists to the author's knowledge, show the added value of MAxSym. A distinction is made between steady state and transient models, and it is also illustrated how to construct simple three-dimensional models for multiple wells applying the principle of superposition. Finally, a brief introduction is given on inverse modeling and two examples are presented that illustrate the analysis of pumping and slug tests.

The given examples are fictitious and do not correspond to real aquifer systems. In some cases, the quality of presented figures might be poor. However, the reader is able to run the examples by himself and reproduce the figures, since the MATLAB code is presented in the text and the corresponding scripts are provided with the MAxSym source files in the "Examples" directory. Before running a script, it is recommended to first execute command `clear classes` at the MATLAB prompt. This clears all variables, functions, and class definitions from the base workspace. See MATLAB help for more information on the `clear` command. Some of the presented MATLAB statements could be optimized for better performance. However, learning how to write efficient MATLAB code is out of the scope of this manual.

Steady state models

Extraction from confined aquifer

The simplest radial flow model is the Thiem (1906) model, which is used to simulate permanent flow towards a fully penetrating well in a confined homogeneous layer. In this example, it is illustrated how a Thiem model is setup using MAxSym and the results are compared to the analytical solution, which is calculated as:

$$s(r) = \frac{Q}{2\pi T}\ln\left(\frac{r}{R}\right) \qquad (r \leq R)$$

where *s* is the drawdown, *r* is the radial distance, *T* is the aquifer transmissivity, *Q* is the well discharge, and *R* is the radius of influence. Drawdown is zero for distances $r \geq R$.

As example, a Thiem model is set up with aquifer transmissivity *T* equal to 50 m²/d and well discharge *Q* equal to 100 m³/d. The radius of the inner grid boundary is 0.1 m, the radius of the outer grid boundary is 1000 m, and 40 rings are considered. The last ring is defined as a constant head ring with zero drawdown and the radius of its nodal circle corresponds to the radius of influence *R*. The following statements are found in script "..\Examples\SteadyState\confined.m":

```
% create Model object
m = MAxSym.Model;

% define steady state
m.settime(0);
```

```
% set grid
m.setgrid(logspace(-1,3,41),1,true);

% set parameters
m.par.constant = [false(1,m.grid.nr-1), true];
m.par.kr = 50;

% set stresses
m.stress.q = [100, zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,1);

% run model
m.run;

% analytical solution
s = m.stress.q(1)/2/pi/m.par.kr * log(m.grid.r./m.grid.r(end));

% distance-drawdown graph
figure
semilogx(m.grid.r,m.s,'k-',m.grid.r,s,'kx')
set(gca,'fontsize',12)
xlabel('distance (m)')
ylabel('drawdown (m)')
```

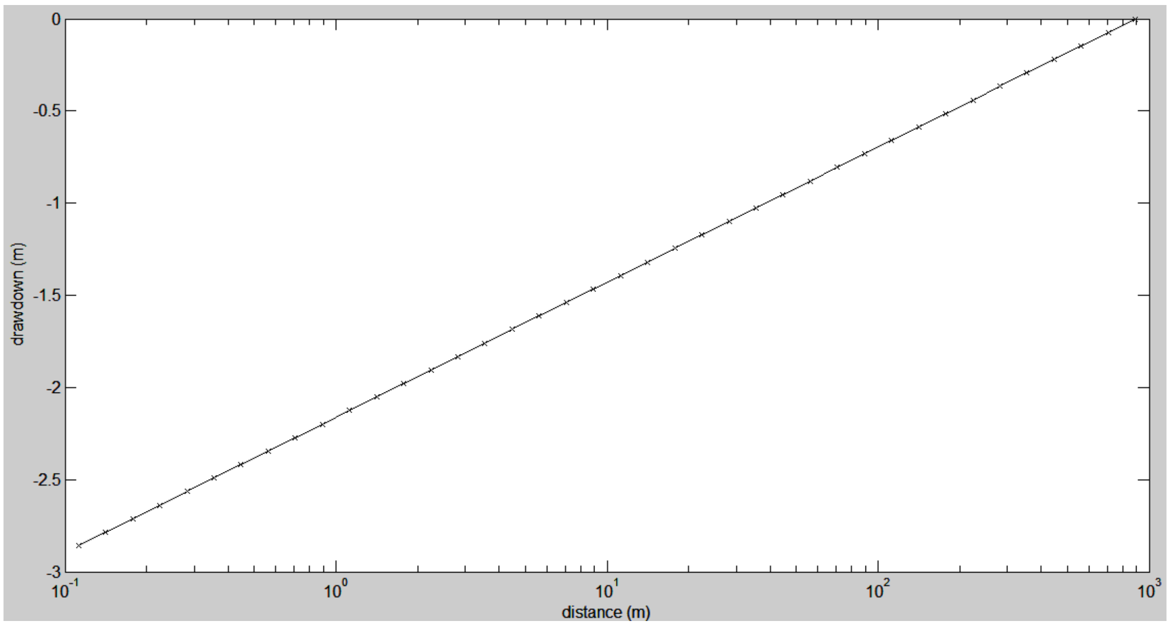


Figure 13. Plot of drawdown versus distance for the Thiem model. The solid line is calculated using MAxSym, the crosses correspond to the analytical solution.

The resulting plot in Figure 13 shows a graph of drawdown as a function of *log(r)*. The volumetric budget for the entire model is calculated as the sum of the budget terms of the variable head cells or `sum(m.bud(1:end-1))`. An easier way to check total volumetric budget is to access property `m.totbud`. The first element returns total budget for the variable-head rings:

```
>> m.totbud(1) % variable heads
```

```
ans =

 -3.2259e-012
```

The result is close to zero, as it should. The second element returns total budget for the constant-head rings:

```
>> m.totbud(2) % constant heads

ans =

  100.0000
```

There is only one constant-head ring, so this result is equal to the budget term in the last ring. The result of 100 m³/d is also equal to the well discharge, as is required according to the law of mass conservation. Radial flow terms `m.qr` are also equal to -100 m³/d, as they should.

The Thiem model is linear and one-dimensional, hence, the system can be solved directly without having to iterate. Therefore, the ADI solver is used and the maximum number of iterations `mni` is set 1. The criterion of convergence is set 1e-5, but its value is irrelevant if only 1 iteration is performed. The radial discretization is also irrelevant for a confined single layer model, since MAxSym uses the Thiem equation to calculate the radial flow between two adjacent rings.

This also explains the assumption that drawdown varies linearly as a function of *log(r)* between two nodal circles to apply interpolation for calculation of drawdown at an arbitrary distance. For instance, if drawdown is needed at 100 m from the well, than the MATLAB function `interp1` may be used and the logarithm of radial distances must be passed:

```
>> interp1(log(m.grid.r),m.s,log(100))

ans =

   -0.6963
```

However, the `Model` class is provided with methods to interpolate drawdown, and calling method `m.interp1r` is the easiest way to calculate drawdown at a distance of 100 m:

```
>> m.interp1r(1,100)

ans =

   -0.6963
```

This gives the same result as applying the Thiem equation:

```
>> m.stress.q(1)/2/pi/m.par.kr*log(100/m.grid.r(end))

ans =

   -0.6963
```

A final remark concerns the use of ADI when a Thiem model is set up defining two constant drawdowns instead of one constant drawdown and the extraction rate. Consider the following simple model that consists of three rings where the first and the last ring have a constant drawdown equal to 1 and 0 respectively:

```
>> clear classes
>> m = MAxSym.Model;
>> m.settime(0);
>> m.setgrid(logspace(0,1,4),1,true);
>> m.par.kr = 100;
>> m.par.constant = logical([1 0 1]);
>> m.stress.s0 = [1 0 0];
```

Because the logarithm of the distance between the nodal circles is equal, the drawdown in the middle ring must be 0.5. Running the model using ADI, however, gives a wrong result:

```
>> m.setsolver(1,1)
>> m.run
>> m.s

ans =

     1     0     0
```

whereas the SIP solver yields the correct result:

```
>> m.setsolver(1,1,1)
>> m.run
>> m.s

ans =

    1.0000    0.5000         0
```

The discharge is given by:

```
>> m.qr

ans =

         0  409.3129  409.3129         0
```

which is the same as the discharge derived from the analytical expression. From this simple model it is concluded that the use of ADI must be avoided to solve models containing constant heads only.

Extraction from unconfined aquifer

The Thiem-Dupuit formula is used to simulate permanent flow towards a well in an unconfined aquifer:

$$s(r) = \sqrt{D^2 - \frac{Q}{\pi K^r} \ln\left(\frac{R}{r}\right)} - D \qquad (r \leq R)$$

where *s* is the drawdown, *r* is the radial distance, *D* is the aquifer thickness, *Q* is the well discharge, K^r is the radial conductivity, and *R* is the radius of influence. Drawdown is zero for distances $r \geq R$, meaning that the hydraulic head is equal to the layer thickness *D*. Note that calculations that incorporate the Dupuit (1863) assumption will indicate a lower water table in the vicinity of pumped wells and when a seepage surface can be expected.

As an example, consider an aquifer with a thickness of 10 m, radial conductivity equal to 5 m/d, and well discharge equal to 100 m³/d. The radius of the inner grid boundary is 0.1 m, the radius of the outer grid boundary is 1000 m, and 40 rings are considered. The last ring is defined as a constant head ring with zero drawdown and the radius of its nodal circle corresponds to the radius of influence *R*. The following statements are found in script "..\Examples\SteadyState\unconfined.m":

```
% create Model object
m = MAxSym.Model;

% define steady state
m.settime(0);

% set grid
m.setgrid(logspace(-1,3,41),10,false);

% set parameters
m.par.constant = [false(1,m.grid.nr-1), true];
m.par.kr = 5;

% set stresses
m.stress.q = [100, zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,100,5);

% run model
m.run;

% analytical solution
s = sqrt(m.grid.D^2 - m.stress.q(1)/pi/m.par.kr * ...
    log(m.grid.r(end)./m.grid.r)) - m.grid.D;

% distance-drawdown graph
figure, semilogx(m.grid.r,m.s,'k-',m.grid.r,s,'kx')
set(gca,'fontsize',12)
xlabel('distance (m)')
ylabel('drawdown (m)')
```

The resulting plot in Figure 14 shows a graph of drawdown as a function of *log(r)*. This graph may be compared with Figure 13 that shows the distance-drawdown graph of the corresponding Thiem model. The only difference between the two models is the Thiem model assumes a constant saturated thickness, while in the Dupuit model the saturated thickness is equal to the hydraulic head, and consequently it is a function of radial distance. It is seen that the Dupuit model calculates a larger drawdown close to the well. If the aquifer is phreatic, the Dupuit model is likely to be more accurate, but it is not exact either, since application of the Dupuit assumption leads to a lower water table in the vicinity of pumped wells, as already mentioned. This remark also holds for MAxSym models of multi-layer systems with a phreatic top layer that contains an extraction well.

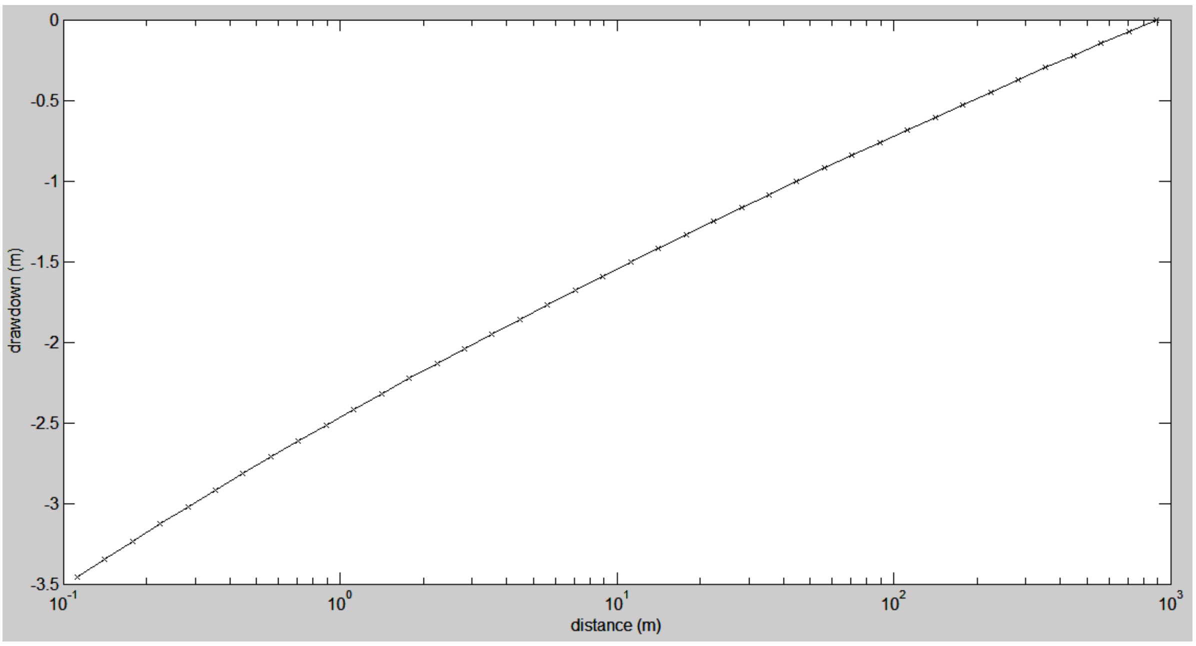


Figure 14. Plot of drawdown versus distance for the Thiem-Dupuit model. The solid line is calculated using MAxSym, the crosses correspond to the analytical solution.

The volumetric budget for the entire model `m.totbud(1)` is 1.7195e-12 m^3/d. The budget term in the last ring `m.bud(end)` is 100 m^3/d, which is equal to the well discharge, as is required according to the law of mass conservation. Radial flow terms are also equal to -100 m^3/d, as they should.

The Thiem-Dupuit model is non-linear, and therefore the system must be solved iteratively. The number of iterations the SIP solver needed to solve the system is 7, which is stored in `m.niter`. Note that the model run is aborted when drawdown in one of the rings is greater than the aquifer thickness.

Extraction from unconfined aquifer with recharge

Drawdown due to permanent flow towards a well in an unconfined aquifer with an infiltration flux *N* is given by the following equation (Verruijt, 1970):

$$s(r) = \sqrt{D^2 - \frac{Q}{\pi K^r}\ln\left(\frac{R}{r}\right) - \frac{N}{2K^r}(r^2 - R^2)} - D \qquad (r \leq R)$$

where *s* is the drawdown, *r* is the radial distance, *D* is the aquifer thickness, *Q* is the well discharge, K^r is the radial conductivity, and *R* is the radius of influence. Drawdown is zero for distances $r \geq R$, meaning that the hydraulic head is equal to the layer thickness *D*. If the radius of influence *R* is determined by a circular recharge area, where $Q = \pi R^2 N$, then substituting *R* leads to:

$$s(r) = \sqrt{D^2 - \frac{Q}{2\pi K^r}\left[\ln\left(\frac{Q}{\pi N r^2}\right) - 1\right] - \frac{Nr^2}{2K^r}} - D \qquad \left(r \leq \sqrt{Q/\pi N}\right)$$

Note that this model also applies the Dupuit (1863) assumption.

As an example, suppose the aquifer has a thickness of 10 m, radial conductivity is 5 m/d, well discharge is 100 m^3/d, and recharge flux is 5e-4 m/d. The radius of the inner grid boundary is 0.1 m, the radius of the outer grid boundary is determined by the well discharge and the areal recharge as explained above. The radial extension of the grid is discretized into 30 rings. Additionally, a very small constant head ring is added at the outer grid boundary to implement the boundary condition of zero drawdown at distance *R*. The discharge matrix `m.stress.q` defines the well discharge *Q* for the first ring and the recharge for the other rings. For each ring, the recharge flux must be converted to a negative discharge value by multiplying –*N* by the horizontal surface area of the rings. The recharge in the first ring is neglected.

The following statements are found in script "..\Examples\SteadyState\unconfinedPlusRecharge.m":

```
% Radius of influence
Q = 100;
N = 5e-4;
R = sqrt(Q/pi/N);

% create Model object
m = MAxSym.Model;

% define steady state
m.settime(0);

% set grid
rb = logspace(-1,log10(R),31);
m.setgrid([rb,rb(end)+1e-5],10,false);

% set parameters
m.par.constant = [false(1,m.grid.nr-1), true];
m.par.kr = 5;

% set stresses
m.stress.q = [Q, -N*m.grid.hs(2:end-1), 0];

% set solver
m.setsolver(1e-5,100,5);

% run model
m.run;

% analytical solution
s = sqrt(m.grid.D^2 - ...
    Q/2/pi/m.par.kr * ...
    (log(Q/pi/N./(m.grid.r).^2)-1) - ...
    N/2/m.par.kr*m.grid.r.^2) - ...
    m.grid.D;

% distance-drawdown graph
figure
semilogx(m.grid.r,m.s,'k-',m.grid.r,s,'kx')
set(gca,'fontsize',12)
xlabel('distance (m)')
ylabel('drawdown (m)')
```

Figure 15 shows the resulting plot of drawdown as a function of *log(r)* and compares the MAxSym result to the analytical solution. The plot may be compared with the distance-drawdown graph from previous example (Figure 14). The only difference is the addition of a circular infiltration area which determines the radius of influence *R*.

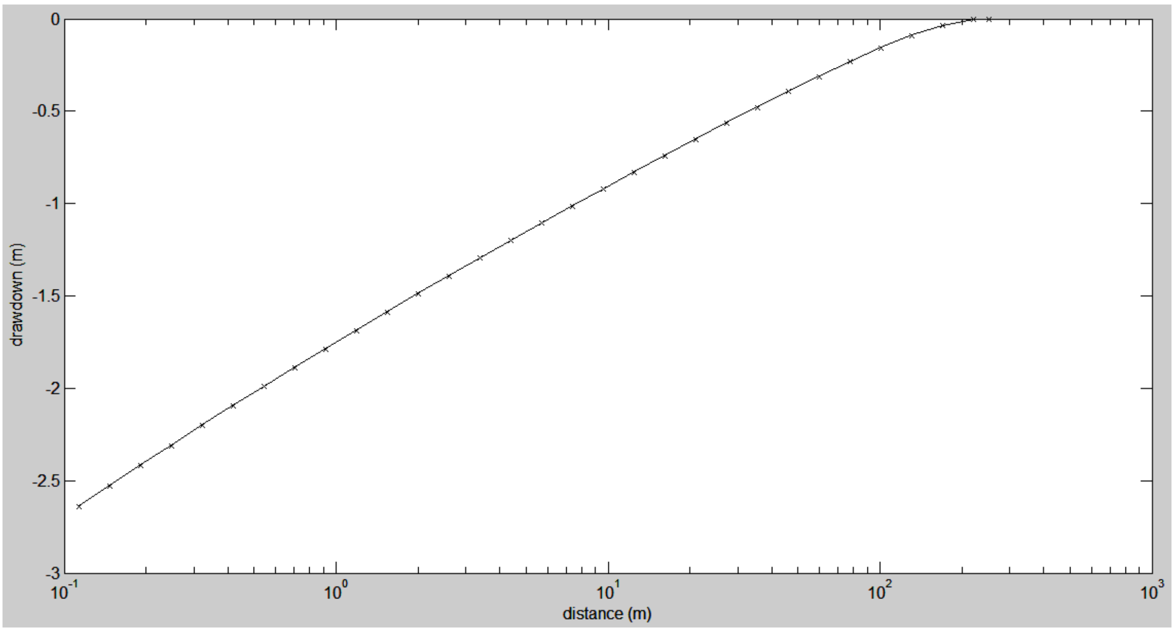


Figure 15. Plot of drawdown versus distance for the Thiem-Dupuit model with recharge. The solid line is calculated using MAxSym, the crosses correspond to the analytical solution.

The SIP solver was used and 6 iterations were required to solve the system of equations. In general, the use of SIP is recommended to solve models with areal infiltration. Total volumetric budget `m.totbud(1)` is 1.1918e-12 m³/d. The volumetric budget for the constant head ring `m.totbud(2)` is also close to zero, since the total areal recharge `abs(sum(m.stress.q(2:end-1)))` is equal to the well discharge `m.stress.q(1)`.

Extraction from leaky aquifer

The simplest example of a multi-layer system concerns the simulation of permanent flow to a fully penetrating well in a leaky aquifer of infinite lateral extent. The aquifer is bounded below by an impervious layer, and above by an aquitard overlain by a constant-head aquifer. The aquitard is incompressible and has vertical flow only. Under these conditions and assumptions, drawdown in the leaky aquifer is calculated using the De Glee (1930) formula:

$$s(r) = \frac{-Q}{2\pi T} K_0\left(\frac{r}{\sqrt{TC^z}}\right)$$

where *s* is the drawdown, *Q* is the well discharge, *T* is the transmissivity of the leaky aquifer, C^z is the vertical resistance of the aquitard, and *r* is the radial distance. The function K_0 is the modified Bessel function of the second kind and zero order, which is evaluated in MATLAB using function `besselk`.

In this example, the model parameters are *T* = 10 m²/d, C^z = 1000 d, and *Q* = 100 m³/d. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 80 rings are defined. Script is "..\Examples\SteadyState\leaky.m" sets the model up and runs it:

```
% create Model object
m = MAxSym.Model;

% define steady state
m.settime(0);

% set grid
m.setgrid(logspace(-1,7,81),[1;1],true);

% set parameters
m.par.constant = [true;false];
m.par.kr = [0;10];
m.par.cz = 1000;

% set stresses
m.stress.q = zeros(m.grid.nz,m.grid.nr);
m.stress.q(2,1) = 100;

% set solver
m.setsolver(1e-5,1);

% run model
m.run;

% analytical solution
s = -m.stress.q(2,1)/2/pi/m.par.kr(2) * ...
    besselk(0,m.grid.r/sqrt(m.par.kr(2)*m.par.cz));

% distance-drawdown graph
figure
semilogx(m.grid.r,m.s(2,:),'k-',m.grid.r,s,'kx')
set(gca,'fontsize',12)
xlabel('distance (m)')
ylabel('drawdown (m)')
```

Two layers are defined in the MAxSym model, and the top layer is defined as constant-head layer by setting the `constant` property in the `Parameters` object. The aquitard is conceptualized as a semi-confining bed between the two layers. This is done by assigning its resistance C^z to property `cz`. Thickness and radial conductivity values for the top layer are irrelevant, because there is no radial flow in the layer, but MAxSym still requires a value is set. The value for the `confined` property is irrelevant for the same reason. The thickness of the second layer is set 1, so its radial conductivity is equal to the aquifer transmissivity *T*.

Figure 16 shows the resulting distance-drawdown graph and compares the MAxSym results to the analytical solution. The ADI solver was used to solve the system of equations and only 1 iteration was needed since the system can be solved directly. Total volumetric budget for the variable-head rings `m.totbud(1)` is equal to -4.5825e-13 m^3/d, which could also be obtained by taking the sum of all budget terms from the lower layer: `sum(m.bud(2,:))`. Total volumetric budget for the constant-head rings `m.totbud(2)` could also be calculated by taking the sum of all budget terms of the top layer: `sum(m.bud(1,:))`. The result is equal to the well discharge, as is required according to the law of mass conservation.

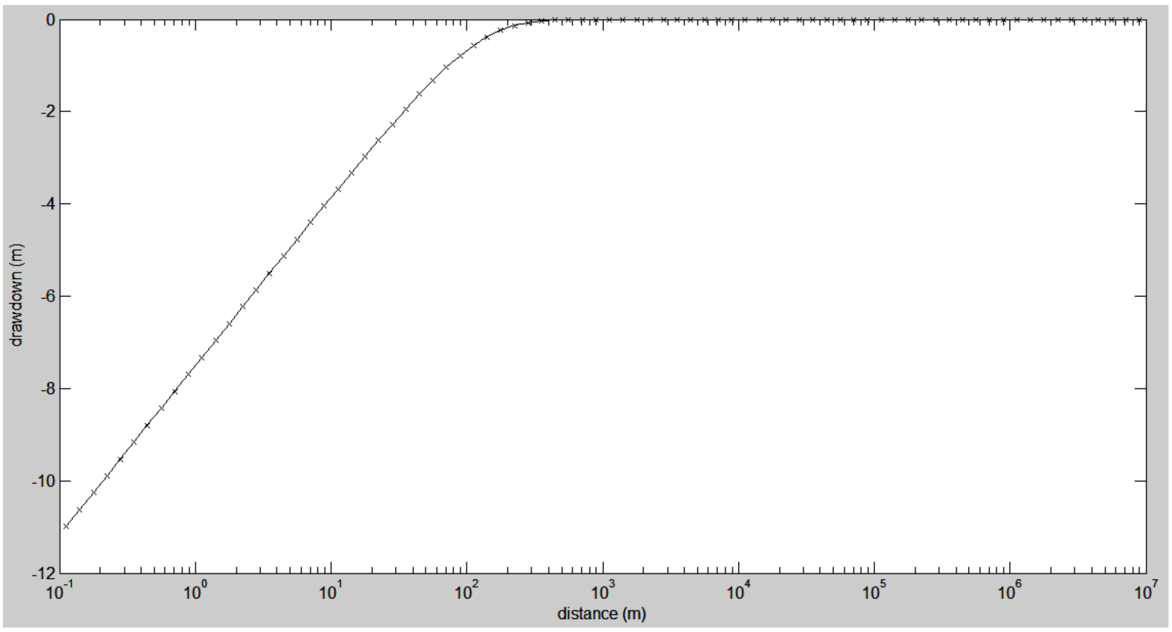


Figure 16. Plot of drawdown versus distance for the De Glee model. The solid line is calculated using MAxSym, the crosses correspond to the analytical solution.

The De Glee model assumes the leaky aquifer has an infinite lateral extent. We notice this assumption is met by the MAxSym model by verifying drawdown and flow properties for the last ring, which are all very close to zero:

```
>> m.s(2,end)

ans =

 -3.8835e-194

>> m.qr(2,end-1)

ans =

 -4.5025e-183

>> m.qz(2,end)

ans =

 -4.5025e-183
```

Extraction from leaky multi-aquifer system

The first who generalized the De Glee formula for a multi-layer system was Hemker (1984). A system of *n* aquifers of infinite lateral extent is considered, confined by *n+1* aquitards. A constant head layer is assumed above the upper aquitard as well as below the lower aquitard. The transmissivity of aquifer *i* is T_i and the vertical resistance C^z of aquitard *i* is denoted as c_i. Aquifers and aquitards are numbered from top to bottom. Each aquifer *i* may contain a fully penetrating well from which water is extracted with a constant discharge Q_i. The analytical solution is obtained by decomposing the system matrix *A* into its eigenvalues and eigenvectors:

$$A = VWV^{-1}$$

where *A* is an *n x n* tridiagonal matrix defined as:

$$A = \begin{bmatrix} 1/T_1c_1 + 1/T_1c_2 & -1/T_1c_2 & 0 & \cdots & 0 & 0 \\ -1/T_2c_2 & 1/T_2c_2 + 1/T_2c_3 & -1/T_2c_3 & \cdots & 0 & 0 \\ 0 & -1/T_3c_3 & 1/T_3c_3 + 1/T_3c_4 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1/T_nc_n & 1/T_nc_n + 1/T_nc_{n+1} \end{bmatrix}$$

W is an *n x n* diagonal matrix with the *n* eigenvalues w_i along the main diagonal, and *V* is an *n x n* matrix containing the corresponding eigenvectors in its columns. The drawdown in the *n* aquifers is obtained by the following matrix equation:

$$\bar{s} = VKV^{-1}\bar{q}$$

where $\bar{s}$ is a column vector containing the *n* drawdown values $s_i(r)$ at distance *r* for each aquifer *i*, *K* is an *n x n* diagonal matrix with $K_0(rw_i^{1/2})$ along the main diagonal with K_0 the modified Bessel function of the second kind and zero order, and $\bar{q}$ is an *n x 1* column vector with the *i*-th entry equal to $-Q_i/(2\pi T_i)$.

In this example, we consider 3 aquifers with T_1 = 100 m²/d, T_2 = 10 m²/d, and T_3 = 200 m²/d. The resistances for the 4 aquitards are c_1 = 100 d, c_2 = 500 d, c_3 = 300 d, and c_4 = 1000 d. An extraction well is considered in aquifer 1 with Q_1 = 100 m³/d and in aquifer 3 with Q_3 = 1000 m³/d. The following statements are found in script "..\Examples\SteadyState\leakyMultiAquifer.m":

```
% input
T = [100; 10; 200];
c = [100; 500; 300; 1000];
Q = [100; 0; 1000];

% create Model object
m = MAxSym.Model;

% define steady state
m.settime(0);

% set grid
m.setgrid(logspace(-1,7,81),ones(5,1),true);

% set parameters
m.par.constant = [true; false(3,1); true];
m.par.kr = [0; T; 0];
m.par.cz = c;

% set stresses
m.stress.q = zeros(m.grid.nz,m.grid.nr);
m.stress.q(2:end-1,1) = Q;

% set solver
m.setsolver(1e-5,100,5);

% run model
m.run;
```

```
% analytical solution
a = 1./T./c(1:end-1);
b = 1./T./c(2:end);
q = -Q/2/pi./T;
A = diag(a+b) - diag(a(2:end),-1) - diag(b(1:end-1),1);
[V,W] = eig(A);
for i = 1:m.grid.nr
    K = diag(besselk(0,m.grid.r(i)*sqrt(diag(W))));
    s(:,i) = V*K/V*q;
end

% distance-drawdown graph
figure
semilogx(m.grid.r,m.s(2:end-1,:)','-')
hold on
semilogx(m.grid.r,s,'x')
set(gca,'fontsize',12)
xlabel('distance (m)')
ylabel('drawdown (m)')
legend(strcat('layer',num2str((1:3)')));
```

The MAxSym solution is obtained by considering 5 layers where the top and bottom layers are constant-head layers and layers 2, 3, and 4 correspond to the 3 aquifers. The aquitards are implemented by assigning the aquitard resistances to the `cz` property of the `Parameters` object. Each layer thickness is set 1, so aquifer transmissivities can be assigned directly to the `kr` property. Transmissivity and thickness for the constant-head layers must also be set, although they are not considered in the calculations, because there is no radial flow in these layers. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 80 rings are defined. The resulting distance-drawdown graph is shown in Figure 17.

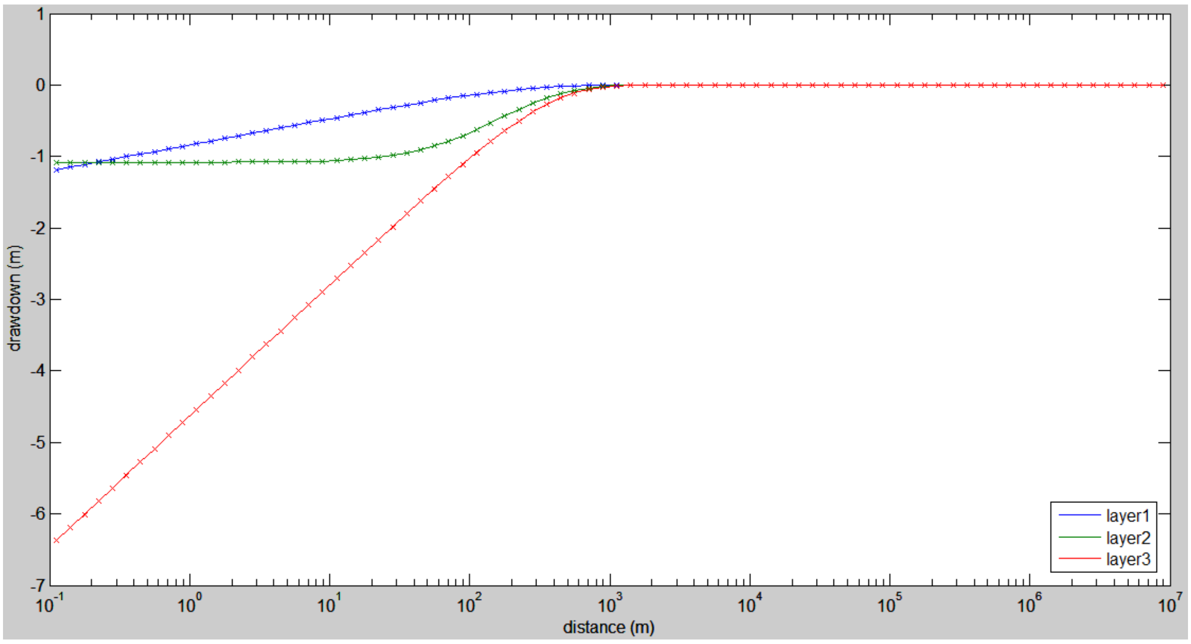


Figure 17. Plot of drawdown versus distance for a steady state multi-layer model. The solid line is calculated using MAxSym, the crosses correspond to the analytical solution given by Hemker.

The SIP solver is used and `m.niter` indicates that 8 iterations are required to solve the system of equations. Total volumetric budget for the variable-head layers `m.totbud(1)` is 2.7245e-4 m³/d. A smaller budget could be obtained by considering a smaller criterion for convergence. Total volumetric budget for the constant-head layers `m.totbud(2)` is equal to `sum(m.stress.q(:))` according to the law of mass conservation. The assumption of aquifers of lateral infinite extent is met, because drawdown and flow properties for the last model rings are all very close to zero.

Extraction from multi-aquifer system with recharge

The last example of steady state modeling concerns a multi-layer system with areal recharge in the top layer. The analytical solution for this problem is given by Bakker and Strack (2003) and it is obtained by superposition of two analytical elements: the first element is the solution for an extraction well in a multi-aquifer system, the second element is the solution for a circular infiltration area in a multi-aquifer system.

The solution for both elements is obtained by decomposing the transpose of system matrix *A* into its eigenvalues and eigenvectors, similar to the eigenvalue decomposition in previous example. The difference from previous example is that the multi-aquifer system is considered confined, meaning that c_1 and c_{n+1} are infinite. In this case, system matrix *A* has one zero eigenvalue and consequently, there are an infinite number of solutions. A unique solution can be obtained, for example, by specifying a constant head in one of the aquifers at a specified distance *R*.

A solution for the multi-aquifer well situated in aquifer *m* is given by:

$$s_i^W(r) = \frac{Q}{2\pi T}\ln(r) + \sum_{j=1}^{n-1} \frac{x_j v_{ij}}{2\pi T_i} K_0\left(r\sqrt{w_j}\right)$$

where $s_i^w(r)$ is the drawdown in aquifer *i* at distance *r*, *Q* is the well discharge, *T* is the comprehensive transmissivity which is equal to the sum of all aquifer transmissivities T_i, K_0 is the modified Bessel function of the second kind and zero order, w_j is the *j*-th non zero eigenvalue, and v_{ij} is the *i*-th element of the eigenvector corresponding to the *j*-th non zero eigenvalue. The *n-1* constants x_j are determined by solving the following set of *n-1* linear equations:

$$\sum_{j=1}^{n-1} x_j v_{ij} = \frac{QT_i}{T} \quad (i \neq m)$$

where *i* is the index of any aquifer that is not extracted.

A solution for the multi-aquifer circular infiltration area with flux *N* and radius R_c is given by:

$$s_i^c(r) = \frac{-N}{4T}\left(r^2 - R_c^2\right) + \sum_{j=1}^{n-1} \frac{y_j v_{ij}}{T_i}\left[\frac{1}{R_c\sqrt{w_j}} - K_1\left(R_c\sqrt{w_j}\right) I_0\left(r\sqrt{w_j}\right)\right] \quad (r \leq R_c)$$

$$s_i^c(r) = \frac{-NR_c^2}{2T}\ln(r/R_c) + \sum_{j=1}^{n-1} \frac{y_j v_{ij}}{T_i} I_1\left(R_c\sqrt{w_j}\right) K_0\left(r\sqrt{w_j}\right) \quad (r > R_c)$$

where $s_i^c(r)$ is the drawdown in aquifer *i* at distance *r*, I_0 and I_1 are the modified Bessel functions of the first kind and zero and first order respectively, and K_0 and K_1 are the modified Bessel functions of the second kind and zero and first order respectively. These functions are evaluated in MATLAB using the functions `besseli` and `besselk`. The *n-1* constants y_j are determined by solving the following set of *n-1* linear equations:

$$\sum_{j=1}^{n-1} \frac{y_j v_{ij}}{R_c} \sqrt{w_j} = \frac{-NT_i}{T} \quad (i \neq 1)$$

where *i* is the index of the different aquifers but the first.

The final solution for drawdown *s* is obtained by superposition of the two previous solutions plus a constant drawdown *C/T* with *C* an arbitrary constant:

$$s_i(r) = s_i^w(r) + s_i^c(r) + \frac{C}{T}$$

Constant *C* is determined, for example, by specifying drawdown equal to zero at distance $R \geq R_c$ in the first layer:

$$C = -[s_i^w(R) + s_i^c(R)]\,T$$

In this example, we consider 3 aquifers with T_1 = 50 m²/d, T_2 = 100 m²/d, and T_3 = 10 m²/d. The 2 aquitard resistances are c_1 = 100 d and c_2 = 250 d. An extraction well in the second aquifer with discharge *Q* = 500 m³/d, and a circular infiltration area with flux *N* = 5e-4 m/d and radius $R_c = (Q/N/\pi)^{1/2}$ are considered. Finally, the constant head in the first layer is assumed at a large distance *R* = 1e7 m to approximate a system of infinite extent. The implementation for this example is given in script "..\Examples\SteadyState\multiAquiferPlusRecharge.m":

```
% input
T = [50; 100; 10];
c = [100; 250];
Q = [0; 500; 0];
N = 5e-4;
Rc = sqrt(sum(Q)/pi/N);

% create Model object
m = MAxSym.Model;

% define steady state
m.settime(0);

% set grid
rb1 = logspace(-1,log10(Rc),41);
rb2 = logspace(log10(Rc),7,41);
m.setgrid([rb1,rb2(2:end)],ones(3,1),true);

% set parameters
m.par.constant = false(m.grid.nz,m.grid.nr);
m.par.constant(1,end) = true;
m.par.kr = T;
m.par.cz = c;
```

```
% set stresses
m.stress.q = zeros(m.grid.nz,m.grid.nr);
m.stress.q(:,1) = Q;
m.stress.q(1,1:40) = -N*m.grid.hs(1:40);

% set solver
m.setsolver(1e-5,100,5);

% run model
m.run;

% analytical solution
c = [Inf; c(:); Inf];
a = 1./T./c(1:end-1);
b = 1./T./c(2:end);
A = diag(a+b) - diag(a(2:end),1) - diag(b(1:end-1),-1);
[V,W] = eig(A);
W = diag(W);
[~,i] = min(W);
W(i) = [];
W = repmat(sqrt(W(:)'),3,1);
V(:,i) = [];
t = T/sum(T);

b = true(3,1);
[Q,i] = max(Q);
b(i) = false;
X = V(b,:)\(Q*t(b));
X = repmat(X',3,1);
Y = (V(2:end,:)/Rc.*W(2:end,:))\(-N*t(2:end));
Y = repmat(Y',3,1);

for i = 1:m.grid.nr
    x = X/2/pi .* besselk(0,m.grid.r(i)*W) .* V;
    sw(:,i) = (Q/2/pi * log(m.grid.r(i)) * t + sum(x,2)) ./ T;
    if m.grid.r(i) <= Rc
        y = Y.*V .* (1./W/Rc - ...
            besselk(1,Rc*W).*besseli(0,m.grid.r(i)*W));
        sc(:,i) = (-N/4 * (m.grid.r(i)^2-Rc^2)*t + sum(y,2)) ./ T;
    else
        y = Y.*V .* besseli(1,Rc*W).*besselk(0,m.grid.r(i)*W);
        sc(:,i) = (-N*Rc^2/2*log(m.grid.r(i)/Rc)*t + sum(y,2)) ./ T;
    end
end

s = sw + sc -(sw(1,end)+sc(1,end));

% distance-drawdown graph
figure
semilogx(m.grid.r,m.s','-')
hold on
semilogx(m.grid.r,s','x')
set(gca,'fontsize',12)
xlabel('distance (m)')
ylabel('drawdown (m)')
legend(strcat('layer',num2str((1:3)')));
```

The MAxSym model considers two cylindrical zones: the first extends from the inner grid radius of 0.1 m to the infiltration radius R_c, the second extends from the infiltration radius R_c to the outer grid radius which is set equal to R. Both zones are discretized into 40 rings. The last ring of the top layer is defined as constant-head ring, although this is not required here, because the outer grid boundary is at large distance, and consequently, it does not influence the result. Note that the analytical solution always requires a constant head to be defined, even if it is at large distance. Three layers are defined and each layer thickness is set 1, so aquifer transmissivities can be assigned directly to the `kr` property of the `Parameters` object. The aquitards are implemented by assigning the aquitard resistances to the `cz` property. Extraction well and infiltration area are defined by setting the `q` property of the `Stresses` object.

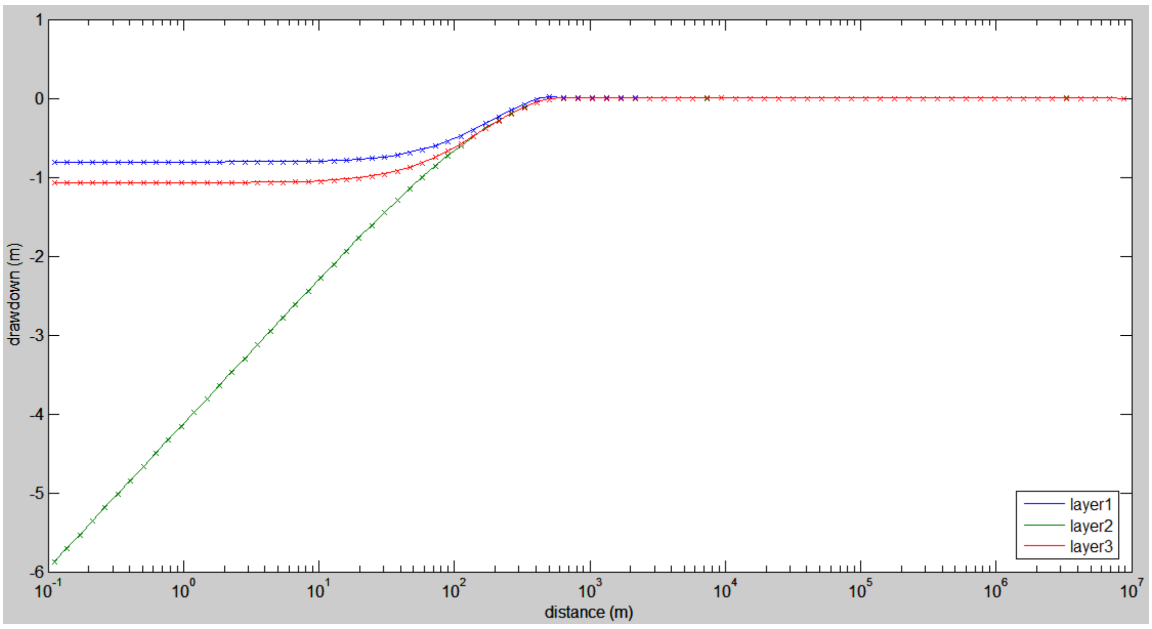


Figure 18. Plot of drawdown versus distance for a steady state multi-aquifer model with recharge. The solid line is calculated using MAxSym, the crosses correspond to the analytical solution (Bakker and Strack, 2003).

The resulting plot comparing the MAxSym solution to the analytical solution is shown in Figure 18. The SIP solver needed 96 iterations to solve the system, which is indicated by `m.niter`. Total volumetric budget `m.totbud(1)` is 0.6421 m^3/d. Setting a smaller criterion for convergence and a larger maximum number of iterations would result into smaller error on total budget.

The presented model may also be applied in case of a phreatic top layer if drawdown in the top layer is negligibly small compared to the layer thickness. Consider, for instance, the same aquifer system as above, but now, a phreatic top layer is defined with thickness 2 m. To implement this model and compare it with the analytical solution that assumes a constant saturated thickness in each layer, we run script "..\Examples\SteadyState\unconfinedMultiAquiferPlusRecharge.m ", which is obtained by copying the script given above and modifying the two statements indicated in red:

```
% set grid
rb1 = logspace(-1,log10(Rc),41);
rb2 = logspace(log10(Rc),7,41);
m.setgrid([rb1,rb2(2:end)],[2; ones(2,1)],false);
```

```
% set parameters
m.par.constant = false(m.grid.nz,m.grid.nr);
m.par.constant(1,end) = true;
m.par.kr = T./m.grid.D;
m.par.cz = c;
```

Figure 19 shows the resulting plot. It is seen that the unconfined MAxSym model calculates significantly larger drawdowns in the top layer, whereas drawdowns in the other layers are virtually the same.

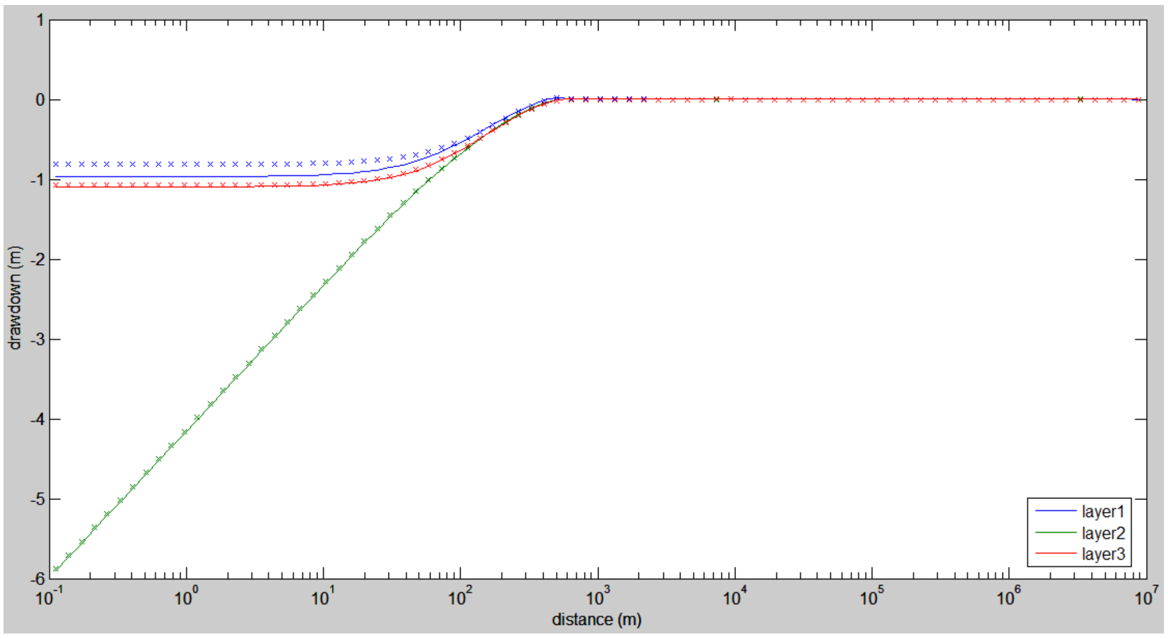


Figure 19. Plot of drawdown versus distance for a steady state, unconfined multi-layer model with recharge. The solid line is calculated using MAxSym, the crosses correspond to the analytical solution given by Bakker and Strack (2003). The latter assumes a constant saturated thickness for the top layer.

Transient models

Pumping test in confined aquifer

The first example of a transient model illustrates how to simulate radial flow towards a fully penetrating well in a confined aquifer of infinite lateral extent. This one-dimensional model is known as the Theis (1935) model. Drawdown calculated with MAxSym is compared to the analytical solution, which is given by:

$$s(r,t) = \frac{-Q}{4\pi T} W\left(\frac{r^2 S}{4tT}\right)$$

where *s* is the drawdown, *Q* is the well discharge, *T* is the transmissivity, *S* is the storativity, *r* is the radial distance, and *t* is the time. The function *W(u)* is known as the Well Function or Exponential Integral:

$$W(u) = \int_{u}^{\infty} e^{-y}\, dy/y$$

It can be evaluated using the MATLAB function `expint`.

In our example, the model parameters are *T* = 10 m^2/d, *S* = 1e-3, and *Q* = 100 m^3/d. Duration of the test is 1e4 days, which is subdivided in 90 time steps. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 80 rings are defined. The implementation for this example is given in script "..\Examples\Transient\pumpingTestConfined.m":

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,4,91)));

% set grid
m.setgrid(logspace(-1,7,81),1,true);

% set parameters
m.par.kr = 10;
m.par.ss = 1e-3;

% set stresses
m.stress.q = [100,zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,1);

% run model
m.run;

% analytical solution
ir = 1:10:31;
[r,t] = meshgrid(m.grid.r(ir),m.time.t);
u = r.^2./t * m.par.ss/m.par.kr/4;
s = -m.stress.q(1)/4/pi/m.par.kr * expint(u);

% time-drawdown graph
figure
semilogx(m.time.t,squeeze(m.s(1,ir,:))','-')
hold on
semilogx(m.time.t,s,'x')
xlabel('time (d)')
ylabel('drawdown (m)')
legend(strcat(num2str(m.grid.r(ir)','%.2f'),'m'))
```

Running the statements given above results in the time-drawdown graph shown in Figure 20. Note that the thickness of the aquifer is set 1. In this way, the horizontal conductivity and the specific elastic storage coefficient are equal to aquifer transmissivity *T* and storativity *S* respectively. The maximum number of iterations for the ADI solver is set 1, because the model is one-dimensional and linear, and hence, the solution is obtained directly without having to iterate.

Total volumetric budget for the entire model at the end of each time step is `m.totbud(:,1)`. The maximum total budget is obtained by evaluating `max(abs(m.totbud(:,1)))` which is equal to 3.7467e-12 m^3/d. Note that the extracted water is delivered exclusively by the aquifer storage, and hence, the sum of all storage decrease terms `abs(sum(squeeze(m.qs)))` is equal to the well discharge for each time step. Finally, it is verified if the boundary condition at infinity is satisfied:

```
>> min(squeeze(m.s(1,end,:)))

ans =

 -1.3031e-074

>> min(squeeze(m.qr(1,end-1,:)))

ans =

 -7.3322e-067
```

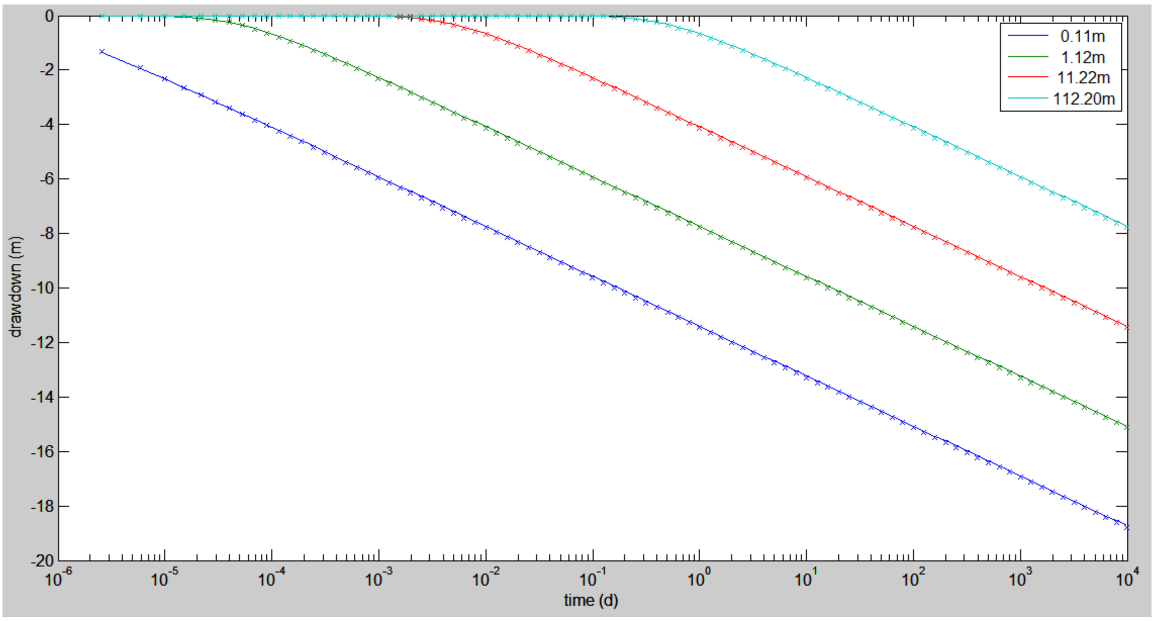


Figure 20. Time-drawdown graph for the Theis (1935) model calculated using MAxSym (solid lines) and the analytical expression (crosses).

When drawdown at arbitrary distances and times is required, bilinear interpolation must be applied. This is done using method `interp2` of class `Model`. The following statements calculate drawdown at 1 m, 5 m, and 10 m at times `1:100` days:

```
% interpolate drawdown
r = [1 5 10];
t = 1:100;
si = m.interp2(1,r,t);

% analytical solution
[t,r] = meshgrid(t,r);
u = r.^2./t * m.par.ss/m.par.kr/4;
s = -m.stress.q(1)/4/pi/m.par.kr * expint(u);

% plot interpolated drawdown vs time
figure
plot(t',squeeze(si)','k-',t',s','kx')
set(gca,'fontsize',12)
xlabel('time (d)')
```

```
    ylabel('drawdown (m)')
```

The resulting Figure 21 compares the interpolated drawdowns to those calculated analytically. The small deviation between both results could be reduced by defining a finer discretization of radial distance and time.

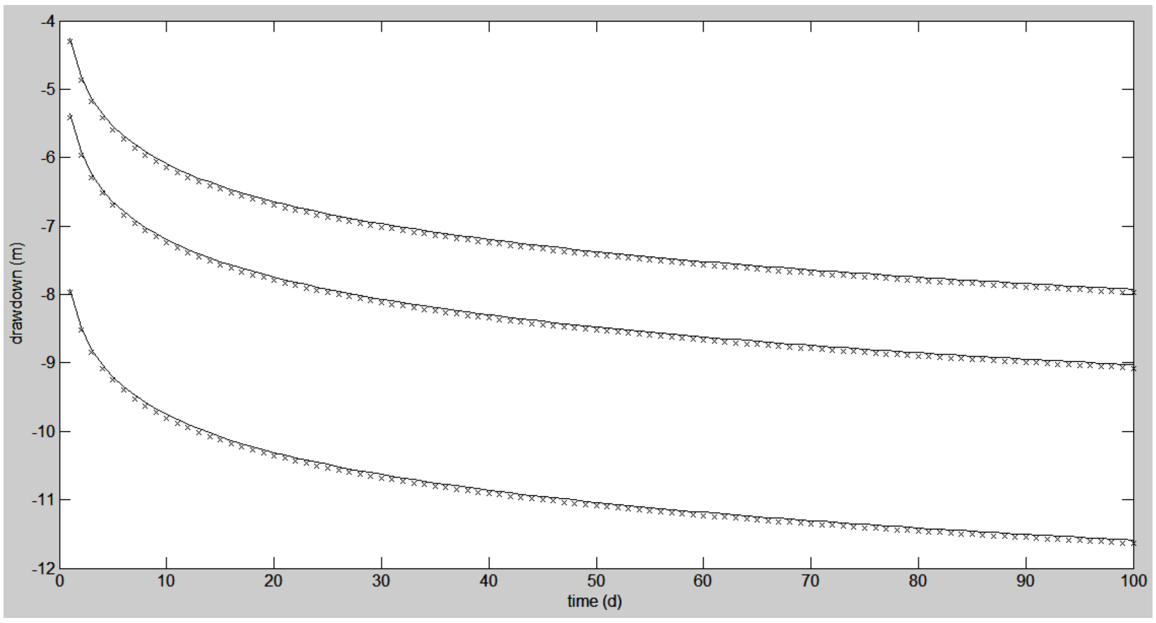


Figure 21. Plot of interpolated drawdown (solid lines) versus time at distances equal to 1 m, 5m, and 10 m. Drawdown calculated analytically is indicated by crosses.

Pumping test in well with large diameter

The Theis (1935) model assumes the well has an infinitesimal small diameter. In practice, this assumption is only valid if the well diameter is sufficiently small so that wellbore storage may be neglected. If the well diameter is large, however, then the applied model must account for the effect of wellbore storage. The analytical solution for a fully penetrating well of large diameter located extracting a confined, homogeneous aquifer of infinite later extent is given by Papadopulos and Cooper (1967):

$$s(r,t) = \frac{-2QS}{\pi^2 T} \int_0^\infty \left(1 - e^{-tTy^2/r_w^2 S}\right) G(r,y)\, dy / (y^2 F(y))$$

where *s* is the drawdown, *r* is the radial distance, *t* is the time, *Q* is the well discharge, r_w is the well radius, *T* is the aquifer transmissivity, and *S* is the aquifer storativity. Functions *G(r,y)* and *F(y)* are defined as:

$$G(r,y) = J_0\left(\frac{yr}{r_w}\right)[yY_0(y) - 2SY_1(y)] - Y_0\left(\frac{yr}{r_w}\right)[yJ_0(y) - 2SJ_1(y)]$$

$$F(y) = [yJ_0(y) - 2SJ_1(y)]^2 + [yY_0(y) - 2SY_1(y)]^2$$

J_0 and J_1 denote Bessel functions of the first kind of zero and first order respectively, and they are evaluated using the MATLAB function `besselj`. Y_0 and Y_1 are Bessel functions of the second kind of zero and first order respectively, and they are evaluated using the MATLAB function `bessely`. The infinite integral can be evaluated numerically using the MATLAB function `quadgk`, which applies the adaptive Gauss-Kronrod quadrature (Shampine, 2008).

Consider the same problem as in previous example with parameters *T* = 10 m²/d, *S* = 1e-3, and *Q* = 100 m³/d, but now, a well radius of 0.5 m is assumed. The inner grid radius is derived from the well radius, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 800 rings are defined to ensure an accurate simulation of drawdown within the well. Duration of the test is 1e4 days, which is subdivided in 90 time steps.

The corresponding script “..\Examples\Transient\pumpingTestLargeWell.m” is printed below:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,4,91)));

% set grid
rb = logspace(log10(0.5),7,801);
rb = [rb(1)^2/rb(2), rb];
m.setgrid(rb,1,true);

% set parameters
m.par.kr = 10;
m.par.ss = [m.grid.rb(2)^2*pi/m.grid.vol(1), ...
            1e-3*ones(1,m.grid.nr-1)];

% set stresses
m.stress.q = [100, zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,1);

% run model with wellbore storage
m.run;
sw = m.s;

% run model without wellbore storage
m.par.ss(1) = m.par.ss(2);
m.run;

% analytical solution
rw = m.grid.rb(2);
ir = 101:100:301;
r = [rw m.grid.r(ir)];

F = @(y) (y.*besselj(0,y)-2*m.par.ss(end)*besselj(1,y)).^2 + ...
         (y.*bessely(0,y)-2*m.par.ss(end)*bessely(1,y)).^2;

G = @(r,y) besselj(0,y*r/rw).*(y.*bessely(0,y)-...
                               2*m.par.ss(end)*bessely(1,y)) - ...
           bessely(0,y*r/rw).*(y.*besselj(0,y)-...
                               2*m.par.ss(end)*besselj(1,y));

H = @(r,y,t) (1-exp(-m.par.kr*t.*y.*y/rw^2/m.par.ss(end))) .* ...
             G(r,y)./y.^2./F(y);
```

```
for i = 1:length(r)
    for k = 1:length(m.time.t)
        s(i,k) = -2*m.stress.q(1)*m.par.ss(end)/pi/pi/m.par.kr * ...
                 quadgk(@(y)H(r(i),y,m.time.t(k)),0,Inf, ...
                        'MaxIntervalCount',1500,'AbsTol',1e-5);
    end
end

% time-drawdown graph
figure
semilogx(m.time.t,squeeze(sw(1,[1 ir],:))','-')
hold on
semilogx(m.time.t,squeeze(m.s(1,[1 ir],:))',':')
semilogx(m.time.t,s','x')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
legend(strcat(num2str(r','%.2f'),'m'))
```

The outer radius of the first ring is set equal to the well radius. The width of the first ring must be sufficiently small because the drawdown in this ring approximates the drawdown at the face of the well screen. This explains why the radial extent of the model grid is more discretized than in previous example. Wellbore storage is taken into account by setting the specific storage coefficient of the first ring equal to $\pi r_w^2/V(1)$ with *V(1)* the volume of the first ring. Dividing by the volume is required because the specific storage coefficients are multiplied by the ring volumes when storage change is calculated.

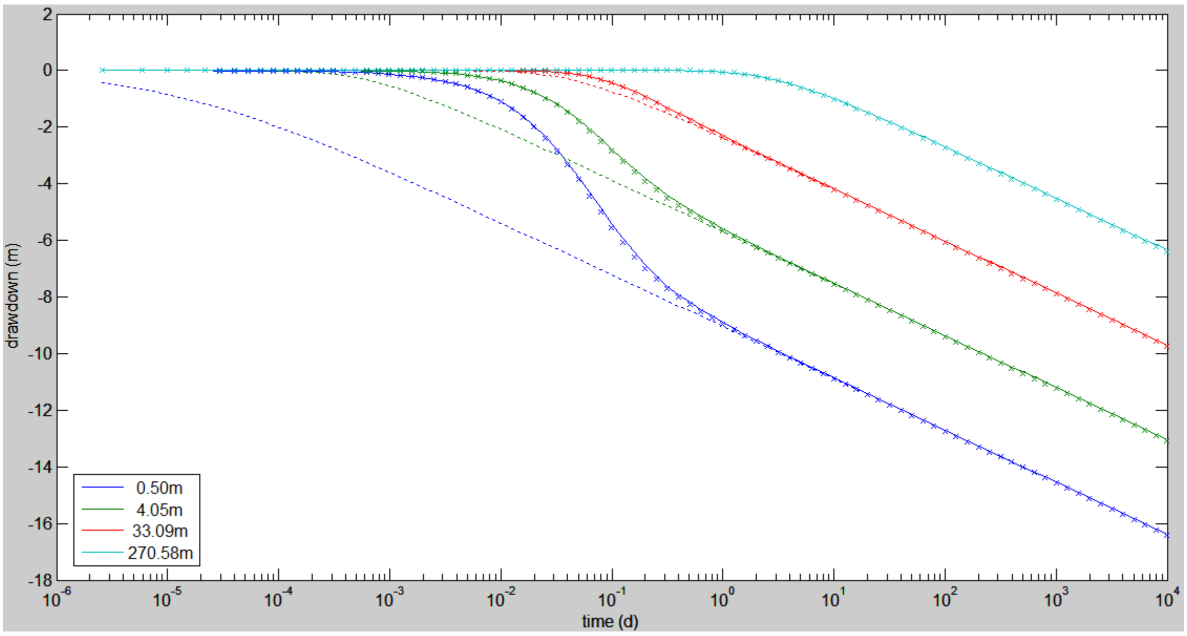


Figure 22. Time-drawdown graph for the pumping test conducted in a well of large diameter. Drawdown is calculated using MAxSym (solid lines) and the analytical expression (crosses) given by Papadopulos and Cooper (1967). The dotted lines indicate the corresponding Theis (1935) solution neglecting wellbore storage, which is calculated using MAxSym.

Figure 22 shows the resulting time-drawdown graphs for the case where wellbore storage is included and the case where wellbore storage is neglected. The model considering wellbore storage is still one-dimensional and linear, hence, the Thomas algorithm can be applied to solve the system directly. This is done by choosing the ADI solver and setting `mni` equal to 1. The maximum absolute value for the total budget `max(abs(m.totbud(:,1)))` is 2.7388e-10 m³/d.

Push-pull test in confined aquifer

In general, a variable discharge test in a confined, homogeneous layer with fully penetrating well is simulated analytically using the Theis (1935) formula and applying the superposition principle. If *n* discharge periods are considered with the pumping rate constant and equal to *Q(i)* during period *i* starting at $t^Q(i-1)$ and ending at $t^Q(i)$, then drawdown *s* at distance *r* and time *t* is calculated as (see, for instance, Lebbe, 1999):

$$s(r,t) = \sum_{i=1}^{n} [Q(i) - Q(i-1)] \; s_u(r, t - t^Q_{(i-1)})$$

where both $t^Q(0)$ and *Q(0)* are zero, and $s_u(r,t)$ is the drawdown at distance *r* and time *t* due to unit discharge, which is calculated using the Theis formula with *Q = 1*:

$$s_u(r,t) = \frac{-1}{4\pi T} W\left(\frac{r^2 S}{4tT}\right)$$

where *T* and *S* are aquifer transmissivity and storativity respectively, and *W(u)* is the Well function.

A first example of a variable discharge test is the push-pull test, which consists of two phases: an injection or “push” phase, followed by an extraction or “pull” phase. Consider a test in an aquifer with transmissivity *T* = 10 m²/d and storativity *S* = 1e-3. The push phase lasts 1 day and water is injected with *Q* = -100 m³/d. The pull phase also lasts 1 day and water is extracted with *Q* = 100 m³/d.

To simulate the test with MAxSym, two stress periods are defined, both subdivided in 50 time steps. The discretization for each stress period is the same to keep superposition straightforward. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 80 rings are defined. Script “..\Examples\Transient\pushpull.m” implements this example:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(repmat(diff(logspace(-5,0,51)),[1 2]),50*ones(1,2));

% set grid
m.setgrid(logspace(-1,7,81),1,true);

% set parameters
m.par.kr = 10;
m.par.ss = 1e-3;

% set stresses
m.stress(1).q = [-100, zeros(1,m.grid.nr-1)];
m.stress(2).q = [100, zeros(1,m.grid.nr-1)];
```

```
% set solver
m.setsolver(1e-5,1);

% run model
m.run;

% analytical solution
ir = 1:10:31;
[r,t] = meshgrid(m.grid.r(ir),m.time.t);
u = r.^2./t * m.par.ss/m.par.kr/4;
s1 = -1/4/pi/m.par.kr * expint(u);
s1(1) = 0;
ndt = [0; cumsum(m.time.ndt(:))];
dq = diff([0; sum(cat(1,m.stress.q),2)]);
s = zeros(size(s1));
for n = 1:length(ndt)-1
    s(ndt(n)+1:end,:) = s(ndt(n)+1:end,:) + dq(n)*s1(1:end-ndt(n),:);
end

% time-drawdown graph
figure
plot(m.time.t,squeeze(m.s(1,ir,:))','-')
hold on
plot(m.time.t,s,'x')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
legend(strcat(num2str(m.grid.r(ir)','%.2f'),'m'))
```

The resulting time-drawdown graph is shown in Figure 23. The model is still one-dimensional and linear, hence, the Thomas algorithm can be applied to solve the system directly. This is done by choosing the ADI solver and setting `mni` equal to 1. The maximum absolute value for the total budget `max(abs(m.totbud(:,1)))` is 2.3073e-9 m^3/d.

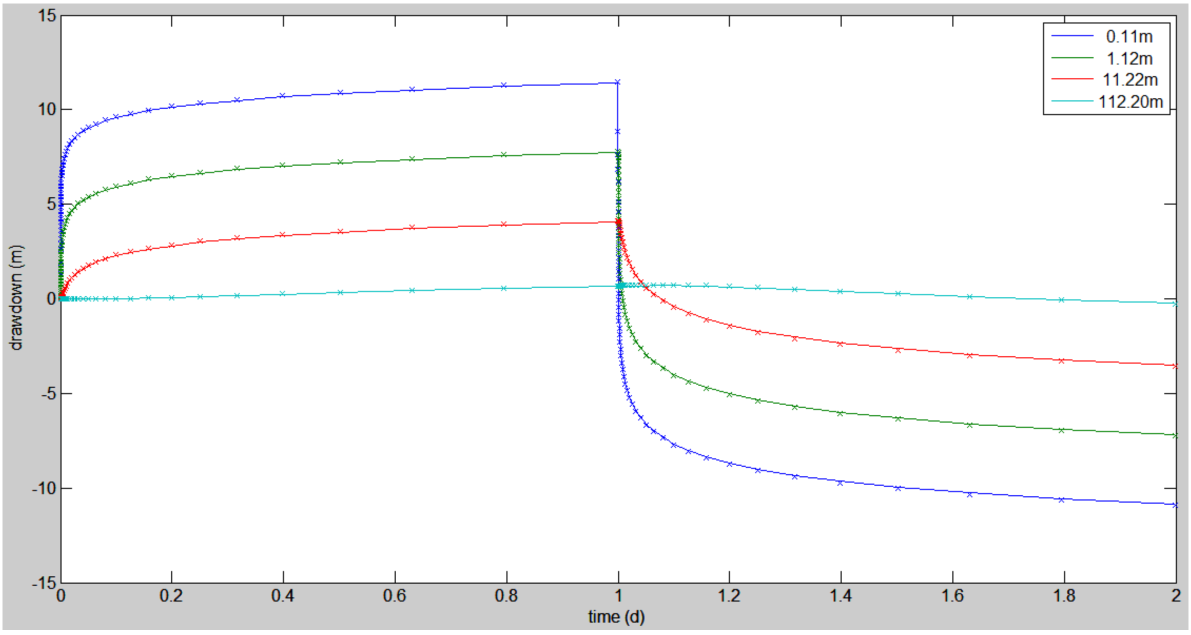


Figure 23. Time-drawdown graph for the push-pull test calculated using MAxSym (solid lines) and the analytical expression (crosses).

Recovery test in confined aquifer

Simulation of a pumping test followed by a recovery phase is done in the same way as the push-pull test simulated in previous example. As an example, the same confined aquifer is considered with transmissivity *T* = 10 m^2/d and storativity *S* = 1e-3. The pumping test lasts 1 day and water is extracted with *Q* = 100 m^3/d, the subsequent recovery phase also lasts 1 day. Again, two stress periods are defined, both subdivided in 50 time steps. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 80 rings are defined. Script "..\Examples\Transient\recovery.m" implements this example:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(repmat(diff(logspace(-5,0,51)),[1 2]),50*ones(1,2));

% set grid
m.setgrid(logspace(-1,7,81),1,true);

% set parameters
m.par.kr = 10;
m.par.ss = 1e-3;

% set stresses
m.stress(1).q = [100, zeros(1,m.grid.nr-1)];
m.stress(2).q = zeros(1,m.grid.nr);

% set solver
m.setsolver(1e-5,1);

% run model
m.run;

% analytical solution
ir = 1:10:31;
[r,t] = meshgrid(m.grid.r(ir),m.time.t);
u = r.^2./t * m.par.ss/m.par.kr/4;
s1 = -1/4/pi/m.par.kr * expint(u);
s1(1) = 0;
ndt = [0; cumsum(m.time.ndt(:))];
dq = diff([0; sum(cat(1,m.stress.q),2)]);
s = zeros(size(s1));
for n = 1:length(ndt)-1
    s(ndt(n)+1:end,:) = s(ndt(n)+1:end,:) + dq(n)*s1(1:end-ndt(n),:);
end

% time-drawdown graph
figure
plot(m.time.t,squeeze(m.s(1,ir,:))','-')
hold on
plot(m.time.t,s,'x')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
legend(strcat(num2str(m.grid.r(ir)','%.2f'),'m'))
```

The resulting time-drawdown graph is shown in Figure 24. Note that it is not required to set the discharge matrix for the second stress period, but it was done here to facilitate the superposition of the analytical solutions. The system is solved directly using ADI and setting `mni` equal to 1. The maximum absolute value for the total budget `max(abs(m.totbud(:,1)))` is 2.2830e-9 m^3/d.

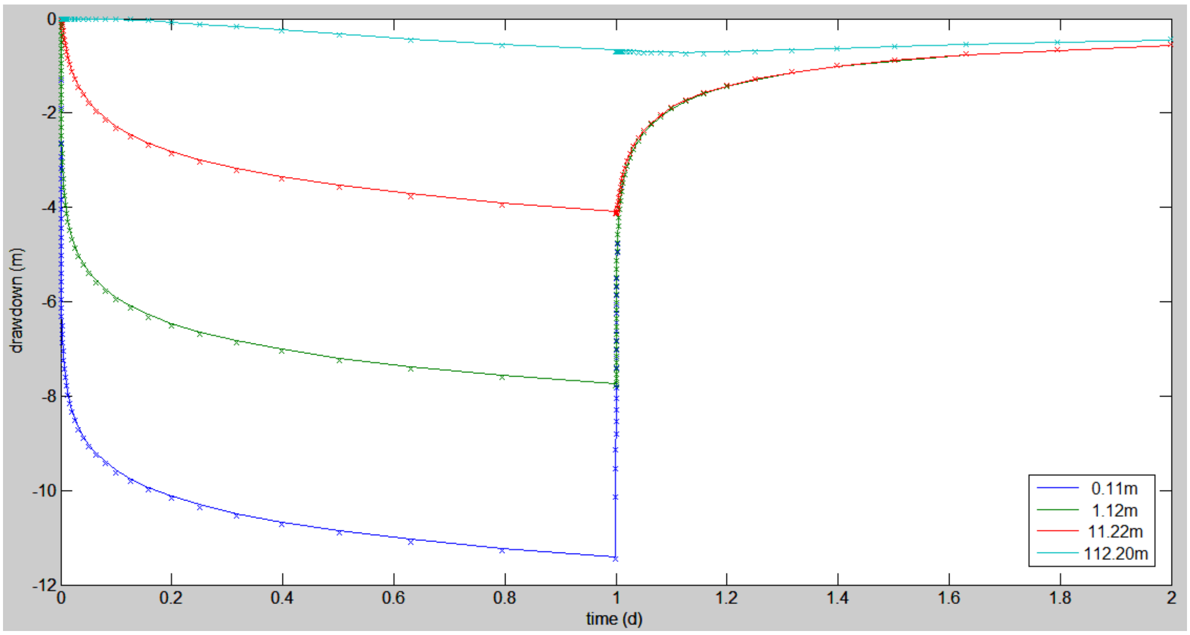


Figure 24. Time-drawdown graph for the recovery test calculated using MAxSym (solid lines) and the analytical expression (crosses).

Recovery test with non-recoverable compression

In some cases, the pumping may cause a significant decrease of the storage coefficient (see, for instance, Helm, 1976). This effect of non-recoverable compression, however, cannot be simulated applying superposition. Using MAxSym, it is simulated by first running the pumping test model and then set up a model for the recovery phase in which the starting drawdowns are adopted from the first model.

We consider the same pumping test as in previous example: T = 10 m^2/d, S = 1e-3, Q = 100 m^3/d, and duration of the test is 1 day which is subdivided into 50 time steps. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 80 rings are defined. Script "..\Examples\Transient\nonrecoverable.m" implements this example:

```
% PUMPING test

% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,0,51)));

% set grid
m.setgrid(logspace(-1,7,81),1,true);

% set parameters
m.par.kr = 10;
```

```
m.par.ss = 1e-3;

% set stresses
m.stress.q = [100, zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,1);

% run model
m.run;

% get drawdown matrix
s = squeeze(m.s);
```

For the recovery phase, we consider three cases: 1) the storage coefficient is the same, 2) the storage coefficient is 10 times smaller, and 3) the storage coefficient is 100 times smaller:

```
% RECOVERY phase

% modify stresses
m.stress.q = [];
m.stress.s0 = s(:,end)';

% loop through different Ss values
ss = 10.^(-5:-3);
for n = 1:length(ss)

    % set ss
    m.par.ss = ss(n);

    % run model
    m.run

    % get drawdown
    sr(:,:,n) = squeeze(m.s);

end

% time-drawdown graph
figure
ir = 11;
t = m.time.t(:);
s = cat(2,repmat(s,[1 1 length(ss)]),sr(:,2:end,:));
plot([t; t(end)+t(2:end)],squeeze(s(ir,:,:)))
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
title(strcat(num2str(m.grid.r(ir)),'m'));
legend(num2str(ss','%.0e'))
```

Figure 25 shows the resulting time-drawdown graphs for the 3 scenarios. A new `Model` object could have been created for the recovery phase model, but instead, the original object `m` was modified because the changes are limited: only the discharge matrix `q` must be cleared and the initial drawdown matrix `s0` must be set equal to the drawdown matrix from the last time step in the pumping test model. Modifying the storage coefficient is done during each loop iteration and the drawdown arrays from each model are stored in a new variable. Finally, the drawdown arrays from pumping and recovery models are concatenated to plot the drawdown graphs.

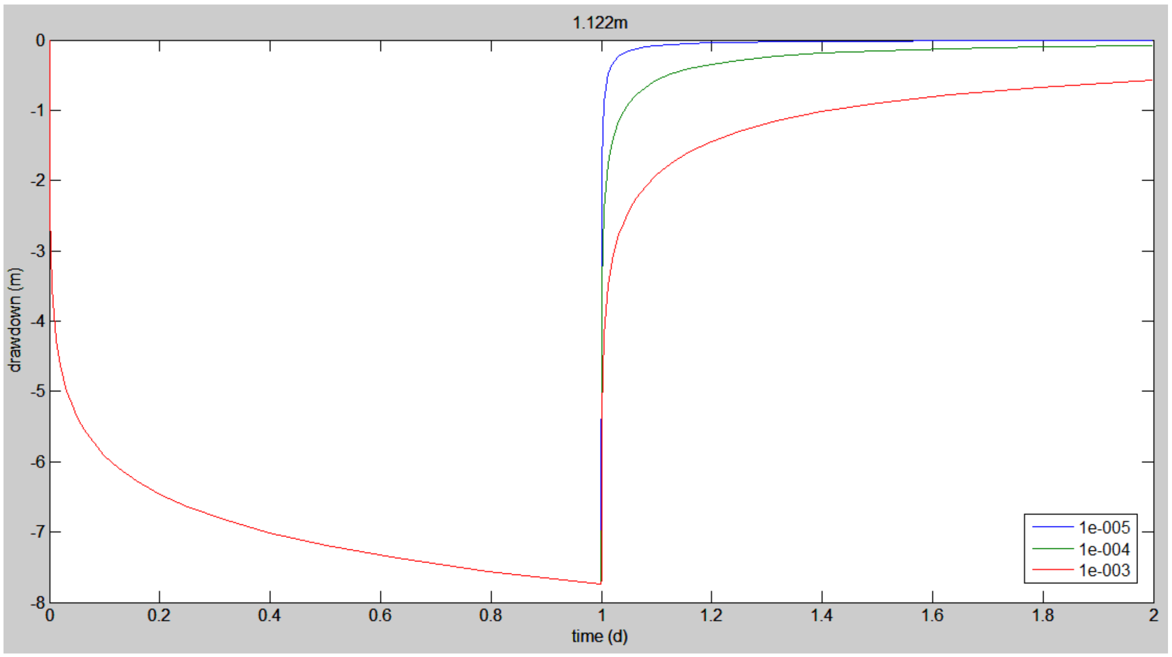


Figure 25. Time-drawdown graph at r = 1.122 m for the recovery test considering 3 scenarios for storage coefficient correction: no correction (red), 10 x smaller coefficient (green), and 100 x smaller coefficient (blue).

Step-drawdown test in confined aquifer

Another example of a variable discharge test is the step-drawdown test. It consists of several steps during which the discharge is increased. If the test is conducted in a confined, homogeneous aquifer, it is simulated analytically using the Theis (1935) formula and applying the superposition principle as explained above.

In this example, a step-drawdown test is simulated in an aquifer with transmissivity T = 10 m^2/d and storativity S = 1e-3. Four steps are considered, corresponding to 4 stress periods of 1 day with *Q(1)* = 100 m^3/d, *Q(2)* = 200 m^3/d, *Q(3)* = 300 m^3/d, and *Q(4)* = 400 m^3/d. Each stress period is subdivided in 50 time steps. Note the discretization for each stress period is the same to facilitate superposition. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 80 rings are defined. Script "..\Examples\Transient\stepTestConfined.m" implements this example and the resulting time-drawdown graph is shown in Figure 26:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(repmat(diff(logspace(-5,0,51)),[1 4]),50*ones(1,4));

% set grid
m.setgrid(logspace(-1,7,81),1,true);

% set parameters
m.par.kr = 10;
m.par.ss = 1e-3;
```

```
% set stresses
m.stress(1).q = [100, zeros(1,m.grid.nr-1)];
m.stress(2).q = [200, zeros(1,m.grid.nr-1)];
m.stress(3).q = [300, zeros(1,m.grid.nr-1)];
m.stress(4).q = [400, zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,1);

% run model
m.run;

% analytical solution
ir = 1:10:31;
[r,t] = meshgrid(m.grid.r(ir),m.time.t);
u = r.^2./t * m.par.ss/m.par.kr/4;
s1 = -1/4/pi/m.par.kr * expint(u);
s1(1) = 0;
ndt = [0; cumsum(m.time.ndt(:))];
dq = diff([0; sum(cat(1,m.stress.q),2)]);
s = zeros(size(s1));
for n = 1:length(ndt)-1
    s(ndt(n)+1:end,:) = s(ndt(n)+1:end,:) + dq(n)*s1(1:end-ndt(n),:);
end

% time-drawdown graph
figure
plot(m.time.t,squeeze(m.s(1,ir,:))','-')
hold on
plot(m.time.t,s,'x')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
legend(strcat(num2str(m.grid.r(ir)','%.2f'),'m'))
```

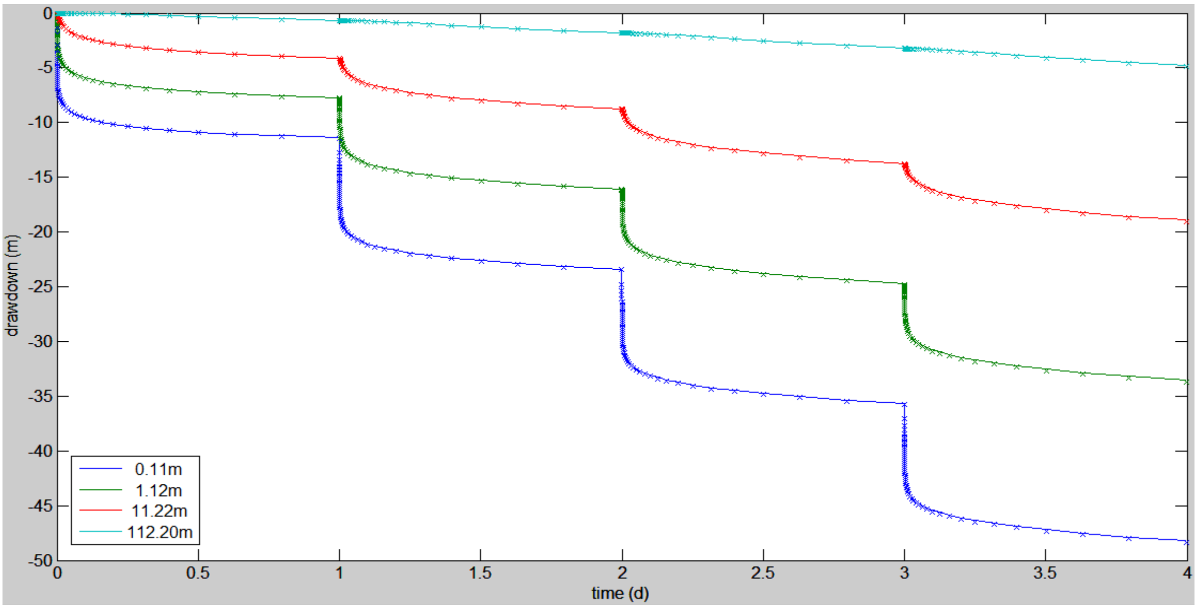


Figure 26. Time-drawdown graph for the step-drawdown test conducted in a confined aquifer calculated using MAxSym (solid lines) and the analytical expression (crosses).

Step-drawdown test in well with infinitesimal skin

In this example, a step-drawdown test is simulated, conducted in a fully penetrating well with infinitesimal skin located in a confined, homogeneous layer. The skin causes a supplementary drawdown *Δs* within the well equal to (Kruseman and de Ridder, 1990):

$$\Delta s = \frac{-C^r Q}{2\pi r_w L}$$

where C^r is the skin resistance, *Q* is the well discharge, r_w is the well radius, and *L* is the screen length. Total drawdown within the well is:

$$s(r_w, t) = s_{aq}(r_w, t) + \Delta s$$

where s_{aq} is the drawdown due to aquifer loss, which is calculated using the Theis (1935) formula and applying superposition if the discharge is variable. Note that it is assumed drawdown within the well is equal to drawdown at the face of the well screen, hence, radial distance is equal to the well radius.

We adopt the same model as in previous example: *T* = 10 m²/d, *S* = 1e-3, and 4 stress periods of 1 day are considered with *Q(1)* = 100 m³/d, *Q(2)* = 200 m³/d, *Q(3)* = 300 m³/d, and *Q(4)* = 400 m³/d. Additionally, an infinitesimal well skin is defined with resistance C^r equal to 0.01 d. The well radius r_w is 0.1 m. Each stress period is subdivided in 50 time steps. The layer thickness is set 1 m. The inner grid radius is derived from the well radius, the outer grid radius approximates the boundary at infinity and is set 1e7 m. To obtain an accurate simulation of drawdown within the well, 801 rings are defined. Script "..\Examples\Transient\stepTestInfSkin.m" implements this example:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(repmat(diff(logspace(-5,0,51)),[1 4]),50*ones(1,4));

% set grid
rb = logspace(-1,7,801);
m.setgrid([rb(1)^2/rb(2), rb],1,true);

% set parameters
m.par.kr = 10;
m.par.ss = 1e-3;
m.par.cr = sparse(1,1,1e-2,1,m.grid.nr-1);

% set stresses
m.stress(1).q = sparse(1,1,100,1,m.grid.nr);
m.stress(2).q = sparse(1,1,200,1,m.grid.nr);
m.stress(3).q = sparse(1,1,300,1,m.grid.nr);
m.stress(4).q = sparse(1,1,400,1,m.grid.nr);

% set solver
m.setsolver(1e-5,1);

% run model
m.run;
```

```
% analytical solution
r = [m.grid.rb(2), m.grid.r(101:100:301)];
[r,t] = meshgrid(r,m.time.t);
u = r.^2./t * m.par.ss/m.par.kr/4;
s1 = -1/4/pi/m.par.kr * expint(u);
s1(1) = 0;
ndt = [0; cumsum(m.time.ndt(:))];
dq = diff([0; sum(cat(1,m.stress.q),2)]);
s = zeros(size(s1));
for n = 1:length(ndt)-1
    s(ndt(n)+1:end,:) = s(ndt(n)+1:end,:) + dq(n)*s1(1:end-ndt(n),:);
    ds(:,n) = m.par.cr(1)*m.stress(n).q(1)/2/pi/m.grid.rb(2);
    s(ndt(n)+2:ndt(n+1)+1,1) = s(ndt(n)+2:ndt(n+1)+1,1) - ds(:,n);
end

% time-drawdown graph
figure
plot(m.time.t,squeeze(m.s(1,1:100:301,:))','-')
hold on
plot(m.time.t,s,'x')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
legend(strcat(num2str(r(1,:)','%.2f'),'m'))
```

Defining the skin in the MAxSym model is done by setting the `cr` property in the `Parameters` object. The none zero entry in `cr` overwrites the corresponding `kr` values for the first two rings. Note that `cr` as well as `q` contain sparse matrices to save memory. The well is implemented by setting the outer radius of the first ring equal to the well radius. The latter is also used to calculate drawdown within the well analytically. Figure 27 shows the resulting time-drawdown graph.

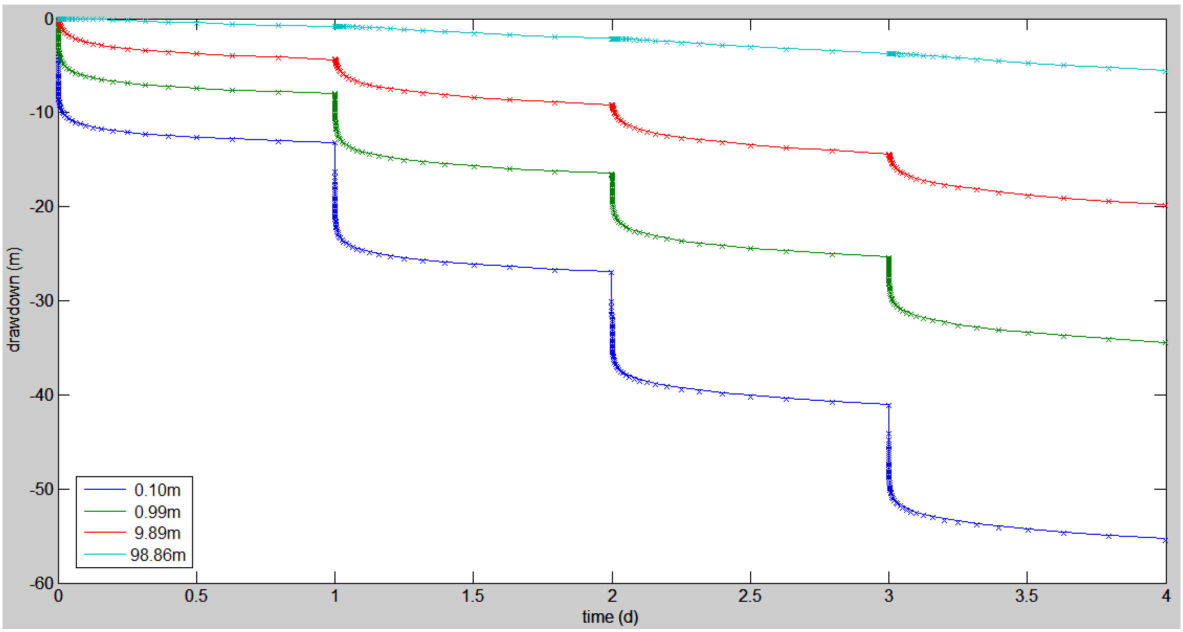


Figure 27. Time-drawdown graph for the step-drawdown test conducted in a well with infinitesimal skin. Drawdown is calculated analytically (crosses) and using MAxSym (solid lines). The curve for 0.1 m corresponds to the drawdown within the well.

Step-drawdown test in well with non-linear well loss

The simulated skin effect in previous example is also referred to as linear well loss. Non-linear well loss related to non-Darcyan flow is defined as (Kruseman and de Ridder, 1990):

$$\Delta s = -CQ^n$$

where *C* is the non-linear well loss coefficient and *n* is the well loss power, which is a constant greater than 1. In most cases *n* is assumed equal to 2. Non-linear well loss cannot be taken into account by setting the `cr` or any other MAxSym property. Therefore, the supplementary drawdown *Δs* is added to the drawdown within the well after it was calculated using MAxSym:

$$s(r_w, t) = s_{aq}(r_w, t) + \Delta s$$

where $s(r_w,t)$ is the total drawdown within the well and $s_{aq}(r_w,t)$ is the drawdown due to aquifer loss, which is calculated using MAxSym. The effect of linear-well loss due to skin effects may also be considered when calculating s_{aq}. Script "..\Examples\Transient\stepTestWellLoss.m" simulates the same model as in previous example, but now, a non-linear well loss with *C* = 1e-5 and *n* = 2 is defined instead of an infinitesimal skin:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(repmat(diff(logspace(-5,0,51)),[1 4]),50*ones(1,4));

% set grid
rb = logspace(-1,7,801);
m.setgrid([rb(1)^2/rb(2),rb],1,true);

% set parameters
m.par.kr = 10;
m.par.ss = 1e-3;

% set stresses
m.stress(1).q = sparse(1,1,100,1,m.grid.nr);
m.stress(2).q = sparse(1,1,200,1,m.grid.nr);
m.stress(3).q = sparse(1,1,300,1,m.grid.nr);
m.stress(4).q = sparse(1,1,400,1,m.grid.nr);

% set solver
m.setsolver(1e-5,10);

% run model
m.run;
```

```
% add well loss
sw = squeeze(m.s(1,1,:));
ndt = [0; cumsum(m.time.ndt(:))];
dq = diff([0; sum(cat(1,m.stress.q),2)]);
for n = 1:length(ndt)-1
    sw(ndt(n)+2:ndt(n+1)+1) = sw(ndt(n)+2:ndt(n+1)+1) - ...
                              1e-5*m.stress(n).q(1)^2;
end
```

```
% time-drawdown graph
figure
plot(m.time.t,squeeze(m.s(1,1,:))','k-',m.time.t,sw,'r-')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
legend('not corrected','corrected')
```

The resulting time-drawdown graph for the pumping well is shown in Figure 28.

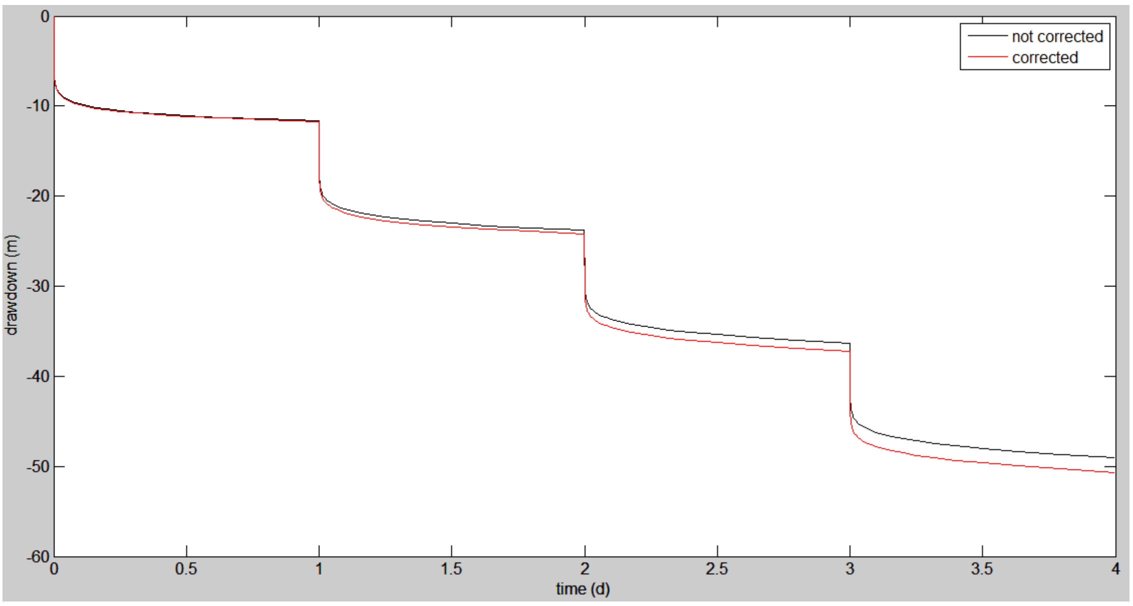


Figure 28. Drawdown within the well for the step-drawdown test conducted in a well with non-linear well loss. The black line shows drawdown due to aquifer loss only, the red line shows drawdown corrected for non-linear well loss.

Pumping test in well with finite-thickness skin

The effect of a finite-thickness skin is simulated by defining a cylindrical zone of radius *R* around the pumping well. If a confined, homogeneous aquifer of infinite lateral extent with fully penetrating well is considered, then drawdown *s* can be calculated analytically using the solution given by Butler (1988):

$$s(r,t) = \frac{Q}{4\pi T_2}\left[\gamma + \ln\left(\frac{R^2 S_2}{4tT_2}\right)\right] + \frac{Q}{2\pi T_1}\ln\left(\frac{r}{R}\right) \qquad (r \le R)$$

$$s(r,t) = \frac{Q}{4\pi T_2}\left[\gamma + \ln\left(\frac{r^2 S_2}{4tT_2}\right)\right] \qquad (r > R)$$

where T_1 is the transmissivity of the cylindrical zone, T_2 and S_2 are the aquifer transmissivity and storativity respectively, *Q* is the pumping rate, *γ* is Euler's constant equal to 0.5772..., *r* is the radial distance, and *t* is the time. The given solution is approximate and valid for large times only. Note that the second term in the right-hand side of the first equation corresponds to the supplementary drawdown caused by the finite-thickness skin. As is the case for an infinitesimal skin, the supplementary drawdown is a linear function of the discharge.

In this example, the skin radius *R* is 1 m, the skin transmissivity T_1 is 1 m²/d, the skin storativity S_1 is 1e-5, the aquifer transmissivity T_2 is 10 m²/d, the aquifer storativity S_2 is 1e-3, and the well discharge *Q* is 100 m³/d. Duration of the test is 1e4 days, which is subdivided in 90 time steps. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m. There are 80 rings defined and the first 10 rings represent the well skin. The following statements implementing the example are found in "..\Examples\Transient\pumpTestFinThickSkin.m":

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,4,91)));

% set grid
m.setgrid(logspace(-1,7,81),1,true);

% set parameters
iR = 10;
m.par.kr = [ones(1,iR) 10*ones(1,m.grid.nr-iR)];
m.par.ss = [1e-5*ones(1,iR) 1e-3*ones(1,m.grid.nr-iR)];

% set stresses
m.stress.q = [100, zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,1);

% run model
m.run;

% analytical solution
it = 51:10:91;
[r,t] = meshgrid(m.grid.r,m.time.t(it));
R = m.grid.rb(iR+1);
b = r <= R;
s = zeros(size(r));
s(b) = m.stress.q(1)/2/pi * ...
       ((0.5772 + log(m.par.ss(end)*R*R/4/m.par.kr(end)./t(b)))...
        /2/m.par.kr(end) + log(r(b)/R)/m.par.kr(1));
b = ~b;
s(b) = m.stress.q(1)/4/pi/m.par.kr(end) * ...
       (0.5772 + log(m.par.ss(end)*r(b).^2/4/m.par.kr(end)./t(b)));
s(s>0) = 0;

% distance-drawdown graph
figure
semilogx(m.grid.r,squeeze(m.s(1,:,it))','-')
hold on
semilogx(m.grid.r,s','x')
set(gca,'fontsize',12)
xlabel('distance (m)')
ylabel('drawdown (m)')
xlim([1e-1 1e5])
ylim([-51 0])
legend(strcat(num2str(m.time.t(it),'%.0f'),'d'))
```

Figure 29 shows the resulting distance-drawdown graph for different times. Note that the system of equations can be solved directly by using the ADI solver and setting the maximum number of iterations equal to 1. Property `m.totbud(:,1)` gives the volumetric budget for the entire model at the end of each time step. The outer model boundary condition is verified by checking drawdown in the last ring: `squeeze(m.s(1,end,:))`.

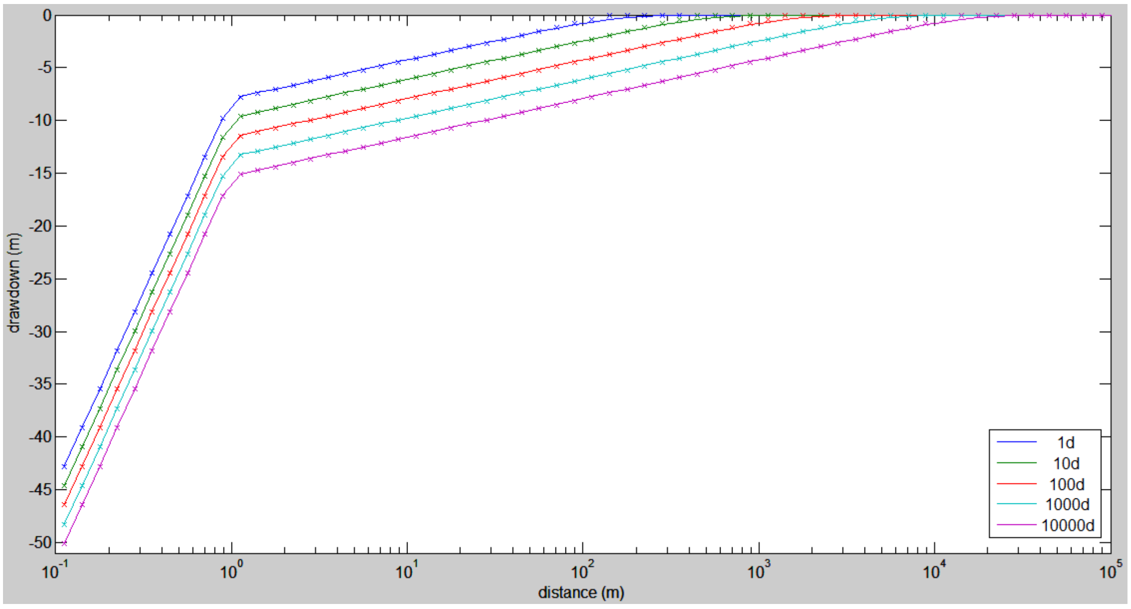


Figure 29. Distance-drawdown graph for the pumping test conducted in a well with finite-thickness skin. Drawdown is calculated using MAxSym (solid lines) and the analytical expression (crosses) given by Butler (1988).

Pumping test in unconfined aquifer

If the aquifer is unconfined, the water table response must be taken into account by defining a specific yield parameter. The analytical solution to the problem of an extracted well fully penetrating an unconfined, homogeneous, vertical anisotropic layer is presented by Neuman (1972). In this example, however, only the MAxSym code is given, because the Neuman solution is much more involved than the analytical solutions given in this manual. Besides, defining a specific yield parameter to account for the amount of water yielded by the water table movement is a simplified way to do, although it is common practice for finite-difference models and for some (semi) analytical models.

Therefore, care should be taken when MAxSym is used to simulate unconfined flow, especially when the phreatic layer is extracted. At this moment, MAxSym only allows the upper layer to be phreatic and model runs are aborted once the water table is below the bottom of the top layer. Consequently, the top layer cannot be discretized into many small sub-layers to allow accurate simulation of vertical flow and related movement of the water table. Ignoring the vertical flow near the water table is however acceptable if the water table drawdown is relatively small compared to the thickness of the top layer. This is mostly the case for pumping tests conducted in an unconfined multi-aquifer system, where the extraction well is not located in the top layer.

In this example, we consider an unconfined aquifer with saturated thickness equal to 10 m, radial conductivity equal to 10 m/d, specific storage coefficient equal to 1e-4 m^{-1}, and specific yield equal to 0.2. The extraction well is fully penetrating and the discharge is 100 m³/d. Duration of the test is 1e4 days, which is subdivided in 90 time steps. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 80 rings are defined. The solution is compared to the equivalent Theis (1935) solution for which the aquifer storativity is set equal to the specific yield. Script "..\Examples\Transient\pumpingTestUnconfined.m" implements this example:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,4,91)));

% set grid
m.setgrid(logspace(-1,7,81),10,false);

% set parameters
m.par.kr = 10;
m.par.ss = 1e-4;
m.par.sy = 0.2;

% set stresses
m.stress.q = [100, zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,10,5);

% run model
m.run;

% analytical Theis solution
ir = 1:10:31;
[r,t] = meshgrid(m.grid.r(ir),m.time.t);
u = r.^2./t * m.par.sy/m.par.kr/m.grid.D/4;
s = -m.stress.q(1)/4/pi/m.par.kr/m.grid.D * expint(u);

% time-drawdown graph
figure
semilogx(m.time.t,squeeze(m.s(1,ir,:))','-')
hold on
semilogx(m.time.t,s,'x')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
legend(strcat(num2str(m.grid.r(ir)','%.2f'),'m'))
```

To define a water table, the third input argument `confined` passed to method `setgrid` must be `false`. Additionally, the specific yield parameter must be assigned to property `sy` in the `Parameters` object. The system is not linear, and therefore, it is required to set the maximum number of iterations `mni` greater than 1. The number of iterations used by the SIP solver is stored in property `m.niter`. The total budget for each time step is given by `m.totbud(:,1)`. Drawdown in the last ring is given by `squeeze(m.s(1,end,:))`. Both commands result into vectors with elements close to zero, as it should.

The resulting time-drawdown plot comparing the MAxSym result to the analytical Theis solution is shown in Figure 30. The latter approximates the large-time solution given by Boulton (1963) for the average head in the unconfined aquifer assuming a constant transmissivity and a very large vertical conductivity (see next example).

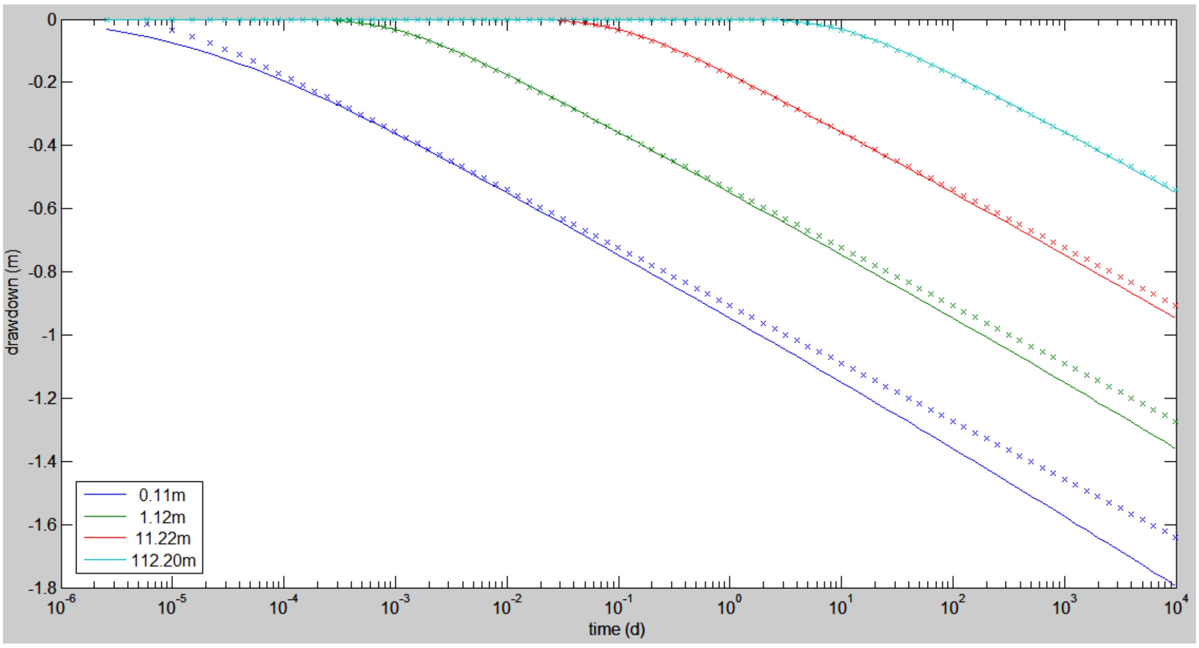


Figure 30. Time-drawdown plot for the pumping test conducted in an unconfined layer. Results are calculated using MAxSym (solid lines) and compared to the corresponding Theis (1935) solution (crosses).

If the radial conductivity is reduced from 10 m/d to 1 m/d, we get the message that the water table drawdown is below the bottom of the top layer for time step 39:

```
>> m.par.kr = 1;
>> m.run
Watertable is below top layer! (time step 39)
>> m.time.t(40)

ans =

    0.0794
```

Drawdown matrix `m.s(:,:,k)` contains `NaN` values only for `k > 39`:

```
>> all(isnan(reshape(m.s(:,:,40:end),[],1)))

ans =

     1
```

Pumping test in aquifer overlain by a water table aquitard

As already mention in previous example, Boulton (1954, 1963) also presented an analytical model to simulate radial flow in an unconfined aquifer. The solution, however, is semi-empirical and later, it was shown by Cooley (1972) and Cooley and Case (1973) that the Boulton model corresponds to an unconfined two-layer model in which the bottom layer is extracted by a fully penetrating well. The top layer is semi-pervious, and hence, it is assumed that horizontal flow and specific elastic storage may be neglected in this layer. Vertical flow in the lower layer is also neglected. Both layers have an infinite lateral extent.

If the specific yield is much larger than the storativity of the lower layer, then the solution for the Boulton-Cooley model is given by (Boulton, 1963):

$$s(r,t) = \frac{-Q}{2\pi T}\int_0^\infty J_0(ry/\sqrt{TC^z})/y\left[1 - \frac{1}{y^2+1}\exp\left(\frac{-\alpha t y^2}{y^2+1}\right) - G(y,t)\right]dy$$

where *s* is the drawdown in the lower layer with transmissivity *T* and storativity *S*, which is extracted at a constant pumping rate *Q*, *r* is the radial distance, and *t* is the time. Parameter *α* is the empirical Boulton delay index, which is in this case defined as $(S_y C^z)^{-1}$ with S_y the specific yield and C^z the vertical resistance of the top layer. J_0 denotes the Bessel function of the first kind of zero order. The infinite integral is evaluated numerically using the `quadgk` function (Shampine, 2008) provided with MATLAB. Function *G(y,t)* is defined as:

$$G(y,t) = \frac{y^2}{y^2+1}\exp\big(-\alpha t(1 + S_y/S)(y^2+1)\big)$$

Function *G* may be omitted to calculate the large-time drawdown. The early-time drawdown corresponding to the first part of the S-shaped time-drawdown graph may be calculated using the Hantush-Jacob solution (Hantush and Jacob, 1955) for a leaky aquifer, that will be discussed later.

In this example, an aquifer with transmissivity of 100 m²/d and storativity of 1e-3 is considered in which a fully penetrating well is located that is extracted with a constant discharge of 100 m³/d. The aquifer is overlain by a semi-pervious layer with thickness of 2 m, radial conductivity of 0.1 m/d, vertical conductivity of 0.02 m/d, specific storage of 1e-5 m^{-1}, and specific yield of 0.2. Recall that the analytical solution neglects the horizontal flow and specific storage in the top layer, and the vertical flow is taken into account by defining the vertical resistance which is equal to 100 d.

In the MAxSym model, two layers are defined where the top layer is set confined. Thickness of the bottom layer is set 1 m to allow T and S to be assigned directly to properties `kr` and `ss`. The thickness and the parameters for the top layer are set as defined above, except for the vertical conductivity, which is taken into account by setting the vertical resistance `cz`. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 80 rings are defined. Duration of the test is 1e4 days, which is subdivided in 90 time steps. The corresponding script "..\Examples\Transient\pumpingTestWatertableAquitard.m" is printed below:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,4,91)));

% set grid
m.setgrid(logspace(-1,7,81),[2;1],false);
```

```
% set parameters
m.par.kr = [0.1;100];
m.par.cz = 100;
m.par.ss = [1e-5;1e-3];
m.par.sy = 0.2;

% set stresses
m.stress.q = zeros(m.grid.nz,m.grid.nr);
m.stress.q(2,1) = 100;

% set solver
m.setsolver(1e-5,10,5);

% run model
m.run;

% Boulton solution
ir = 1:10:31;
alfa = 1/m.par.sy/m.par.cz;
L = sqrt(m.par.kr(2)*m.par.cz);
eta = 1 + m.par.sy/m.par.ss(2);
for i = 1:length(ir)
    r = m.grid.r(ir(i));
    for k = 1:length(m.time.t)
        t = m.time.t(k);
        F = @(x) besselj(0,r.*x/L)./x .* ...
                 (1-1./(x.^2+1).*exp(-alfa*t*x.^2./(x.^2+1))-...
                  x.^2./(x.^2+1).*exp(-alfa*t*eta*(x.^2+1)));
        s(k,i) = -m.stress.q(2,1)/2/pi/m.par.kr(2) * ...
                 quadgk(F,0,Inf,'AbsTol',5e-3);
    end
end

% time-drawdown graph
figure
semilogx(m.time.t,squeeze(m.s(2,ir,:))','-')
hold on
semilogx(m.time.t,s,'x')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
legend(strcat(num2str(m.grid.r(ir)','%.2f'),'m'))
```

The resulting time-drawdown graph comparing the MAxSym result with the analytical solution is given in Figure 31. The number of iterations used by the SIP solver is stored in property `m.niter`. The total budget for each time step is given by `m.totbud(:,1)`. Checking `m.s(:,end,:)` is done to verify if the assumption of infinite lateral extent is met.

A first remark concerns the use of the vertical resistance. By setting property `cz`, it is possible to neglect the vertical flow in the extracted layer and to take into account the vertical resistance defined by the entire thickness of the top layer, as is required for the Boulton-Cooley model. Suppose the vertical conductivities are defined, then MAxSym will calculate a vertical resistance of about 50 d, assuming that the bottom layer has a large vertical conductivity that is negligibly large. The reason lies in the vertical position of the nodal circles, which is the middle of the layers according to the Dupuit (1863) assumption. Hence, MAxSym considers the half thickness of the layers to calculate the vertical resistance between two nodal circles. Both approaches are described in the literature: RADMOD (Reilly and Harbaugh, 1993) and AS2D (Lebbe, 1999) define vertical resistance taking the entire thickness of the phreatic top layer, while MODFLOW-2005 (Harbaugh, 2005) considers the half thickness. It is not clear to the author which one is the best approximation to simulate the effect of a confining phreatic layer. Using MAxSym, however, the modeler is free to choose between both approaches by setting the vertical resistance `cz` or the vertical conductivities `kz`.

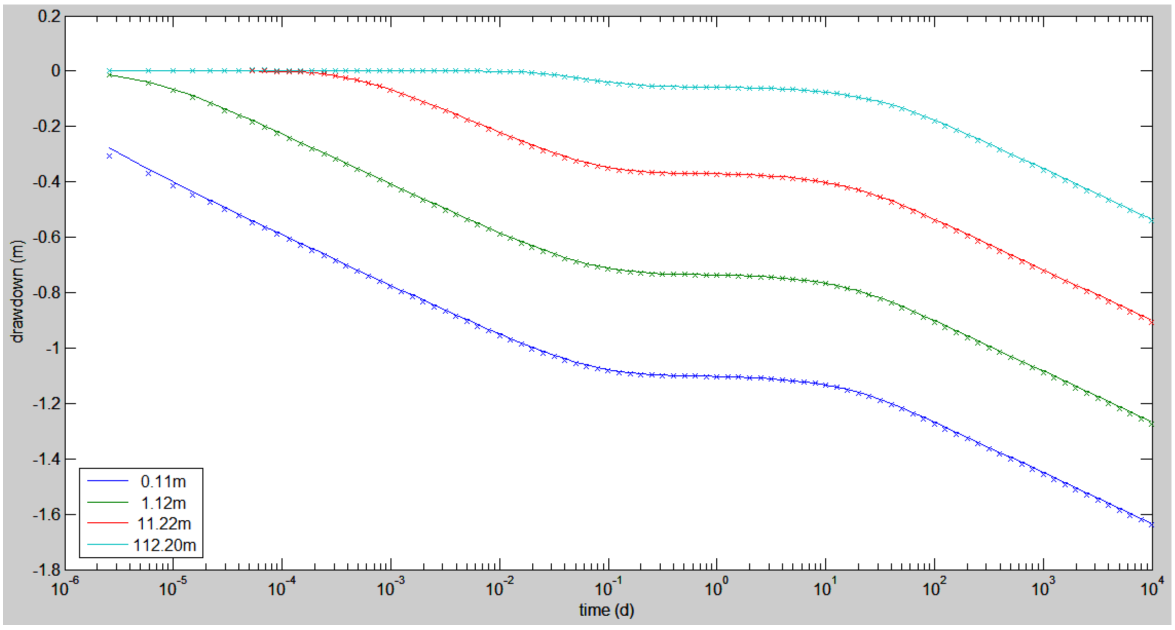


Figure 31. Time-drawdown graph for the Boulton-Cooley model calculated using MAxSym (solid lines) and the analytical expression (crosses).

A second remark is about the application of the Boulton solution when the phreatic layer is pervious. In fact, Boulton's solution was initially derived to simulate flow in a single phreatic layer, meaning that the calculated results in this example may also be interpreted as drawdown due to pumping in a phreatic layer with transmissivity T, storativity S, vertical resistance C^z, and specific yield S_y. As a consequence, MAxSym can be tricked into simulating flow towards a well in a phreatic layer by defining a dummy model layer of unit thickness on top of this layer. The dummy layer must be confined to avoid the model run is aborted, the horizontal conductivity is set zero to neglect horizontal flow, and the specific storage coefficient is set equal to the specific yield. Finally, the total vertical resistance of the phreatic layer is assigned to the vertical resistance between the dummy layer and the layer corresponding to the phreatic layer. This trick was also applied by Louwyck et al. (in press) for the TTim (Bakker, 2010) simulation of an unconfined multi-layer system.

The trick is illustrated here for a phreatic layer with $T = 100$ m²/d, $S = 1e\text{-}3$ m^{-1}, $C^z = 10$ d, and $S_y = 0.2$. The extraction well is fully penetrating and $Q = 100$ m³/d. The following statements are found in script "..\Examples\Transient\BoultonTrick.m":

```
% create Model object
m = MAxSym.Model;
```

```
% set time steps
m.settime(diff(logspace(-5,4,91)));

% set grid
m.setgrid(logspace(-1,7,81),[1;1],true);

% set parameters
m.par.kr = [0;100];
m.par.cz = 10;
m.par.ss = [0.2;1e-3];

% set stresses
m.stress.q = zeros(m.grid.nz,m.grid.nr);
m.stress.q(2,1) = 100;

% set solver
m.setsolver(1e-5,10,5);

% run model
m.run;

% Boulton solution
ir = 1:10:31;
alfa = 1/m.par.ss(1)/m.par.cz;
L = sqrt(m.par.kr(2)*m.par.cz);
eta = 1 + m.par.ss(1)/m.par.ss(2);
for i = 1:length(ir)
    r = m.grid.r(ir(i));
    for k = 1:length(m.time.t)
        t = m.time.t(k);
        F = @(x) besselj(0,r.*x/L)./x .* ...
                 (1-1./(x.^2+1).*exp(-alfa*t*x.^2./(x.^2+1))-...
                  x.^2./(x.^2+1).*exp(-alfa*t*eta*(x.^2+1)));
        s(k,i) = -m.stress.q(2,1)/2/pi/m.par.kr(2) * ...
                 quadgk(F,0,Inf,'AbsTol',5e-3);
    end
end

% time-drawdown graph
figure
semilogx(m.time.t,squeeze(m.s(2,ir,:))','-')
hold on
semilogx(m.time.t,s,'x')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
legend(strcat(num2str(m.grid.r(ir)','%.2f'),'m'))
```

Radial conductivity for the dummy top layer is zero. The dummy layer thickness is 1, so the specific yield can be assigned directly to the specific storage coefficient. The vertical resistance is assigned to the `cz` property. In this way, the thickness of the dummy layer has no influence on the vertical flow.

The resulting time-drawdown plot is shown in Figure 32. It is easy to verify that the Boulton model may be approximated by the corresponding Theis solution with $S = S_y$ if vertical resistance C^z is very small, e.g. 1e-5 d.

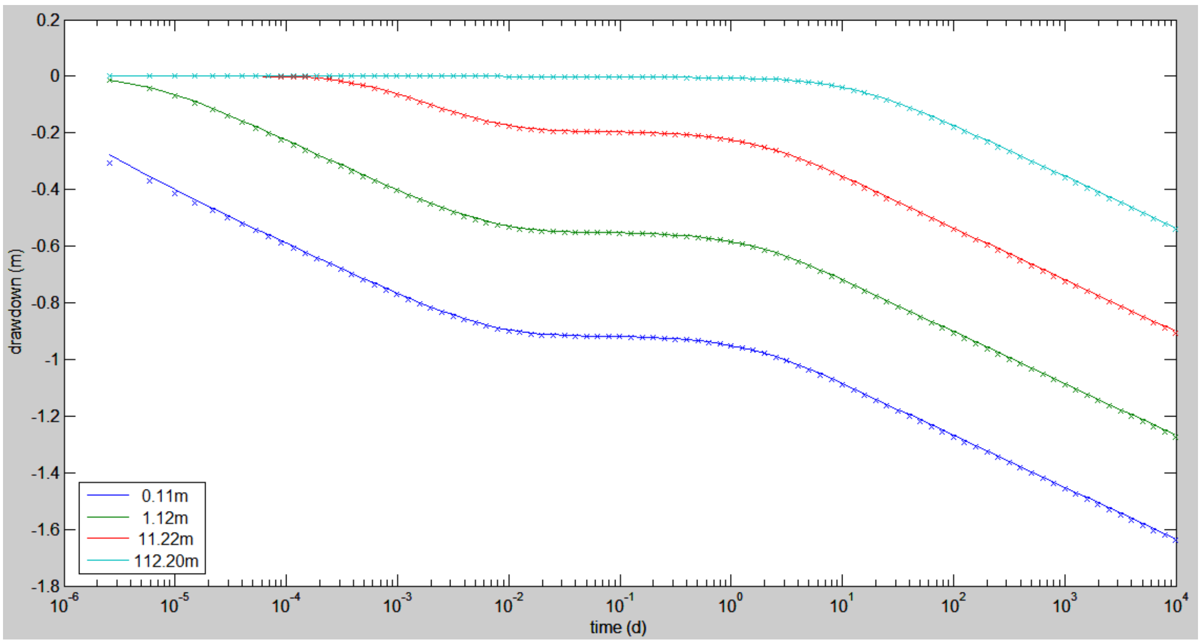


Figure 32. Time-drawdown graph for the Boulton model calculated using MAxSym (solid lines) and the analytical expression (crosses).

Pumping test in unconfined aquifer with recharge

A circular infiltration area may also be defined for transient models. As is the case for steady state models including recharge, the radius *R* defining the infiltration area is derived from the relation $Q = \pi R^2 N$ with *N* the recharge flux. For transient models, this assumption is not required since water can be stored into or removed from the aquifer storage, but it is a convenient way to allow the system to reach steady state.

Consider an unconfined aquifer with saturated thickness of 10 m, radial conductivity of 10 m/d, specific storage coefficient of 1e-4 m^{-1}, and specific yield of 0.2. Additionally, a constant recharge flux of 5e-4 m/d is assumed. The extraction well is fully penetrating and the discharge is 100 m^3/d. Duration of the test is 1e4 days, which is subdivided in 90 time steps. Discretization of radial distance is performed in two steps. First, the infiltration zone extending from the inner grid radius of 0.1 m to radius *R* derived from discharge and recharge is discretized into 30 rings. Second, the zone outside the infiltration area extending from *R* to the outer grid radius of 1e7 m is discretized into 50 rings. Besides radial conductivity and specific storage, specific yield is also set. The discharge matrix `q` defines the well discharge *Q* for the first ring and the recharge for the other rings inside the infiltration area. For each ring, the recharge flux must be converted to a negative discharge value by multiplying –*N* by the horizontal surface area of the rings. Recharge in the first ring is neglected.

Script "..\Examples\Transient\pumpingTestRecharge.m" implements this example:

```
% radius of influence
Q = 100;
N = 5e-4;
R = sqrt(Q/pi/N);

% create Model object
m = MAxSym.Model;
```

```
% set time steps
m.settime(diff(logspace(-5,4,91)));

% set grid
rb = logspace(-1,log10(R),31);
m.setgrid([rb(1:end-1),logspace(log10(R),7,51)],10,false);

% set parameters
m.par.kr = 10;
m.par.ss = 1e-4;
m.par.sy = 0.2;

% set stresses
m.stress.q = [Q, -N*m.grid.hs(2:30), zeros(1,50)];

% set solver
m.setsolver(1e-5,10,5);

% run model
m.run;

% time-drawdown graph
ir = 1:10:21;
figure
semilogx(m.time.t,squeeze(m.s(1,ir,:))','-')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
legend(strcat(num2str(m.grid.r(ir)','%.2f'),'m'))
```

The resulting time-drawdown plot is shown in Figure 33. The total budget for each time step is given by `m.totbud(:,1)`.

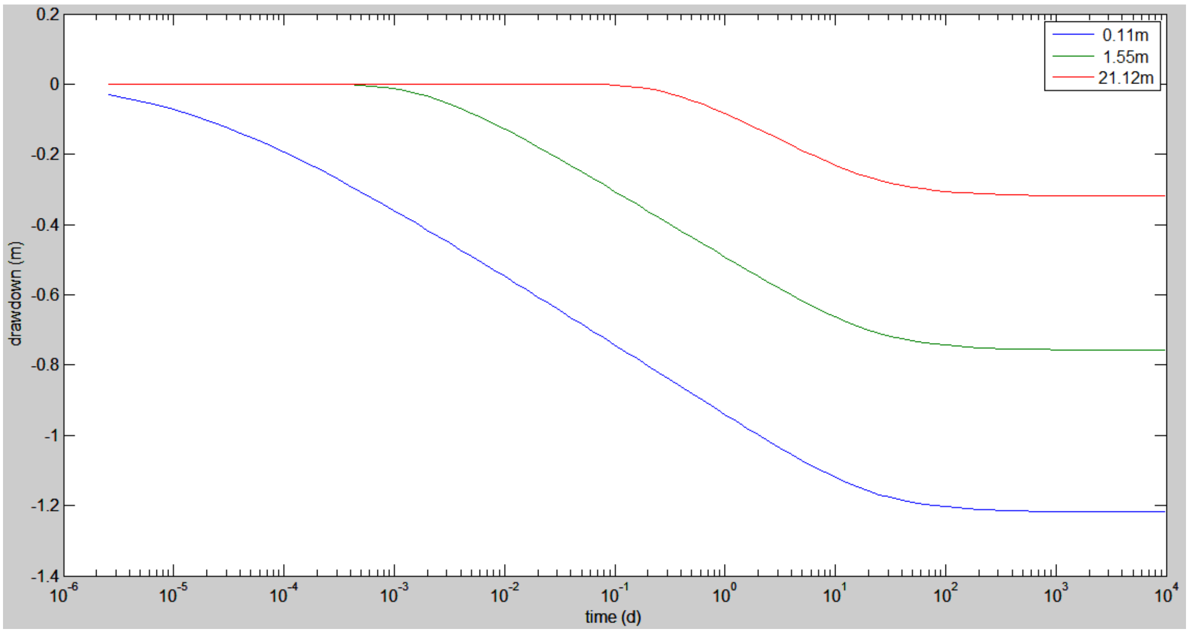


Figure 33. Time-drawdown plot for the pumping test in an unconfined layer with recharge. Results are calculated using MAxSym.

For this example, it is also interesting to look at the relation between drawdown and distance for certain simulation times by plotting some distance-drawdown curves. The following statements create such a plot, which is shown in Figure 34:

```
% distance-drawdown graph
it = 31:10:91;
figure
semilogx(m.grid.r,squeeze(m.s(1,:,it))','-')
hold on
semilogx([R R],ylim(gca),'k:')
set(gca,'fontsize',12)
xlabel('distance (m)')
ylabel('drawdown (m)')
legend(strcat(num2str(m.time.t(it),'%.0e'),'d'))
xlim([0.1 1e3])
```

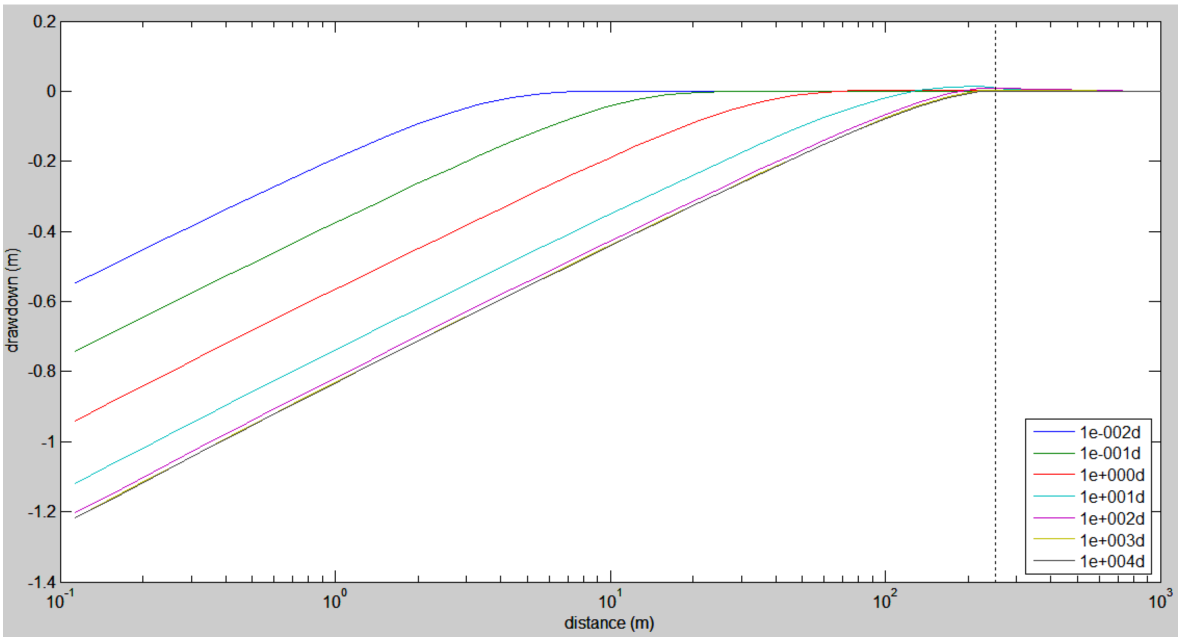


Figure 34. Distance-drawdown plot for the pumping test in an unconfined layer with recharge. Results are calculated using MAxSym. The radius of the infiltration area is 252.3 m, as indicated by the dotted vertical line.

Finally, we would also like to know the amount of extracted water yielded by the aquifer storage and the recharge respectively. The amount of water yielded by the aquifer storage for a certain time step `k` is obtained by summing all negative (or positive) values in matrix `m.qs(:,:,k)`. The remaining amount of water is yielded by the recharge. The following statements calculate the discharge fraction yielded by the aquifer storage as a function of time, and create the plot shown in Figure 35:

```
% plot of storage fraction vs time
figure
qs = squeeze(m.qs);
b = qs >= 0;
qs(b) = 0;
semilogx(m.time.t(2:end),-sum(qs)/Q)
set(gca,'fontsize',12)
xlabel('time (d)')
xlim([1e-3 1e4])
ylim([0 1])
```

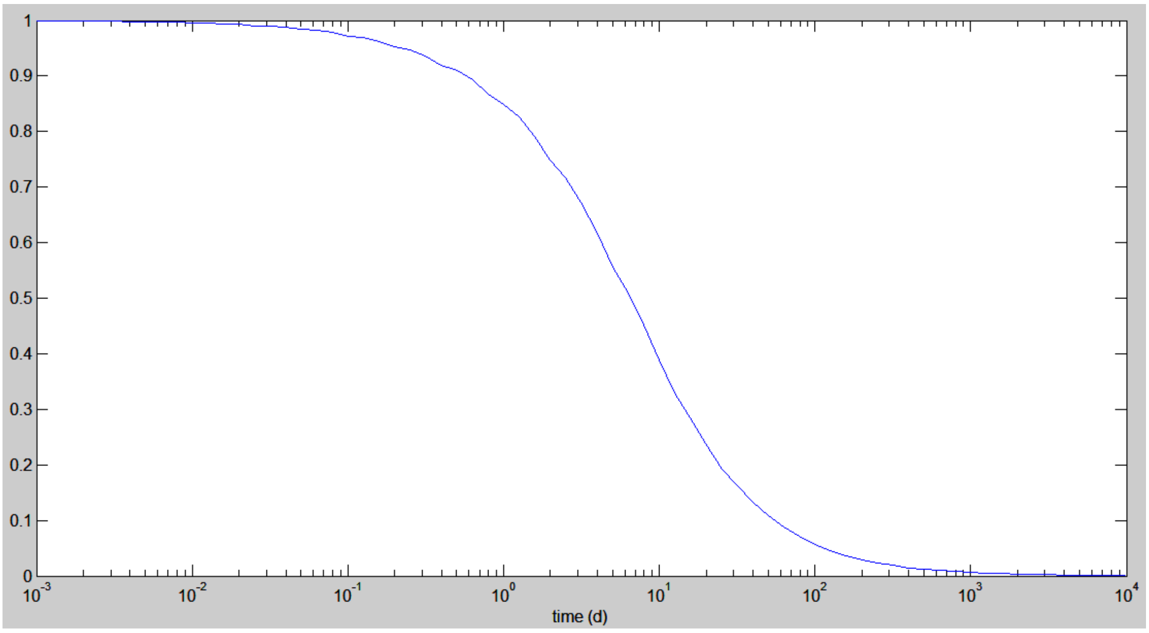


Figure 35. Discharge fraction yielded by the aquifer storage as a function of time for the pumping test in an unconfined layer with recharge. The result is calculated using MAxSym.

Pumping test in leaky aquifer

The transient model that corresponds to the De Glee (1930) model is called the model of Hantush-Jacob and its analytical solution is given by the following expression (Hantush and Jacob, 1955):

$$s(r,t) = \frac{-Q}{4\pi T} W\left(\frac{r^2 S}{4tT}, \frac{r}{\sqrt{TC^z}}\right)$$

where *s* is the drawdown, *Q* is the well discharge, *T* is the aquifer transmissivity, *S* is the aquifer storativity, C^z is the vertical resistance of the overlaying aquitard, *r* is the radial distance, and *t* is the time. Function *W(u,v)* is known as the Hantush Well Function (Hantush and Jacob, 1955):

$$W(u,v) = \int_u^{\infty} e^{-y-v^2/4y}\, dy/y$$

It can be evaluated numerically using the MATLAB function `quadgk`, which applies the adaptive Gauss-Kronrod quadrature (Shampine, 2008).

In this example, the model parameters are *T* = 10 m²/d, *S* = 1e-3, C^z = 1000 d, and *Q* = 100 m³/d. Duration of the test is 1e4 days, which is subdivided into 90 time steps. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 80 rings are defined. Script “..\Examples\Transient\pumpingTestLeaky.m” implements this example:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,4,91)));
```

```
% set grid
m.setgrid(logspace(-1,7,81),[1;1],true);

% set parameters
m.par.constant = [true; false];
m.par.kr = 10;
m.par.cz = 1000;
m.par.ss = 1e-3;

% set stresses
m.stress.q = zeros(m.grid.nz,m.grid.nr);
m.stress.q(2,1) = 100;

% set solver
m.setsolver(1e-5,1);

% run model
m.run;

% analytical solution
ir = 1:10:31;
[r,t] = meshgrid(m.grid.r(ir),m.time.t);
u = r.*r./t * m.par.ss/m.par.kr/4;
for n = 1:numel(u)
    b = r(n)/sqrt(m.par.kr*m.par.cz);
    s(n) = -m.stress.q(2,1)/4/pi/m.par.kr * ...
           quadgk(@(y)exp(-y-b.*b/4./y)./y, u(n), inf);
end
s = reshape(s,size(u));

% time-drawdown graph
figure
semilogx(m.time.t,squeeze(m.s(2,ir,:))','-')
hold on
semilogx(m.time.t,s,'x')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
legend(strcat(num2str(m.grid.r(ir)','%.2f'),'m'))
```

As was the case for the De Glee model, two layers are defined, where the top layer is defined as constant head layer. The aquitard is conceptualized as a semi-confining bed between the two layers, which is done by assigning its resistance C_z to property `cz`. Values for the thickness, the radial conductivity, and the specific storage coefficient are irrelevant for the top layer, because there is no drawdown in this layer throughout the simulation. The value for the `confined` property is irrelevant for the same reason. MAxSym still requires a value set for each layer and by assigning a scalar value to the `kr` and `ss` property, the top layer gets the same value as the bottom layer. The thickness of the second layer is set 1, so radial conductivity is equal to aquifer transmissivity T and specific elastic storage coefficient is equal to storativity S.

The maximum number of iterations for the ADI solver is set 1, because the model is one-dimensional and linear, and hence, its solution can be obtained directly without having to iterate. Figure 36 shows the resulting time-drawdown graph and compares the MAxSym solution with the analytical solution. Note that the maximum drawdown at a certain distance r is the same as the drawdown calculated using the De Glee formula (Figure 16). For small times, the drawdown is the same as the drawdown calculated in the Theis model (Figure 20).

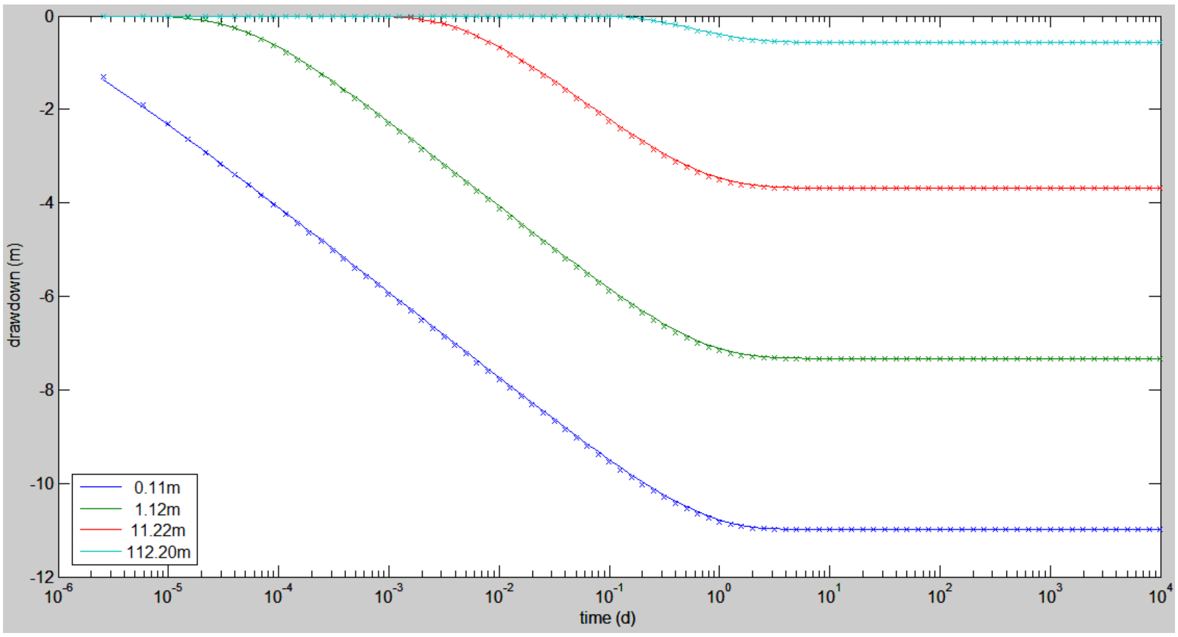


Figure 36. Time-drawdown graph for the Hantush-Jacob model calculated using MAxSym (solid lines) and the analytical expression (crosses).

Total volumetric budget for the variable-head bottom layer at the end of each time step is `m.totbud(:,1)`. The maximum total budget `max(abs(m.totbud(:,1)))` is 2.4130e-12 m^3/d. Property `m.totbud(:,2)` gives the total volumetric budget for the constant-head top layer and it must be equal to the well discharge when maximum drawdown is reached. In fact, total budget for the top layer is equal to the volume of water per time unit due to leakage. Hence, it is always equal to the following expression:

```
>> -sum(squeeze(m.qz(2,:,2:end)))
```

where property `qz` contains the vertical flow terms. Note that the first and last row in the `qz` matrix correspond to the impervious model boundaries. When maximum drawdown is not reached yet, the extracted water is due to both storage and leakage, and consequently, the total sum of both terms must be equal to the extraction rate for each time step:

```
>> sum(squeeze(m.qz(2,:,2:end))) + sum(squeeze(m.qs(2,:,:)))
```

The following statements create a plot that shows the evolution of total leakage and storage decrease as a function of time (Figure 37):

```
% plot of leakage and storage fraction vs time
figure
semilogx(m.time.t(2:end),...
         [sum(squeeze(m.qs(2,:,:)));sum(squeeze(m.qz(2,:,2:end)))]'...
         / -m.stress.q(2,1))
set(gca,'fontsize',12)
xlabel('time (d)')
ylim([0 1])
```

Note that the initial time is ignored, because the volumetric budget terms are calculated at the end of each time step. The vertical flow terms are calculated for each simulation time and hence, the zero terms corresponding to the initial time must also be excluded.

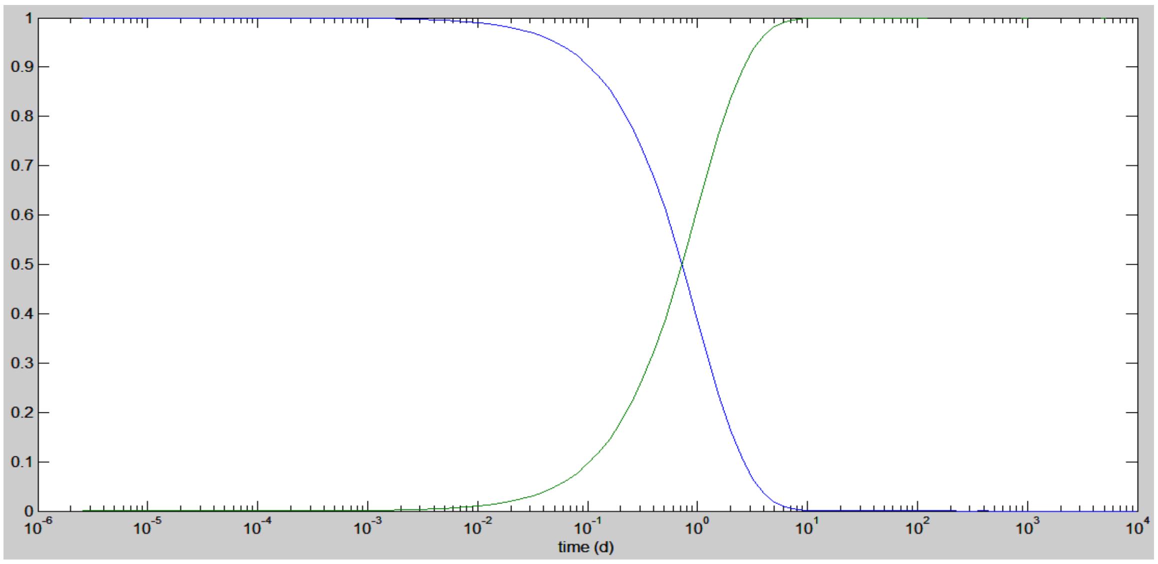


Figure 37. Fraction of the extraction rate due to storage (blue) and leakage (green) as a function of time for the Hantush-Jacob model. Results are calculated using MAxSym.

Pumping test in leaky aquifer with lateral anisotropy

The Hantush-Jacob formula may also be applied in case of a leaky aquifer with laterally anisotropic conductivity (Hantush, 1966):

$$s(r,\theta,t) = \frac{-Q}{4\pi K_e D} W\left(\frac{r^2 S}{4tK_\theta D}, \frac{r}{\sqrt{K_\theta D C^z}}\right)$$

where *D* is the aquifer thickness, *S* is the aquifer storativity, C^z is the vertical resistance of the overlaying aquitard, *Q* is the well discharge, and *W* is the Hantush Well Function. Drawdown *s* not only depends on radial distance *r* and time *t* in this case, but also on the angular position indicated by polar angle ϑ. Radial conductivity also depends on ϑ, which is indicated as K_ϑ. The effective radial conductivity K_e is defined as:

$$K_e = \sqrt{K_{max} K_{min}}$$

where K_{max} is the maximum conductivity and K_{min} is the minimum conductivity. If the direction for the maximum conductivity has polar angle α, then radial conductivity in the direction characterized by angle ϑ is:

$$K_\theta = \frac{1}{\cos^2(\alpha-\theta)/K_{max} + \sin^2(\alpha-\theta)/K_{min}}$$

Calculating $s(r,\vartheta,t)$ using MAxSym is done by first running the corresponding isotropic model with radial conductivity K_e and then calculating drawdown $s(r_a,t)$ with r_a the apparent distance equal to (see, for instance, Lebbe, 1999):

$$r_a = r\sqrt{K_e/K_\theta} = r\sqrt{\cos^2(\alpha-\theta)/a + \sin^2(\alpha-\theta).a}$$

where *a* is the anisotropy factor defined as $(K_{max}/K_{min})^{1/2}$. Calculation of drawdowns corresponding to the apparent distances is done using linear interpolation. The *(x,y)* coordinates for a point with polar coordinates *(r,ϑ)* is calculated as:

$$\begin{cases} x = r\cos(\theta) \\ y = r\sin(\theta) \end{cases}$$

and Cartesian coordinates *(x,y)* are converted into the polar coordinates *(r,ϑ)* as follows:

$$\begin{cases} r = \sqrt{x^2+y^2} \\ \theta = atan2\,(y,x) \end{cases}$$

where *atan2* is the arctangent function that takes into account the quadrant, and that is evaluated in MATLAB using the function with the same name.

As an example, we consider the same model as in previous example with parameters S = 1e-3, C^z = 1000 d, and Q = 100 m³/d, but now, radial conductivity is anisotropic with K_{max} = 20 m/d in the direction determined by angle α = 30°, and K_{min} = 5 m/d. Thickness *D* of the aquifer is 1 m. Duration of the test is 1e4 days, which is subdivided in 90 time steps. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 80 rings are defined.

Script "..\Examples\Transient\pumpTestLatAnisotrop.m" implements this example. First, the MAxSym model is run with conductivity equal to K_e = 10 m/d:

```
% anisotropy
alfa = pi/6;
[kmax,kmin] = deal(20,5);
a = sqrt(kmax/kmin);
ke = sqrt(kmax*kmin);

% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,4,91)));

% set grid
m.setgrid(logspace(-1,7,81),[1;1],true);

% set parameters
m.par.constant = [true; false];
m.par.kr = ke;
m.par.cz = 1000;
m.par.ss = 1e-3;

% set stresses
m.stress.q = zeros(m.grid.nz,m.grid.nr);
m.stress.q(2,1) = 100;

% set solver
m.setsolver(1e-5,1);
```

```
% run model
m.run;
```

Drawdown is calculated at a distance of 1 m and for polar angles equal to 30°, 60°, 90°, and 120°, the latter being the direction of the minimum conductivity. Figure 38 shows the resulting time-drawdown graph comparing the MAxSym result with the analytical solution:

```
% interpolate drawdown corresponding to apparent distances
r = 1;
theta = (30:30:120)/180*pi;
ra = r * sqrt(cos(alfa-theta).^2/a+sin(alfa-theta).^2*a);
sa = squeeze(m.interp1r(2,ra,[]))';

% analytical solution
ktheta = 1./(cos(alfa-theta).^2/kmax+sin(alfa-theta).^2/kmin);
[ktheta,t] = meshgrid(ktheta,m.time.t);
u = r*r./t * m.par.ss./ktheta/4;
v = r./sqrt(ktheta*m.par.cz);
for n = 1:numel(u)
    s(n) = -m.stress.q(2,1)/4/pi/ke * ...
           quadgk(@(y)exp(-y-v(n).*v(n)/4./y)./y, u(n), inf);
end
s = reshape(s,size(u));

% time-drawdown graph
figure
semilogx(m.time.t,sa,'-')
hold on
semilogx(m.time.t,s,'x')
set(gca,'fontsize',12)
title('r = 1m')
xlabel('time (d)')
ylabel('drawdown (m)')
legend(strcat(num2str(theta'/pi*180,'%.0f'),'°'))
```

Finally, it is illustrated how to create a contour plot of drawdown in the leaky aquifer for a certain time. Here, the final simulation time of 1e4 d is taken and a square grid of 200 m x 200 m centered around the pumping well is defined using the MATLAB function `meshgrid`:

```
% contour plot
[x,y] = meshgrid(-100:.5:100);
r = sqrt(x.^2+y.^2);
theta = atan2(y,x);
ra = r .* sqrt(cos(alfa-theta).^2/a+sin(alfa-theta).^2*a);
sa = reshape(m.interp1r(2,ra(:),length(m.time.t)),size(ra));
figure
contourf(x,y,sa);
axis square
colorbar
set(gca,'fontsize',12)
title(['t = ' num2str(m.time.t(end),'%.0f') 'd'])
xlabel('X (m)')
ylabel('Y (m)')
caxis([-9 0])
```

The resulting contour plot is shown in Figure 39. The *(x,y)* coordinates for the grid points are transformed into polar coordinates, which are converted into apparent distances. Drawdowns for the grid points are calculated by interpolating the MAxSym drawdown matrix corresponding to the apparent distances. Method `m.interp1r` is used to perform the linear interpolation. Note that bilinear interpolation should be applied when the selected time does not correspond to one of the simulation times.

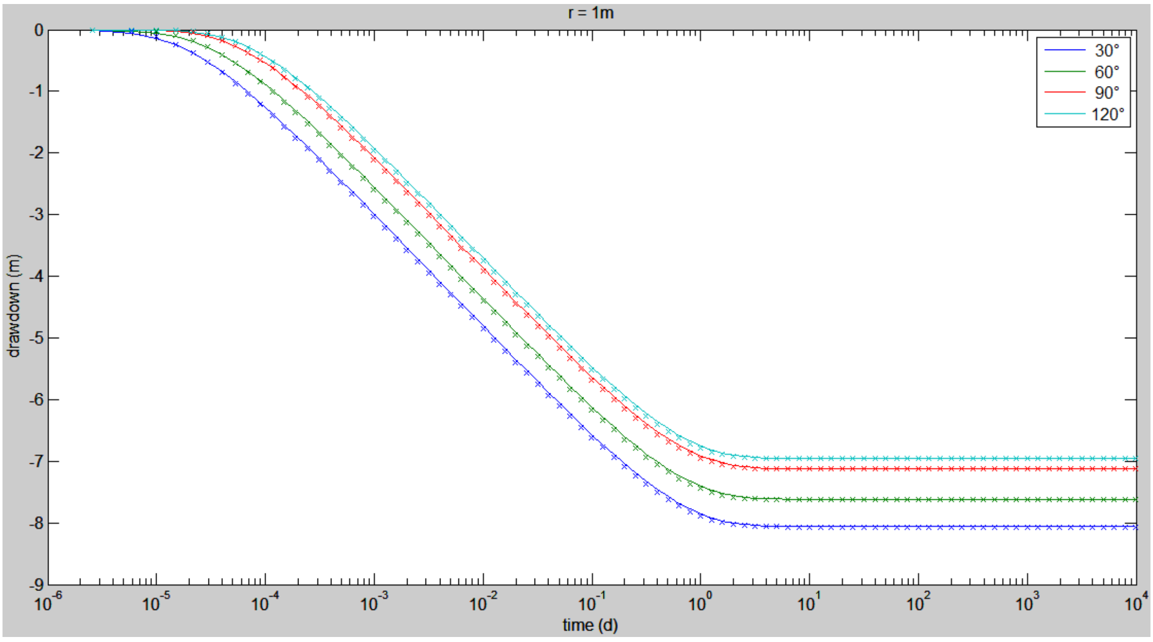


Figure 38. Time-drawdown graph for the Hantush-Jacob model with laterally anisotropic conductivity calculated using MAxSym (solid lines) and the analytical expression (crosses).

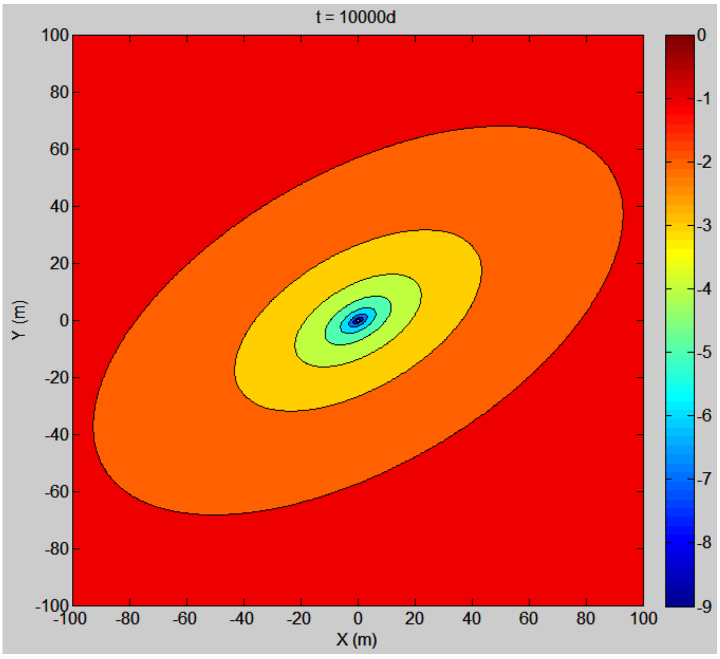


Figure 39. Drawdown contour plot for the Hantush-Jacob model with laterally anisotropic conductivity. Drawdown was calculated using MAxSym.

A final remark concerns the limitation of tricking MAxSym into simulating laterally anisotropic flow. Apparent distances may be used to simulate anisotropic radial flow in confined multi-layer systems only if K_{max}, K_{min} and α are the same for each layer (Lebbe, 1999).

Transient wells in leaky multi-aquifer system

The Hantush-Jacob solution was also generalized by Hemker (1985) for a multi-layer system. A system of *n* aquifers is considered, confined by *n+1* aquitards. A constant-head layer is assumed above the upper aquitard as well as below the lower aquitard. The transmissivity of aquifer *i* is T_i, the storativity of aquifer *i* is S_i, and vertical resistance C^z of aquitard *i* is denoted as c_i. Aquifers and aquitards are numbered from top to bottom. Each aquifer *i* may contain a fully penetrating well from which water is extracted at a constant pumping rate Q_i. The analytical solution is obtained again by decomposing the system matrix *A* into its eigenvalues and eigenvectors, and although Hemker (1985) presents a general solution that is applicable for any multi-aquifer system, we only discuss the special case in which it is assumed diffusivity $T_i/S_i = d$ for all aquifers:

$$A = XYX^{-1}$$

where *A* is an *n x n* tridiagonal matrix defined as:

$$A = \begin{bmatrix} 1/S_1c_1 + 1/S_1c_2 & -1/S_1c_2 & 0 & \cdots & 0 & 0 \\ -1/S_2c_2 & 1/S_2c_2 + 1/S_2c_3 & -1/S_2c_3 & \cdots & 0 & 0 \\ 0 & -1/S_3c_3 & 1/S_3c_3 + 1/S_3c_4 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1/S_nc_n & 1/S_nc_n + 1/S_nc_{n+1} \end{bmatrix}$$

and where *Y* is an *n x n* diagonal matrix with the *n* eigenvalues y_i along the main diagonal and *X* is an *n x n* matrix containing the corresponding eigenvectors in its columns. Drawdown in the *n* aquifers is obtained by the following matrix equation:

$$\bar{s} = XZX^{-1}\bar{q}$$

where $\bar{s}$ is a column vector containing the *n* drawdown values $s_i(r,t)$ for each aquifer *i* at distance *r* and at time *t*, $\bar{q}$ is an *n x 1* column vector with the *i*-th entry equal to $-Q_i/(2\pi T_i)$, and *Z* is an *n x n* diagonal matrix with non-zero elements equal to:

$$Z_{ii} = W\left(\frac{r^2}{4td}, r\sqrt{y_i/d}\right)/2$$

where *W(u,v)* is the Hantush Well Function and *d* is the constant diffusivity value.

We consider 3 aquifers with T_1 = 100 m²/d, T_2 = 10 m²/d, T_3 = 200 m²/d, and *d* = 1e4 m²/d. The resistances for the 4 aquitards are c_1 = 100 d, c_2 = 500 d, c_3 = 300 d, and c_4 = 1000 d. An extraction well is considered in aquifer 1 with Q_1 = 100 m³/d and in aquifer 3 with Q_3 = 1000 m³/d. The simulation lasts 1e4 days and 90 time steps are considered. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 80 rings are defined.

Script "..\Examples\Transient\leakyMultiAquifer.m" implements this example:

```
% input
T = [100; 10; 200];
```

```
c = [100; 500; 300; 1000];
Q = [100; 0; 1000];
d = 1e4;
S = T/d;

% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,4,91)));

% set grid
m.setgrid(logspace(-1,7,81),ones(5,1),true);

% set parameters
m.par.constant = [true; false(3,1); true];
m.par.kr = [0; T; 0];
m.par.ss = [0; S; 0];
m.par.cz = c;

% set stresses
m.stress.q = zeros(m.grid.nz,m.grid.nr);
m.stress.q(2:end-1,1) = Q;

% set solver
m.setsolver(1e-8,10,5);

% run model
m.run;

% analytical solution
ir = [11 31];
a = 1./S./c(1:end-1);
b = 1./S./c(2:end);
q = -Q/2/pi./T;
A = diag(a+b) - diag(a(2:end),-1) - diag(b(1:end-1),1);
[X,Y] = eig(A);
Y = diag(Y);
for i = 1:length(m.time.t)
    t = m.time.t(i);
    for j = 1:length(ir)
        r = m.grid.r(ir(j));
        for n = 1:length(Y)
            v = r*sqrt(Y(n)/d);
            W(n) = quadgk(@(y)exp(-y-v.*v/4./y)./y,r*r/4/t/d,Inf)/2;
        end
        s(:,j,i) = X*diag(W)/X*q;
    end
end

% time-drawdown graphs
figure
for i = 1:length(ir)
    subplot(1,2,i)
    semilogx(m.time.t,squeeze(m.s(2:end-1,ir(i),:))','-')
    hold on
    semilogx(m.time.t,squeeze(s(:,i,:))','x')
    set(gca,'fontsize',12)
```

```
        title(['r = ' num2str(m.grid.r(ir(i)),'%.2f') 'm'])
        xlabel('time (d)')
        ylabel('drawdown (m)')
        legend(strcat('layer',num2str((1:3)')));
    end
```

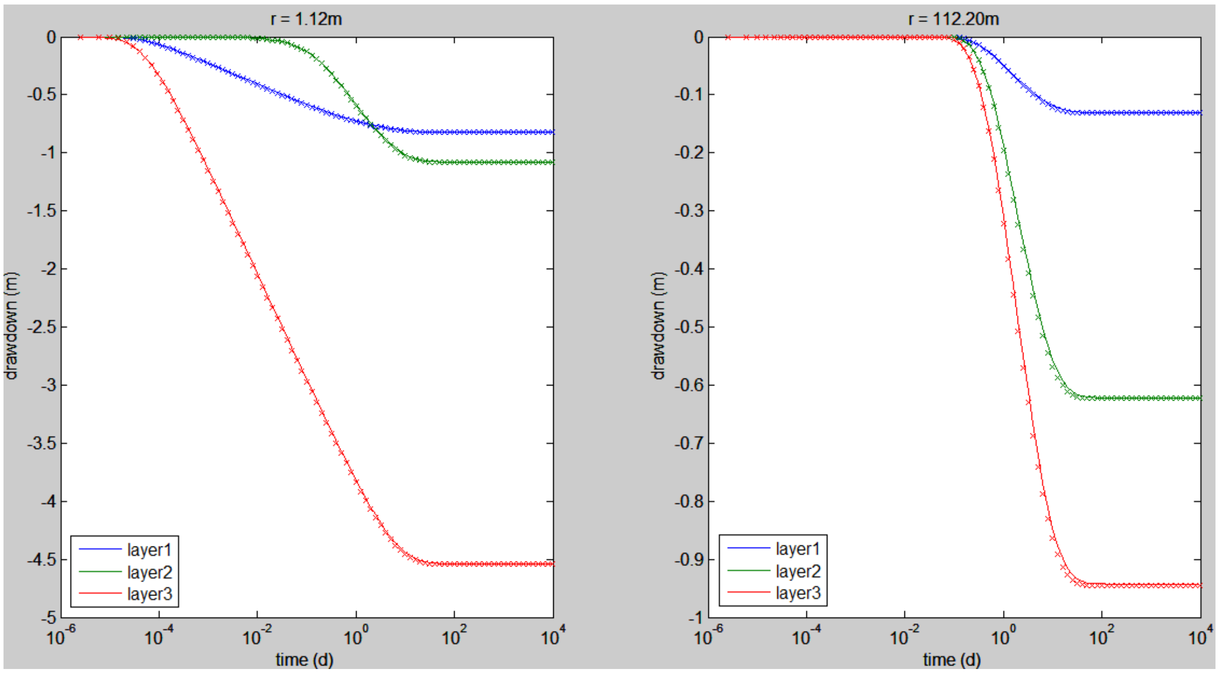


Figure 40. Time-drawdown graphs for the transient Hemker (1985) model calculated using MAxSym (solid lines) and the analytical expression (crosses).

The resulting plot showing the time-distance graphs for two distances is given in Figure 40. The number of layers in the MAxSym model is 5, where the top and bottom layers are defined as constant-head layers. The parameters for these layers must be set, although they are irrelevant. The same is true for the `confined` property. The SIP solver is used, but only a limited number of iterations is needed, which is confirmed by property `m.niter`. A stop criterion of 1e-5, however, yields unacceptably large budget terms and therefore a value of 1e-8 is used. Total volumetric budget for the variable-head layers at the end of each time step is `m.totbud(:,1)`, total volumetric budget for the constant-head layers is `m.totbud(:,2)`. Note that the total storage change for the variable-head layers plus the total volumetric budget for the constant-head layers must be equal to the total discharge `sum(Q(:,1))`:

```
    >> -squeeze(sum(sum(m.qs(2:4,:,:),1),2)) + ...
       squeeze(sum(sum(m.bud([1 5],:,:),1),2))
```

The outer model boundary condition is verified by checking drawdown calculated for the last rings in the variable-head layers: `squeeze(m.s(2:4,end,:))'`.

As for the Hantush-Jacob model, we may again create a plot that shows the fraction of total discharge as a function of time due to storage and leakage respectively. Additionally, we could even look at the contributions of each individual aquifer in this case:

```
    % plot of leakage and storage fraction vs time for each aquifer
    f = squeeze(sum(m.bud([1 5],:,:),2))';
```

```
f(:,3:5) = squeeze(-sum(m.qs(2:4,:,:),2))';
f = f/sum(Q(:,1));
figure
semilogx(m.time.t(2:end),f);
set(gca,'fontsize',12)
xlabel('time (d)')
legend({'leakage top';'leakage bottom';'storage aquifer 1';...
        'storage aquifer 2';'storage aquifer 3';});
```

The resulting plot is shown in Figure 41.

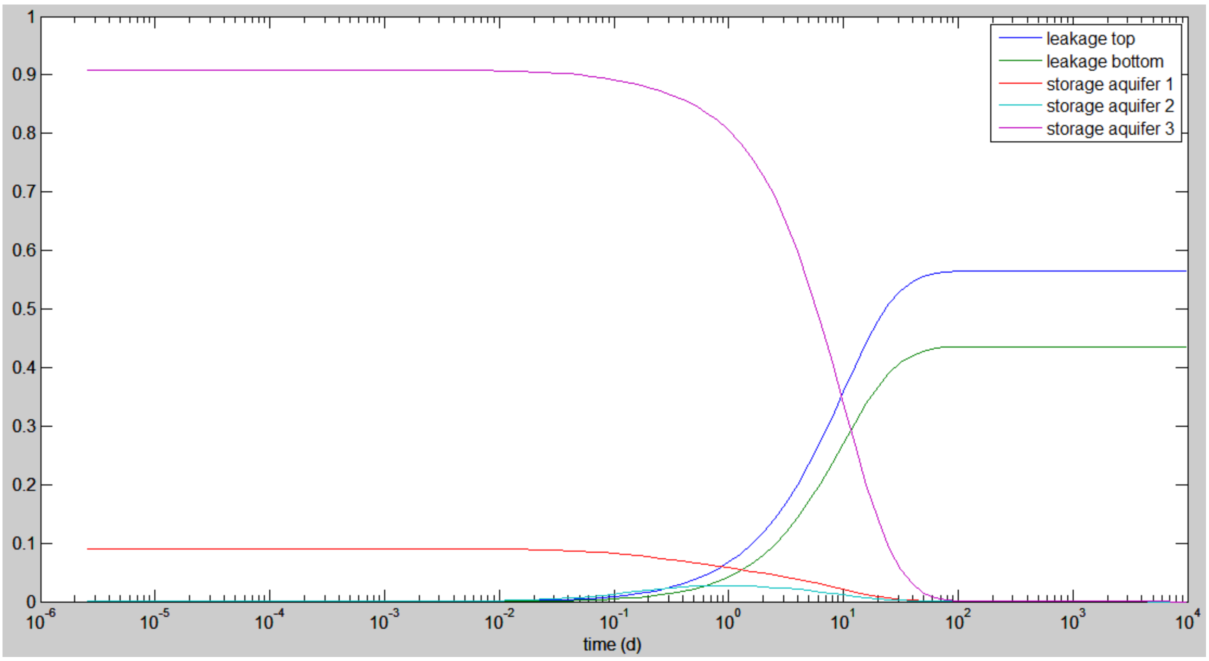


Figure 41. Fraction of total discharge due to storage and leakage as a function of time for the transient Hemker (1985) model. Results are calculated using MAxSym.

Pumping test in partially penetrating well

Consider the same problem as discussed in the Theis example, but now, the extraction well is partially penetrating. This problem is known as the Hantush-Weeks model (Hantush, 1964; Weeks, 1996) and drawdown in an observation well at distance *r* is given by:

$$s(r,t) = \frac{-Q}{4\pi T}\left[W\left(\frac{r^2 S^s}{4tK^r}\right) + F\right]$$

where *s* is the drawdown, *Q* is the well discharge, K^r is the radial conductivity, S^s is the specific elastic storage coefficient, *r* is the radial distance, *t* is the time, and *W(u)* the Exponential Integral. Function *F* is defined as:

$$F = \frac{2D^2}{\pi^2(b-d)(b'-d')}\sum_{n=1}^{\infty}\frac{W\left(\frac{r^2 S}{4tT},\frac{n\pi ar}{D}\right)}{n^2}\left[\sin\left(\frac{n\pi b}{D}\right) - \sin\left(\frac{n\pi d}{D}\right)\right]\left[\sin\left(\frac{n\pi b'}{D}\right) - \sin\left(\frac{n\pi d'}{D}\right)\right]$$

where *D* is the aquifer thickness, *b* and *d* are the depth below the aquifer top of bottom and top of the pumping well screen, *b'* and *d'* are the depth below the aquifer top of bottom and top of the observation well screen, *a* is the anisotropy factor defined as *$(K^z/K^r)^{1/2}$* with *K^z* the vertical conductivity, and *W(u,v)* is the Hantush Well Function.

Consider a confined aquifer with thickness 1 m, radial conductivity 10 m/d, vertical conductivity 1 m/d, and specific storage coefficient 0.001 m^{-1}. The extraction well is in the lower part of the aquifer with *b* = 1 m and *d* = 0.5 m and the pumping rate is 100 m^3/d. Because of the partial penetration of the well, the model grid should have at least 2 layers, but to obtain an accurate simulation of vertical flow, the aquifer is subdivided into 10 layers of 0.1 m. The extraction lasts 1e4 days and 90 time steps are considered. The inner grid radius is 0.1 m, the outer grid radius approximates the boundary at infinity and is set 1e7 m, and 80 rings are defined. Observations are calculated at three distances from the well in layer 3 and 8.

Script "..\Examples\Transient\pumpTestPartWell.m" implements this example:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,4,91)));

% set grid
m.setgrid(logspace(-1,7,81),.1*ones(10,1),true);

% set parameters
m.par.inactive = false(m.grid.nz,m.grid.nr);
m.par.inactive(1:5,1) = true;
m.par.kr = 10;
m.par.kz = 1;
m.par.cz = sparse(m.grid.nz-1,m.grid.nr);
m.par.cz(6:end,1) = 1e-5;
m.par.ss = 1e-3;

% set stresses
m.stress.q = zeros(m.grid.nz,m.grid.nr);
m.stress.q(6:end,1) = 20;

% set solver
m.setsolver(1e-5,1000,5);

% run model
m.run;

% analytical solution
ir = 11:10:31;
ilay = [3 8];
[t,r] = meshgrid(m.time.t,m.grid.r(ir));
u = r.*r./t * m.par.ss/m.par.kr/4;
D = sum(m.grid.D);
[b,d] = deal(D,D/2);
[bq,dq] = deal(ilay/m.grid.nz*D,(ilay-1)/m.grid.nz*D);
a = sqrt(m.par.kz/m.par.kr);

for i = 1:length(ilay)
    F = zeros(size(u));
```

```
        for j = 1:numel(u)
            for n = 1:5
                v = n*pi*a*r(j)/D;
                F(j) = F(j) + ...
                       quadgk(@(y)exp(-y-v.*v/4./y)./y,u(j),inf)/n/n *...
                       (sin(n*pi*b/D)-sin(n*pi*d/D))*...
                       (sin(n*pi*bq(i)/D)-sin(n*pi*dq(i)/D));
            end
        end
        s(i,:,:) = -sum(m.stress.q(:,1))/4/pi/m.par.kr/D * ...
                     (expint(u) + 2*D*D/pi/pi/(b-d)/(bq(i)-dq(i))*F);
    end

    % time-drawdown graph
    figure

    for i = 1:length(ilay)
        subplot(1,2,i)
        semilogx(m.time.t,reshape(m.s(ilay(i),ir,:),length(ir),[])','-')
        hold on
        semilogx(m.time.t,reshape(s(i,:,:),length(ir),[])','x')
        set(gca,'fontsize',12)
        title(['layer ' int2str(ilay(i))])
        xlabel('time (d)')
        ylabel('drawdown (m)')
        ylim([-16 0])
        legend(strcat(num2str(m.grid.r(ir)','%.2f'),'m'))
    end
```

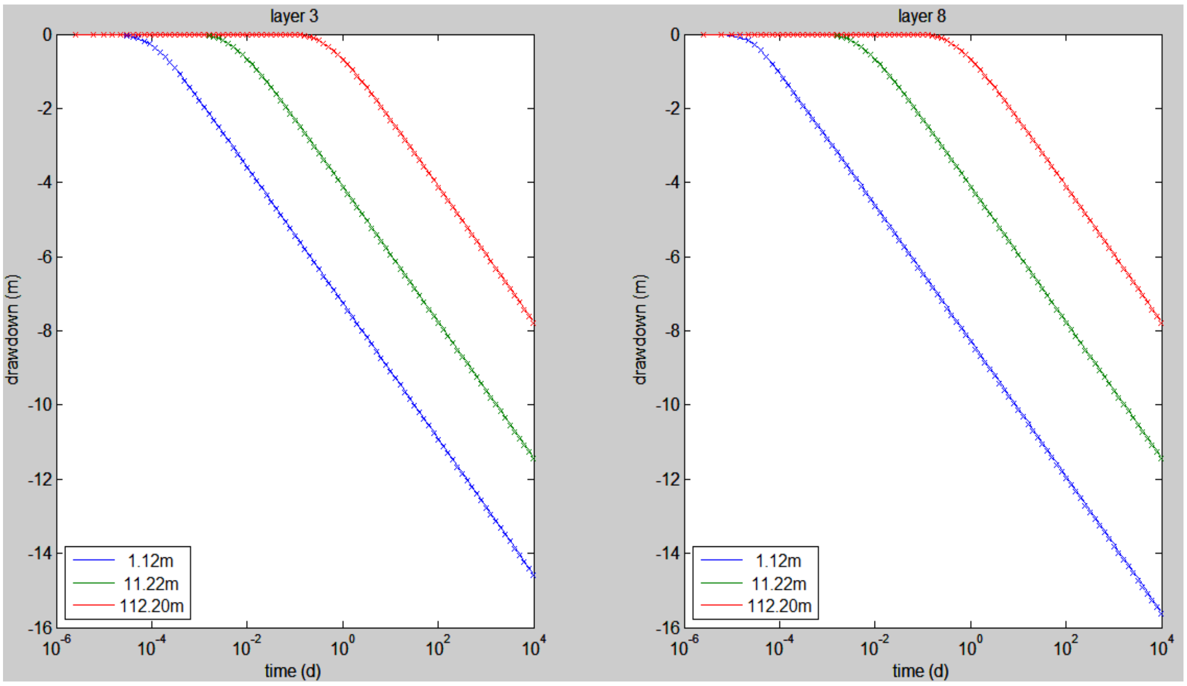


Figure 42. Time-drawdown graph for the Hantush-Weeks model calculated using MAxSym (solid lines) and the analytical expression (crosses) given by Hantush (1964).

Error! Reference source not found. shows the resulting time-drawdown graphs for the selected istances in layer 3 and 8. The SIP solver was used and the maximum number of iterations `mni` was augmented to 1000, because there are 10 times more equations to solve than for the equivalent

Theis model. Moreover, assinging small values to `cz` for the first rings slows down the convergence rate. Getting property `m.niter` tells us, for instance, that 506 iterations were needed in the last time step to solve the system of equations. Total budget for each time step is `m.totbud(:,1)`. Outer model boundary conditions are verified by checking `squeeze(m.s(:,end,:))'`.

The given example clearly illustrates how to define an extraction well when the pumped layer is discretized into sub-layers:

1) First, an equal fraction of the total discharge is assigned to each of the pumped sub-layers. In our example, 5 sub-layers are considered for a total discharge of 100 m^3/d, so 20 m^3/d is assigned to each sub-layer.

2) Second, a very small value is assigned to the vertical resistance `cz` between the discharged rings to obtain the same drawdown in each of these rings. Here, a value of 1e-5 d is used and this value overwrites the vertical resistance derived from the input `kz`. By keeping the drawdown in the discharged rings the same, flow through the well screen will not be uniform, as can be verified by getting `m.qr(6:end,2,:)`.

3) Finally, the non-screened upper part of the well is taken into account by defining the first rings inactive if they are not discharged. Note that drawdown in these inactive rings is set `NaN`, while the budget terms are set zero.

Figure 43 shows the time-drawdown graph for the first ring compared with the analytical solution. As expected, the curves for the discharged rings are the same. The following statements create this plot, where the well radius is equal to `m.grid.r(1)`:

```
% analytical solution for drawdown in the pumping well
r = m.grid.r(1);
u = r*r./m.time.t * m.par.ss/m.par.kr/4;
F = zeros(size(u));

for j = 1:numel(u)
    for n = 1:10
        v = n*pi*a*r/D;
        F(j) = F(j) + ...
               quadgk(@(y)exp(-y-v.*v/4./y)./y, u(j),inf)/n/n *...
               (sin(n*pi*b/D)-sin(n*pi*d/D))^2;
    end
end

s = -sum(m.stress.q(:,1))/4/pi/m.par.kr/D * ...
    (expint(u) + 2*D*D/pi/pi/(b-d)^2 * F);

% time-drawdown graph
figure
semilogx(m.time.t,squeeze(m.s(6:end,1,:))','k',m.time.t,s,'kx')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
```

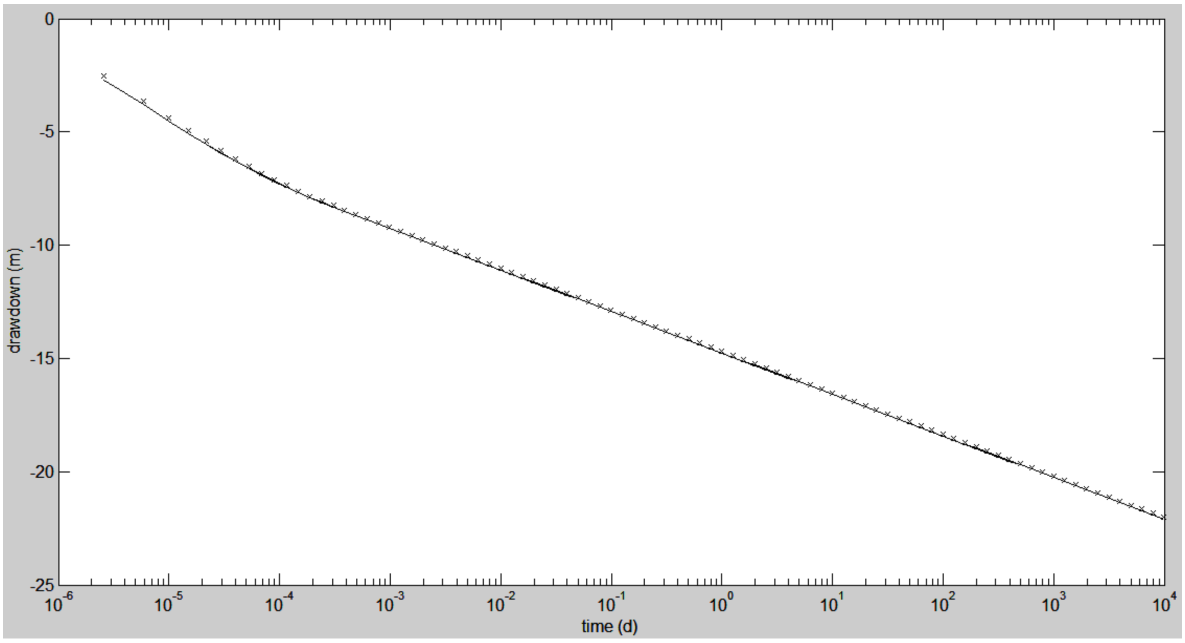


Figure 43. Drawdown as a function of time in the first ring representing the well for the Hantush-Weeks model calculated using MAxSym (solid lines) and the analytical expression (crosses) given by Hantush (1964).

Pumping test in unconfined multi-aquifer system

The advantage of using a numerical code such as MAxSym for simulation of axi-symmetric flow becomes obvious when modeling pumping tests that combine different features discussed in previous examples. Suppose we want to simulate a pumping test conducted in a partially penetrating well of large diameter with finite-thickness skin and located in an unconfined multi-aquifer system. Elaboration of an analytical solution for this kind of problem would be cumbersome or even impossible, while the use of MAxSym to set up the required model is as straightforward as setting up the simple models discussed above.

Consider, for example, an unconfined system of infinite lateral extent consisting of five layers with the following characteristics:

- Layer 1: D = 5 m, K^r = 10 m/d, K^z = 1 m/d, S^s = 1e-4 m^{-1}, S^y = 0.2
- Layer 2: D = 1 m, K^r = 0.5 m/d, K^z = 0.1 m/d, S^s = 1e-4 m^{-1}
- Layer 3: D = 6 m, K^r = 5 m/d, K^z = 1 m/d, S^s = 1e-5 m^{-1}
- Layer 4: D = 3 m, K^r = 0.01 m/d, K^z = 5e-4 m/d, S^s = 1e-6 m^{-1}
- Layer 5: D = 4 m, K^r = 1 m/d, K^z = 0.2 m/d, S^s = 1e-5 m^{-1}

where D is the layer thickness, K^r is the radial conductivity, K^z is the vertical conductivity, S^s is the specific storage coefficient, and S^y is the specific yield. An extraction well of radius 0.25 m is present in layer 3. The well casing is situated in layers 1, 2 and 3, and the well screen with length 2 m is situated in the middle of layer 3. An impervious clay seal with thickness 0.05 m surrounds the casing in layers 1 and 2. In layer 3, there is a well skin with thickness 0.05 m and conductivity K^r = K^z = 50 m/d. The well is extracted at a constant pumping rate equal to 250 m³/d for a period of 100 days.

The model discretization is as follows. The simulation time is discretized into 70 time steps. The model layers correspond to the layers in the aquifer system and additional sub-layers are defined to simulate vertical flow more accurately: the top layer is subdivided into 2 sub-layers to account for the water table movement, and the extracted layer must have at least 3 sub-layers to account for the partial penetration of the well. Furthermore, the non-screened parts of the extracted layer are subdivided into 2 sub-layers to simulate accurately the vertical flow towards the well. The screened part of layer 3 is not subdivided into sub-layers because the flow is mainly horizontal in this part. Moreover, discretization of the screened part would slow down significantly the iterative process due to the low resistances that are required to get drawdown in the well rings equal.

The radial distance is discretized in two steps. First, the skin zone between 0.25 and 0.3 m is subdivided into 10 rings and an additional ring corresponding to the well is defined accordingly. Second, the infinite extent of the aquifer system is approximated by defining an outer radius of 1e6 m, and 70 rings are defined. Inactive rings are defined where the well is not screened. The impervious clay seal is also defined using inactive rings. When setting the aquifer parameters, one should keep in mind that the first 11 rings in the pumped layer correspond to the well skin. Finally, the wellbore storage is taken into account because of the large diameter of the well.

Script "..\Examples\Transient\unconfinedMultiAquifer.m" implements this example:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,2,71)));

% set grid
rb = logspace(log10(0.25),log10(0.3),11);
rb = [rb(1)^2/rb(2), rb];
rb = [rb(1:end-1),logspace(log10(0.3),6,71)];
m.setgrid(rb,[2;3;1;1;1;2;1;1;3;4],false);

% set parameters
m.par.inactive = false(m.grid.nz,m.grid.nr);
m.par.inactive(1:3,1:10) = true;
m.par.inactive([4:5,7:8],1) = true;
m.par.kr = repmat([10;10;0.5;5;5;5;5;5;0.01;1],1,m.grid.nr);
m.par.kr(4:8,1:10) = 50;
m.par.kz = repmat([1;1;0.1;1;1;1;1;1;0.0005;0.2],1,m.grid.nr);
m.par.kz(4:8,1:10) = 50;
m.par.ss = repmat(1e-5*[10;10;10;1;1;1;1;1;0.1;1],1,m.grid.nr);
m.par.ss(6,1) = m.grid.rb(2)^2*pi/m.grid.vol(1);
m.par.sy = 0.2;

% set stresses
m.stress.q = sparse(m.grid.nz,m.grid.nr);
m.stress.q(6,1) = 250;

% set solver
m.setsolver(1e-6,50,5);

% run model
m.run;

% time-drawdown graph
figure
```

```
ilay = [1 6 10];
ir = 19;
semilogx(m.time.t,[squeeze(m.s(6,1,:)),squeeze(m.s(ilay,ir,:))'])
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
str = {'pumping well'};
for n = ilay
    str{end+1} = sprintf('layer %d, r = %.2fm',n,m.grid.r(ir));
end
legend(str)
```

Figure 44 shows the resulting time-drawdown curves for pumping well and observation wells at *r* = 1.5 m in the top layer near the water table, in the screened part of the pumped layer, and in the lower layer. Maximum total budget `max(m.totbud(:,1))` is 8.471e-4 m^3/d. Outer model boundary conditions are verified by checking `squeeze(m.s(:,end,:))'`.

When the model setup is more complicated, it may be useful to create a plot showing a vertical cross-section through the aquifer. The following statements create such a plot for the model setup in this example:

```
% plot model set up
figure
x = [0.1 m.grid.rb(end)];
y = m.grid.zb';
set(axes('parent',gcf),'xlim',x,'xscale','log',...
    'ylim',y([end 1]),'box','on','layer','top','fontsize',12)
x = [x fliplr(x)];
xlabel('r (m)')
ylabel('z (m)')

% layers
nl = cumsum([1 2 1 5 1 1]);
c = {'y','g','y','g','y'};
for n = 1:5
    patch(x,y(nl([n,n,n+1,n+1])),c{n},'parent',gca)
end

% well
rw = m.grid.rb(2);
rs = m.grid.rb(12);
patch([x(1),rw,rw,x(1)],[y([1 1]),y(nl([4 4]))],'b','parent',gca)

% clay seal and skin
patch([rw,rs,rs,rw],[y([1 1]),y(nl([3 3]))],'k','parent',gca)
patch([rw,rs,rs,rw],y(nl([3 3 4 4])),'r','parent',gca)

% sub-layers
line([rw,m.grid.rb(end)],[y;y],'color','k','parent',gca)

% screen
y = linspace(y(6),y(7),10);
line([x(1),rw],[y;y],'color','k','parent',gca)
```

Figure 45 shows the resulting cross-section plot.

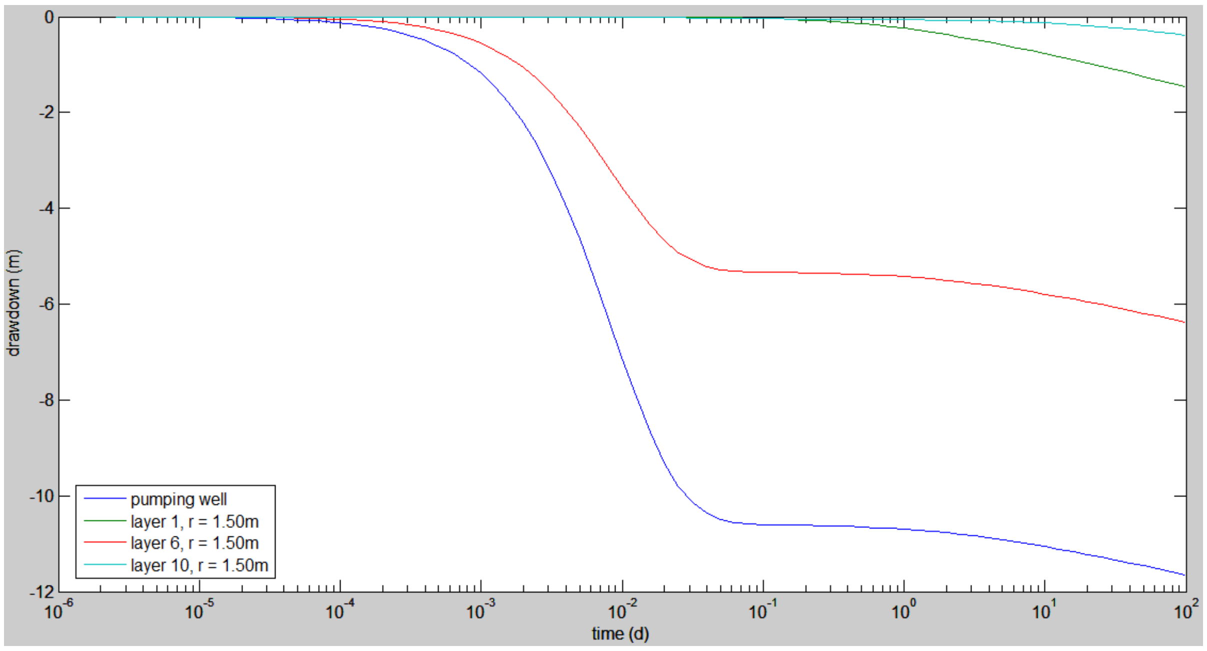


Figure 44. Time-drawdown curves for the unconfined multi-aquifer model calculated using MAxSym.

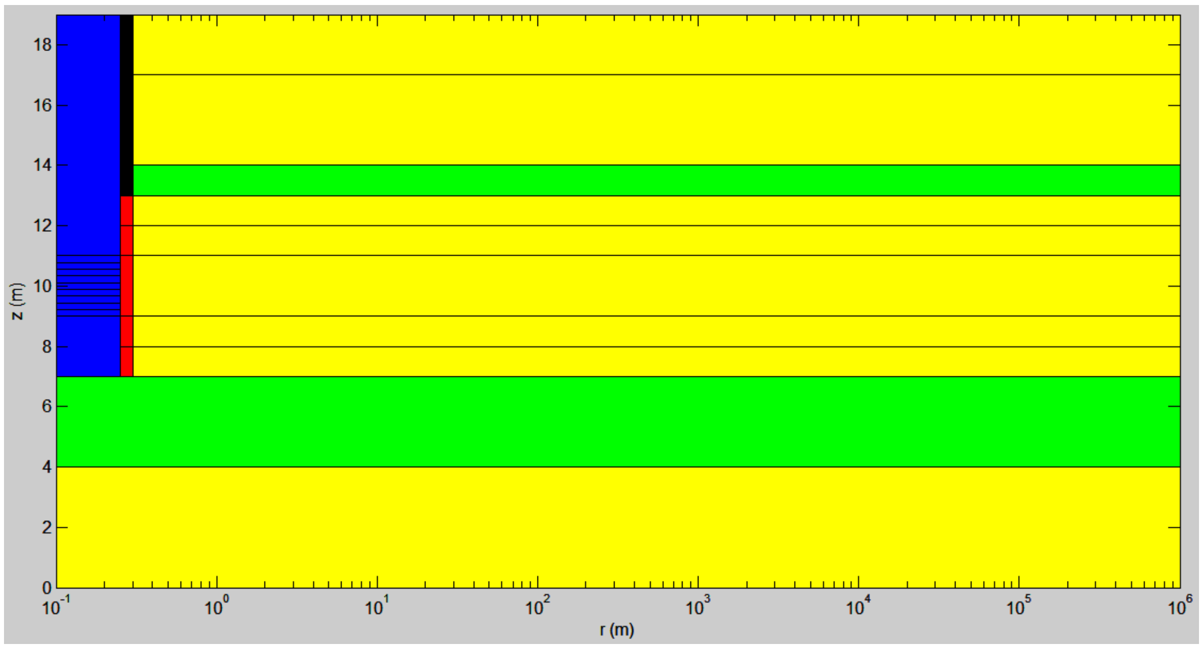


Figure 45. Schematization for the unconfined multi-aquifer model: yellow and green zones correspond to layers in the aquifer system, the blue zone corresponds to the extraction well, the black zone to the clay seal, and the red zone to the well skin. Well screen and sub-layers are also indicated.

Slug test in confined aquifer

A slug test performed in a well that fully penetrates a confined, homogeneous aquifer of infinite lateral extent can be simulated using the Cooper et al. (1967) solution:

$$H(t) = \frac{8SH_0}{\pi^2} \int_0^\infty e^{-tTy^2/r_w^2 S}\, dy/(yF(y))$$

where H is the head change within the well with radius r_w, H_0 is the initial head change within the well at $t = 0$ with t the time, T is the aquifer transmissivity, and S is the aquifer storativity. Function *F(y)* is defined as:

$$F(y) = [yJ_0(y) - 2SJ_1(y)]^2 + [yY_0(y) - 2SY_1(y)]^2$$

with J_0 and J_1 the Bessel functions of the first kind and order 0 and 1 resp., and Y_0 and Y_1 the Bessel functions of the second kind and order 0 and 1 resp. Note that it is assumed the head change within the well is equal to the drawdown at the well screen face $s(r_w,t)$. The infinite integral can be evaluated numerically using the MATLAB function `quadgk` (Shampine, 2008).

In this example, the model parameters are $T = 1$ m²/d, $S = 1e\text{-}5$, $r_w = 0.03$ m, and $H_0 = 1$ m. Duration of the test is 1 day, which is subdivided in 250 time steps. The inner grid radius is derived from the well radius, the outer grid radius is set 1000 m, and 400 rings are defined. The following statements to run the model are implemented in script "..\Examples\Transient\slugTestConfined.m":

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-6,1,251)));

% set grid
rb = logspace(log10(0.03),3,401);
rb = [rb(1)^2/rb(2), rb];
m.setgrid(rb,1,true);

% set parameters
m.par.kr = 1;
m.par.ss = [m.grid.rb(2)^2*pi/m.grid.vol(1), ...
            1e-5*ones(1,m.grid.nr-1)];

% set stresses
m.stress.s0 = [1 ,zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,1);

% run model
m.run;

% analytical solution
F = @(y)  (y.*besselj(0,y)  - 2*m.par.ss(end)*besselj(1,y)).^2 + ...
          (y.*bessely(0,y)  - 2*m.par.ss(end)*bessely(1,y)).^2;
G = @(y,t) exp(-m.par.kr*t/m.grid.rb(2)^2.*y.*y/m.par.ss(end)) ...
         ./y./F(y);
for n = 1:length(m.time.t)
    s(n)  = 8*m.stress.s0(1)*m.par.ss(end)/pi/pi * ...
           quadgk(@(y)G(y,m.time.t(n)),0,Inf);
end

% time-drawdown graph
figure
semilogx(m.time.t,squeeze(m.s(1,1,:)),'k',m.time.t,s,'kx')
set(gca,'fontsize',12)
```

```
    xlabel('time (d)')
    ylabel('drawdown (m)')
    xlim([1e-6 1])
```

The given example clearly illustrates how to define a head specified well for simulation of a slug test:

1) First, the outer radius of the first ring must be equal to the well radius. The width of the first ring must be sufficiently small because the drawdown in this ring approximates the drawdown at the face of the well screen. This explains why the radial extent of the model grid is more discretized than in previous examples.

2) Second, the wellbore storage is taken into account by setting the specific storage coefficient of the first ring equal to $\pi r_w^2/V(1)$ with *V(1)* the volume of the first ring. Dividing by the volume is required, because the specific storage coefficients are multiplied by the ring volumes when storage changes are calculated. To obtain an accurate simulation of the wellbore storage, simulation time is also more discretized than in previous examples.

3) Finally, the initial head change is defined by setting the `s0` property in the `Stresses` object.

The thickness of the aquifer is set 1. In this way, the horizontal conductivity and the specific elastic storage coefficient are equal to aquifer transmissivity *T* and storativity *S* respectively. The maximum number of iterations for the ADI solver is set 1, because the model is one-dimensional and linear, and hence, its solution is obtained directly without having to iterate.

Figure 46 shows the resulting time-drawdown graph. Total budget at the end of each time step is given by `m.bud(:,1)`, and maximum total budget `max(abs(m.bud(:,1)))` is 5.0128e-12 m^3/d. The outer model boundary condition is verified by checking `squeeze(m.s(1,end,:))`.

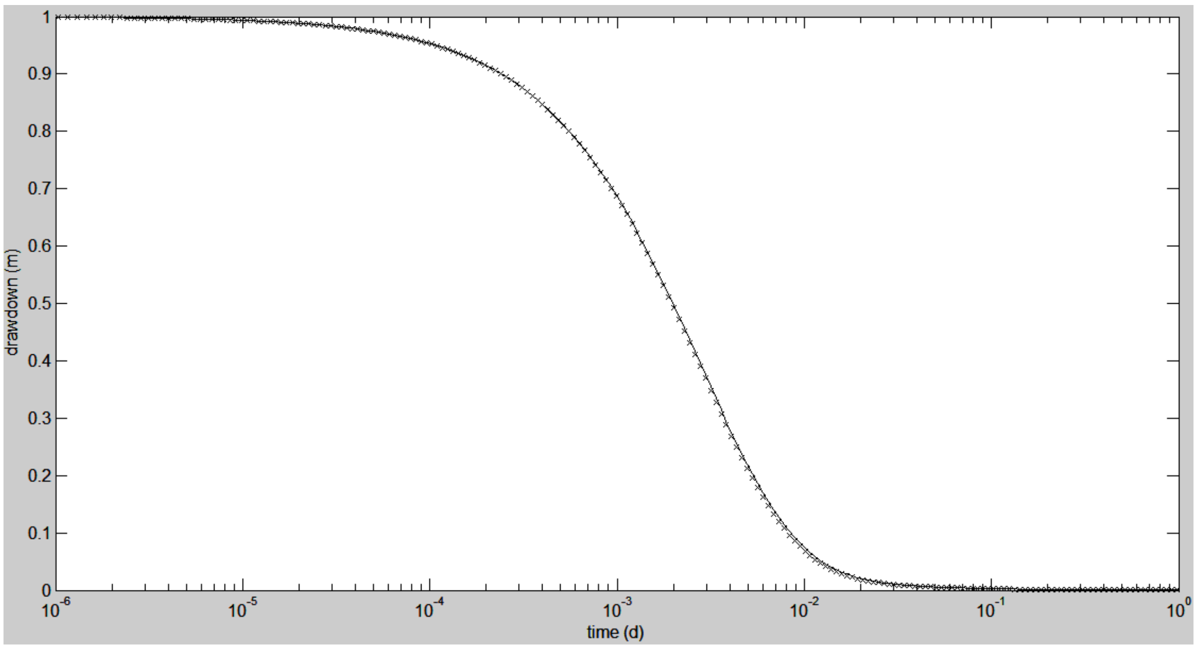


Figure 46. Time-drawdown graph for the Cooper et al. (1967) model calculated using MAxSym (solid lines) and the analytical expression (crosses).

Slug test with non-instantaneous head change

Slug test models usually assume an instantaneous head change within the well at the start of the test. In practice, however, this assumption is not always met. Using MAxSym, it is possible to take into account the non-instantaneous injection or removal of water into or from the well.

Consider the same test as described in previous example: T = 1 m²/d, S = 1e-5, r_w = 0.03 m, and H_0 = 1 m. Suppose it takes 3 seconds to realize the total head change, where it is assumed the head change is linear as a function of time. After the total head change is realized, a simulation period of 1 day is considered, which is subdivided in 250 time steps. The inner grid radius is derived from the well radius, the outer grid radius is set 1000 m, and 400 rings are defined. The following statements to run the model are implemented in script "..\Examples\Transient\slugTestNonInstant.m":

```
% create Model object
m = MAxSym.Model;

% set time steps
n = 31;
ti = linspace(0,3,n)/24/60/60; % 3 seconds
m.settime([diff(ti),diff(logspace(-6,1,251))],[ones(1,n-1),250]);

% set grid
rb = logspace(log10(0.03),3,401);
rb = [rb(1)^2/rb(2), rb];
m.setgrid(rb,1,true);

% set parameters
m.par.kr = 1;
m.par.ss = [m.grid.rb(2)^2*pi/m.grid.vol(1), ...
            1e-5*ones(1,m.grid.nr-1)];

% set stresses
for k = 1:n-1
    m.stress(k).s0 = [1/(n-1) ,zeros(1,m.grid.nr-1)];
end

% set solver
m.setsolver(1e-5,1);

% run model
m.run;

% analytical solution
F = @(y) (y.*besselj(0,y) - 2*m.par.ss(end)*besselj(1,y)).^2 + ...
         (y.*bessely(0,y) - 2*m.par.ss(end)*bessely(1,y)).^2;
G = @(y,t) exp(-m.par.kr*t/m.grid.rb(2)^2.*y.*y/m.par.ss(end)) ...
           ./y./F(y);
t0 = m.time.t(n);
for k = n:length(m.time.t)
    s(k-n+1) = 8*m.par.ss(end)/pi/pi * ...
               quadgk(@(y)G(y,m.time.t(k)-t0),0,Inf);
end

% time-drawdown graph
figure
semilogx(m.time.t,squeeze(m.s(1,1,:)),'k',m.time.t(n:end),s,'r')
```

```
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
xlim([1e-6 1])
```

The initial time interval during which the total head change is realized, is discretized into 30 uniform stress periods of 0.1 second. For each stress period, an initial head change of 1/30 m is defined in the first ring. An additional stress period of 1 day is defined that corresponds to the duration of the slug test with instantaneous head change. The latter is solved analytically using the Cooper et al. (1967) expression. No stresses are defined for the last stress period.

Figure 47 shows the resulting time-drawdown graph for the MAxSym solution as well as for the corresponding analytical solution. Total budget for each time step is `m.totbud(:,1)`, the maximum total budget `max(abs(m.totbud(:,1)))` is equal to 3.8061e-12 m^3/d. The outer model boundary condition is verified by checking `squeeze(m.s(1,end,:))`.

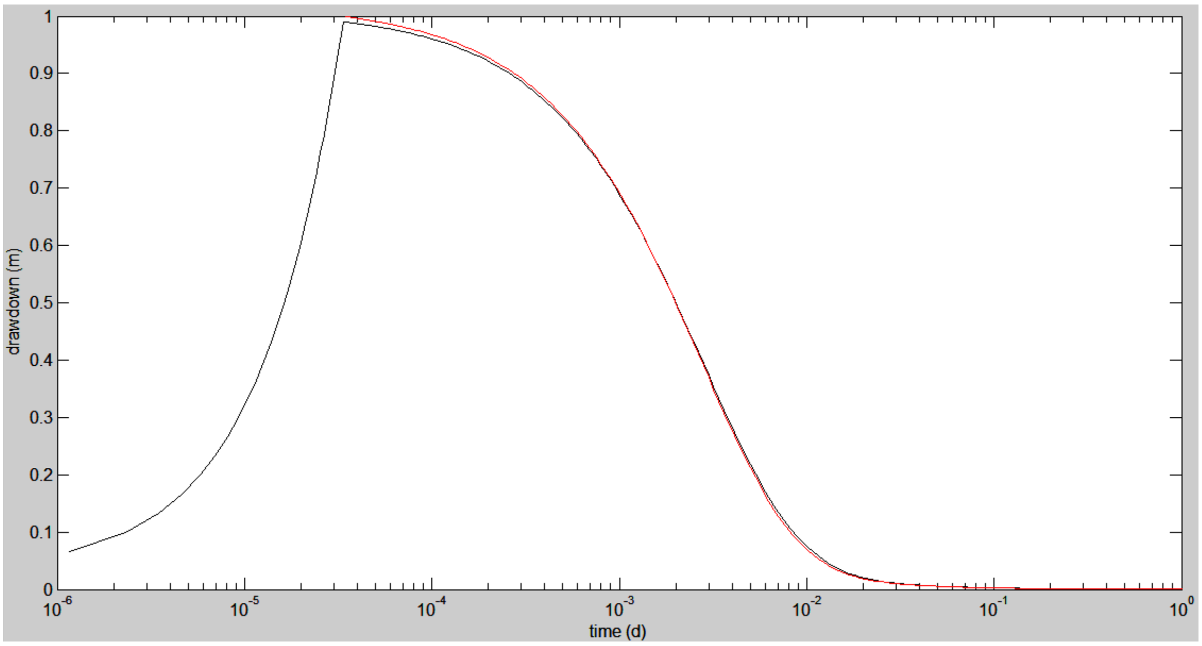


Figure 47. Time-drawdown graph for the slug test with non-instantaneous head change calculated using MAxSym (black) compared to the corresponding analytical Cooper et al. (1967) solution (red).

A final remark concerns the interpolation of drawdown for times within the interval during which the head change is realized. In this case, the head changes defined at the beginning of the stress periods are not instantaneous, hence, method `m.interp1t` will give wrong results, since MAxSym assumes head changes are instantaneous. Suppose we want to know the drawdown within the well after 1.45 seconds. The MAxSym method gives:

```
>> t = 1.45/60/60/24;
>> m.interp1t(1,1,t)

ans =

    0.4973
```

The correct drawdown within the well is calculated using MATLAB function `interp1`:

```
>> interp1(log10(m.time.t),squeeze(m.s(1,1,:)),log10(t))

ans =

    0.5142
```

Slug test in well with finite-thickness skin

Both infinitesimal and finite-thickness well skins are implemented the same way for slug tests as for pumping tests. Semi-analytical solutions exist in the literature (e.g. Yeh and Yang, 2006), but their implementation is based on numerical inversion techniques which are not provided with the standard MATLAB environment. Louwyck et al. (in press) verified the MAxSym model against the semi analytical KGS model (Hyder et al., 1994) for slug tests in a well with finite-thickness skin. In this example, only the MAxSym implementation is given.

Consider the same test as described in previous example: T = 1 m²/d, S = 1e-5, r_w = 0.03 m, and H_0 = 1 m. The initial head change is instantaneous and a well skin is taken into account with outer radius equal to 0.05 m. The skin extent is discretized into 20 rings, including the well ring. The outer grid radius is 1000 m and the total number of rings is 400. A simulation period of 1 day is considered, which is subdivided into 250 time steps. Concerning the well skin transmissivity T_{skin}, five scenarios are defined: T_{skin} = 0.01 m²/d, T_{skin} = 0.1 m²/d, T_{skin} = 1 m²/d, T_{skin} = 10 m²/d, and T_{skin} = 100 m²/d. Script “..\Examples\Transient\slugTestFinThickSkin.m” implements this example:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-6,1,251)));

% set grid
rb = logspace(log10(0.03),log10(0.05),20);
rb = [rb(1)^2/rb(2), rb(1:end-1), logspace(log10(0.05),3,381)];
m.setgrid(rb,1,true);

% set parameters
m.par.kr = ones(1,m.grid.nr);
m.par.ss = [m.grid.rb(2)^2*pi/m.grid.vol(1), ...
            1e-5*ones(1,m.grid.nr-1)];

% set stresses
m.stress.s0 = [1 ,zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,1);

% loop through different kr values for skin
kskin = logspace(-2,2,5);
for n = 1:length(kskin)

    % set kr
    m.par.kr(1:20) = kskin(n);

    % run model
    m.run;
```

```
        % get well drawdown
        s(:,n) = squeeze(m.s(1,1,:));

    end

    % time-drawdown graph
    figure
    semilogx(m.time.t,s)
    set(gca,'fontsize',12)
    xlabel('time (d)')
    ylabel('drawdown (m)')
    xlim([1e-6 1])
    legend(strcat(num2str(kskin','%g'),'m²/d'))
```

Figure 48 shows the resulting time-drawdown graphs.

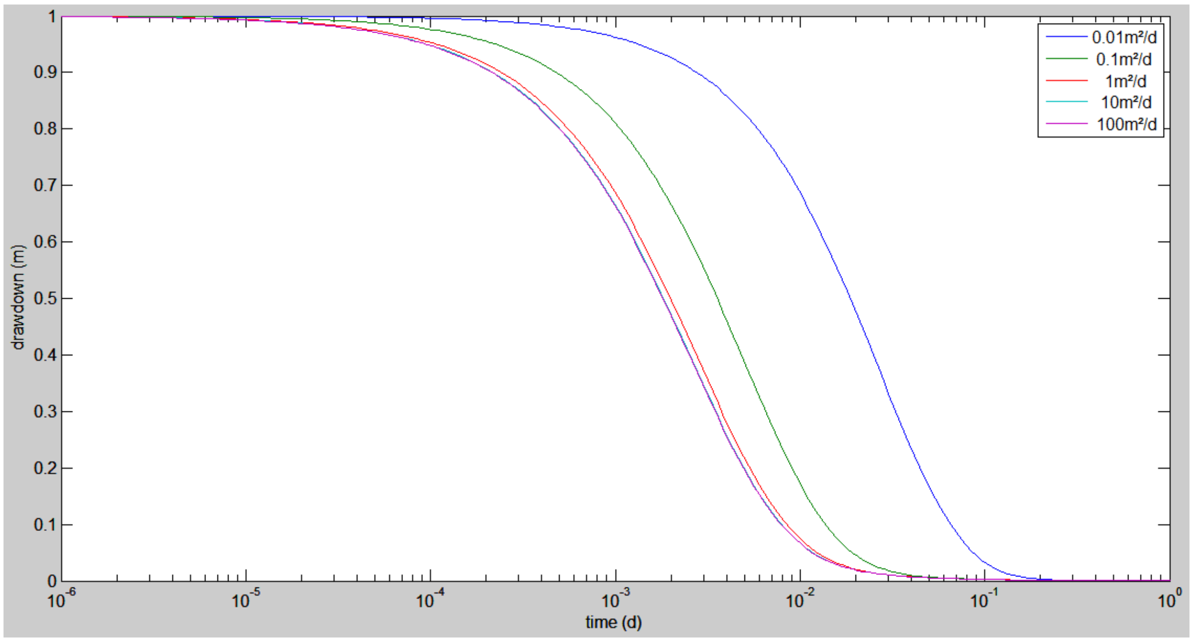


Figure 48. Time-drawdown graphs for the slug test conducted in a well with finite-thickness skin. Five different skin transmissivities are considered (see legend). Results are calculated using MAxSym.

Slug test in partially penetrating well

The Cooper et al. (1967) model assumes a fully penetrating well. When the well is partially penetrating and the aquifer is homogeneous and confined, then the solution may be calculated using the KGS model (Hyder et al., 1994). Louwyck et al. (in press) verified the MAxSym code against the KGS model for slug tests conducted in a partially penetrating well. In this example, only the MAxSym implementation is given.

A confined aquifer is considered with thickness equal to 1 m, horizontal conductivity equal to 1 m/d, vertical conductivity equal to 0.1 m/d, and specific storage coefficient equal to 1e-5 m^{-1}. The partially penetrating well has a radius of 0.03 m, the screen has a length of 0.5 m and is situated in the lower part of the aquifer. The inner grid radius is derived from the well radius, the outer grid radius is set 1000 m, and 400 rings are defined. To obtain an accurate simulation of vertical flow towards the well, the aquifer is subdivided into 10 sub-layers of thickness 0.1 m. A simulation period of 1 day is considered, which is subdivided in 250 time steps.

Script "..\Examples\Transient\slugTestPartWell.m" implements this example:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-6,1,251)));

% set grid
rb = logspace(log10(0.03),3,401);
rb = [rb(1)^2/rb(2), rb];
m.setgrid(rb,0.1*ones(10,1),true);

% set parameters
m.par.inactive = false(m.grid.nz,m.grid.nr);
m.par.inactive(1:5,1) = true;
m.par.kr = 1;
m.par.kz = 0.1;
m.par.cz = [1e-8, zeros(1,m.grid.nr-1)];
m.par.ss = [m.grid.rb(2)^2*pi/m.grid.vol(1)/5, ...
            1e-5*ones(1,m.grid.nr-1)];

% set stresses
m.stress.s0 = [1,zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-6,500,5);

% run model
m.run;

% time-drawdown graph
figure
semilogx(m.time.t,squeeze(m.s(6:10,1,:))')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
xlim([1e-6 1])
```

The screened part of the aquifer contains 5 sub-layers, and therefore, the wellbore storage coefficient `ss` for the first rings corresponding to the well is divided by 5. Also, a very small value of 1e-8 d is assigned to the vertical resistance `cz` for the first rings to obtain the same drawdown in each of these rings. Note that the non-zero `cz` values overwrite the input `kz` values. The well casing in the upper part of the aquifer is taken into account by defining the first rings in the upper sub-layers inactive. The parameters and stresses set for these rings are ignored during the calculation process and consequently, we do not have to define full `nz` x `nr` matrices when setting parameters and stresses.

Figure 49 shows the time-drawdown graph for the 5 first rings corresponding to the well screen. As required, the 5 curves coincide. To obtain this result, it is necessary to define a sufficiently small value for the vertical resistance between the first rings. Also, a smaller criterion for convergence is needed and a larger maximum number of iterations. The maximum total budget is 4.148e-4 m^3/d, which is obtained by evaluating `max(abs(m.totbud(:,1)))`. Outer model boundary conditions are verified by checking `squeeze(m.s(:,end,:))'`.

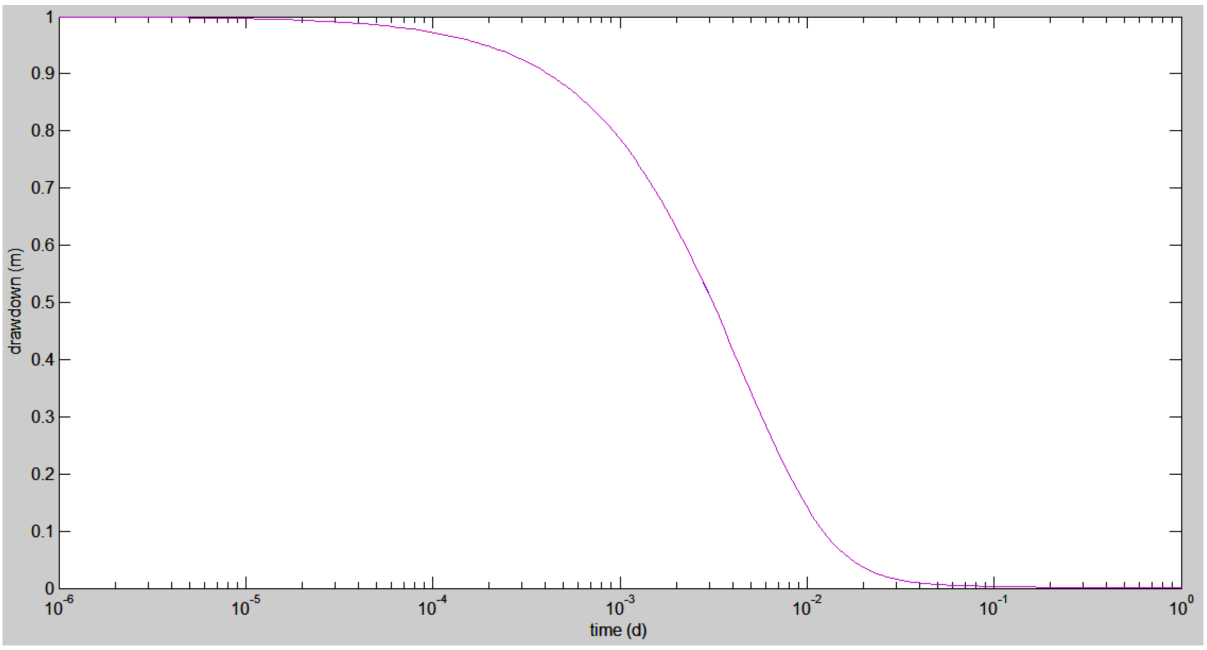


Figure 49. Time-drawdown graph for the slug test conducted in a partially penetrating well calculated using MAxSym. The curves for the 5 well rings coincide.

Slug test in unconfined aquifer

MAxSym may be used to simulate a slug test in an unconfined aquifer if the well screen does not intersect the water table. As is the case for pumping tests, it is possible to simulate a slug test conducted in a well fully penetrating a phreatic layer, but the results will be inaccurate. Therefore, this example presents a case where the well is screened in the lower part of the unconfined aquifer. The semi analytical solution could be obtained by using the KGS model (Hyder et al., 1994), in which the water table is considered as a constant-head boundary. Using MAxSym, the water table is taken into account by specifying the specific yield, but a constant head may be applied as well. In this example, both approaches are implemented and compared to the case where the aquifer is confined.

An unconfined aquifer is considered with thickness equal to 1 m, horizontal conductivity equal to 1 m/d, vertical conductivity equal to 0.1 m/d, specific storage coefficient equal to 1e-5 m^{-1}, and specific yield equal to 0.2. The partially penetrating well has a radius of 0.03 m, the screen has a length of 0.5 m and is situated in the lower part of the aquifer. The inner grid radius is derived from the well radius, the outer grid radius is set 1000 m, and 400 rings are defined. The aquifer is subdivided into 4 sub-layers of thickness 0.25 m. A simulation period of 1 day is considered, which is subdivided into 250 time steps.

Three cases are simulated: 1) the confined case, 2) the unconfined case with constant head, and 3) the unconfined case with specific yield. For each case, a new grid must be defined, which causes the instantiation of a new `Parameters` and `Stresses` object. As a consequence, all parameters and stresses must be re-defined for each case. Script “..\Examples\Transient\slugTestUnconfined.m” implements this example:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-6,1,251)));

% set solver
m.setsolver(1e-6,50,5);

% CONFINED CASE

% set grid
rb = logspace(log10(0.03),3,401);
rb = [rb(1)^2/rb(2),rb];
m.setgrid(rb,0.25*ones(4,1),true);

% set parameters
m.par.inactive = false(m.grid.nz,m.grid.nr);
m.par.inactive(1:2,1) = true;
m.par.kr = 1;
m.par.kz = 0.1;
m.par.cz = [1e-8,zeros(1,m.grid.nr-1)];
m.par.ss = [rb(2)^2*pi/m.grid.vol(1)/2,1e-5*ones(1,m.grid.nr-1)];

% set stresses
m.stress.s0 = [1,zeros(1,m.grid.nr-1)];

% run model
m.run;
s1 = squeeze(m.s(end,1,:));
disp(max(abs(m.totbud(:,1))))

% UNCONFINED CASE - CONSTANT HEAD

% set grid
m.setgrid(rb,[0.25;0.25*ones(4,1)],true);

% set parameters
m.par.constant = [true; false(4,1)];
m.par.inactive = false(m.grid.nz,m.grid.nr);
m.par.inactive(1:3,1) = true;
m.par.kr = 1;
m.par.kz = 0.1;
m.par.cz = [1e-8,zeros(1,m.grid.nr-1)];
m.par.ss = [rb(2)^2*pi/m.grid.vol(1)/2,1e-5*ones(1,m.grid.nr-1)];

% set stresses
m.stress.s0 = [1,zeros(1,m.grid.nr-1)];

% run model
m.run;
s2 = squeeze(m.s(end,1,:));
disp(max(abs(m.totbud(:,1))))
```

```
% UNCONFINED CASE - SPECIFIC YIELD

% set grid
m.setgrid(rb,0.25*ones(4,1),false);

% set parameters
m.par.inactive = false(m.grid.nz,m.grid.nr);
m.par.inactive(1:2,1) = true;
m.par.kr = 1;
m.par.kz = 0.1;
m.par.cz = [1e-8,zeros(1,m.grid.nr-1)];
m.par.ss = [rb(2)^2*pi/m.grid.vol(1)/2,1e-5*ones(1,m.grid.nr-1)];
m.par.sy = 0.2;

% set stresses
m.stress.s0 = [1,zeros(1,m.grid.nr-1)];

% run model
m.run;
s3 = squeeze(m.s(end,1,:));
disp(max(abs(m.totbud(:,1))))


% time-drawdown graph
figure
semilogx(m.time.t,[s1,s2,s3])
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
xlim([1e-6 1])
legend('confined','constant head','specific yield')
```

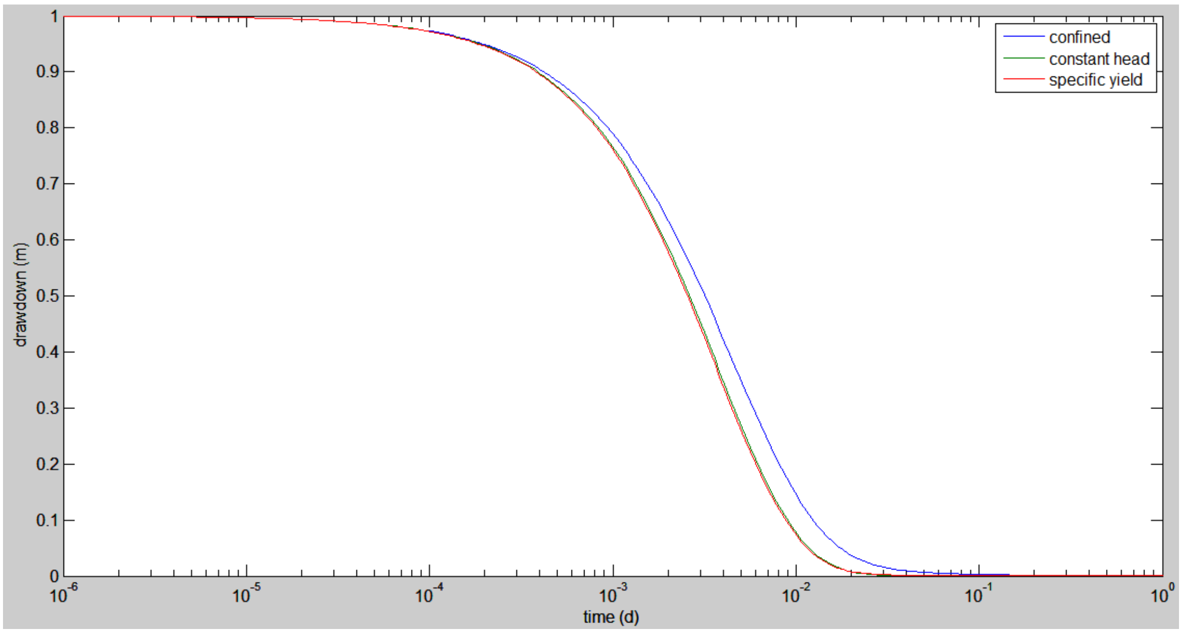


Figure 50. Time-drawdown graph for the slug test conducted in a partially penetrating well located in an unconfined aquifer. The water table is considered as constant head or specific yield. Results are calculated using MAxSym and compared to the case where the aquifer is confined.

The time-drawdown plot comparing the results for the three cases is shown in Figure 50. The maximum total budget for each case is displayed after each model run. The budget error could be reduced by specifying a smaller criterion for convergence and a larger maximum number of iterations. Additionally, the outer grid boundary conditions could have been verified by displaying `max(max(squeeze(m.s(:,end,:))))` after each model run.

The screened part of the aquifer contains 2 sub-layers, and therefore, the wellbore storage coefficient `ss` for the first rings representing the well is divided by 2. Also, a very small value of 1e-8 d is assigned to the vertical resistance `cz` for the first rings. The well casing in the upper part of the aquifer is taken into account by defining the first rings in the upper sub-layers inactive. In the second case, the water table is defined as a constant-head layer on top of the model grid and it is required to adopt the thickness and vertical conductivity of the aquifer layers to obtain the same time-drawdown graph as the specific yield case. The values for the horizontal conductivity and the specific storage coefficient are irrelevant for this layer.

Slug interference test in multi-layer system

Using MAxSym, it is possible to simulate a slug interference test conducted in a well that is located in a multi-layer system. The well may be partially penetrating and/or be surrounded by a skin. Drawdown is measured in the slugged well and additionally, in one or more observation wells.

In this example, we consider a phreatic system consisting of two layers separated by a thin clay layer with vertical resistance of 10 d. The characteristics of the layers are:

- Layer 1: D = 2 m, K^r = 5 m/d, K^z = 1 m/d, S^s = 1e-5 m^{-1}, S^y = 0.2
- Layer 2: D = 4 m, K^r = 1 m/d, K^z = 0.1 m/d, S^s = 1e-5 m^{-1}

where D is the layer thickness, K^r is the radial conductivity, K^z is the vertical conductivity, S^s is the specific storage coefficient, and S^y is the specific yield. The slugged well of radius 0.03 m is present in layer 2 and the top of the well screen with length 1 m is located 1 m below the bottom of layer 1. The well casing extends from the top of layer 1 to the bottom of layer 2, and a well skin is present with infinitesimal thickness and radial resistance equal to 0.01 d. The initial head change within the well is instantaneous and equal to 2 m.

The model discretization is as follows. The simulation period of 1 d is discretized into 250 time steps. The top layer corresponds to the first model layer and the non-screened parts of the bottom layer are discretized into sub-layers to obtain an accurate simulation of the vertical flow. In total, 7 model layers are defined. The clay layer between the top and bottom layer is taken into account by assigning its resistance to the vertical resistance `cz` between model layer 1 and 2. The inner grid radius is derived from the well radius. The outer grid radius is equal to 1000 m and 401 rings are defined. The first rings corresponding to the non-screened parts of the well are set inactive. The infinitesimal well skin is considered by assigning its resistance to the radial resistance `cr` between the first and second ring. The wellbore storage term is assigned to the storage coefficient `ss` of the first ring.

Script "..\Examples\Transient\slugTestMultiLayer.m" implements this example:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-6,1,251)));

% set grid
rb = logspace(log10(0.03),3,401);
rb = [rb(1)^2/rb(2), rb];
m.setgrid(rb,[2;0.5;0.5;1;0.5;0.5;1],false);

% set parameters
m.par.inactive = false(m.grid.nz,m.grid.nr);
m.par.inactive([1:3 5:7],1) = true;
m.par.kr = [5;ones(6,1)];
m.par.cr = [0.01,zeros(1,m.grid.nr-2)];
m.par.kz = [1;0.1*ones(6,1)];
m.par.cz = [10;zeros(5,1)];
m.par.ss = [m.grid.rb(2)^2*pi/m.grid.vol(1), ...
            1e-5*ones(1,m.grid.nr-1)];
m.par.sy = 0.2;

% set stresses
m.stress.s0 = sparse(m.grid.nz,m.grid.nr);
m.stress.s0(4,1) = 2;

% set solver
m.setsolver(1e-6,20,5);

% run model
m.run;

% drawdown at 1m
s1 = squeeze(m.interp1r(4,1,[]));

% time-drawdown graph
figure
semilogx(m.time.t,[squeeze(m.s(4,1,:)),s1],'-')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
xlim([1e-6 0.1])
legend('slugged well','1.00m')
```

The resulting time-drawdown plot is shown in Figure 51. Besides the head change within the well, drawdown is also plotted for an observation well at 1 m in model layer 4, i.e. the layer containing the well screen. Drawdown at 1 m is calculated using method `m.interp1r`, which applies linear interpolation in the dimension of *r*.

Total volumetric budget at the end of each time step is `m.totbud(:,1)`, maximam total budget `max(abs(m.totbud(:,1)))` is 7.1595e-5 m^3/d. Outer grid boundary conditions are verified by evaluating `max(max(m.s(:,end,:)))`, which yields 6.0030e-41 m. Both values are close to zero, so the calculated results are considered accurate.

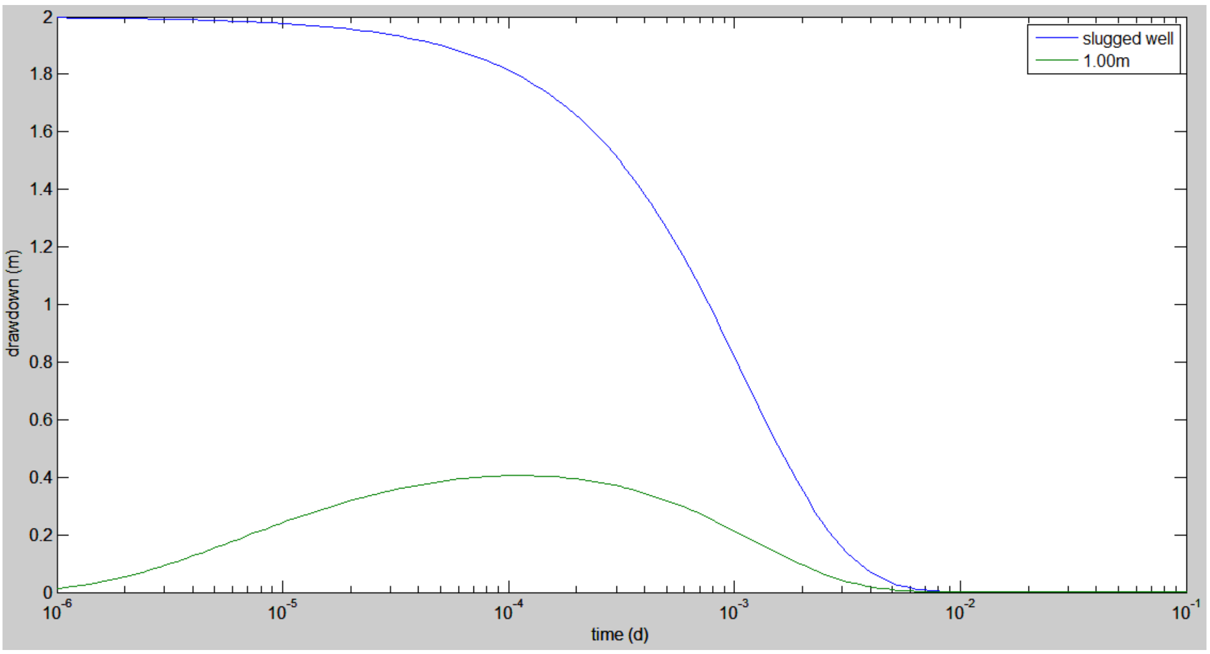


Figure 51. Time-drawdown graph for the slug interference test conducted in a phreatic multi-layer system calculated using MAxSym. The observation well at 1 m is located at the same level of the well screen.

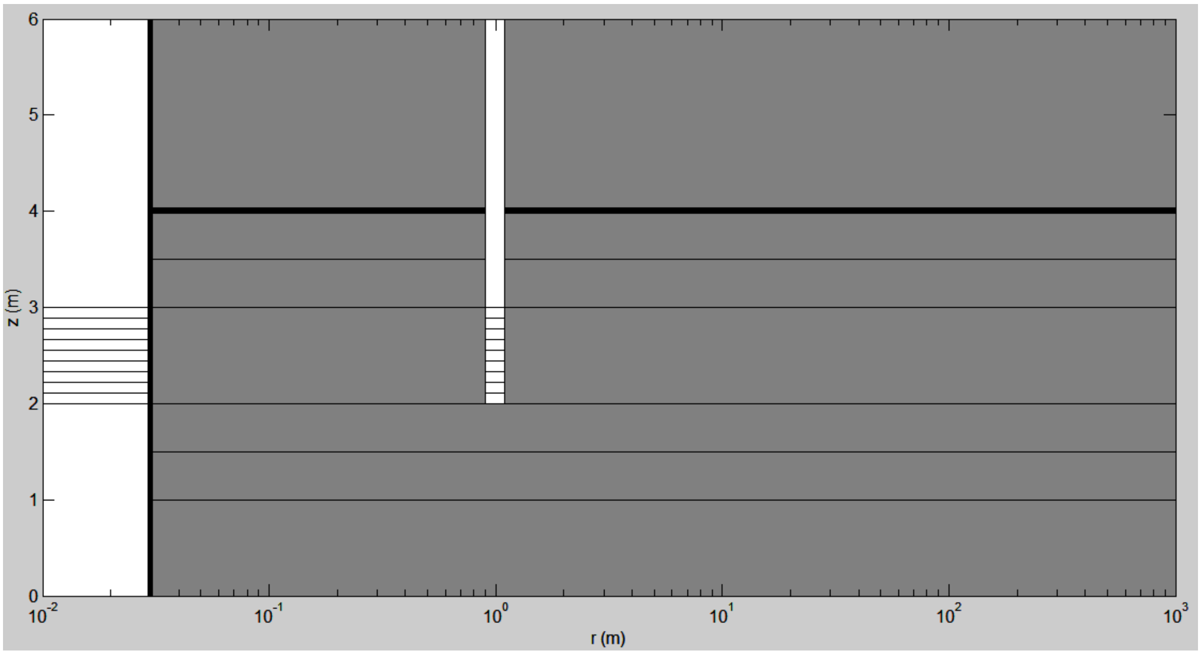


Figure 52. Model setup for the slug test conducted in a phreatic multi-layer system. The white zone corresponds to the well, the gray zone is the aquifer. The well skin and the thin clay layer are indicated by a bold black line. Well screen and sub-layers are also plotted, as is the observation well at 1 m. Note that the observation well is assumed to have an infinitesimal width in the MAxSym model, so it does not influence the radial flow in the aquifer system.

Finally, a plot of the model setup is shown in Figure 52, which is created by the following statements:

```
    % plot model set up
    figure
    x = [0.01 m.grid.rb(end)];
```

```
y = m.grid.zb';
set(axes('parent',gcf),'xlim',x,'xscale','log',...
    'ylim',y([end 1]),'box','on','layer','top','fontsize',12)
xlabel('r (m)')
ylabel('z (m)')

% layers and sub-layers
patch([x fliplr(x)],y([1 1 end end]),0.5*ones(1,3),'parent',gca)
rw = m.grid.rb(2);
hl = line([rw,x(end)],[y;y],'color','k','parent',gca);
set(hl(2),'linewidth',5) % clay layer

% well
patch([x(1),rw,rw,x(1)],y([1 1 end end]),'w','parent',gca)
line([rw rw],y([1 end]),'color','k','linewidth',4,'parent',gca) % skin
y = linspace(y(4),y(5),10);
line([x(1),rw],[y;y],'color','k','parent',gca) % screen

% observation well
dx = 0.1;
patch(1+dx*[-1,1,1,-1],...
      [y([end end]),m.grid.zb([1 1])'],'w','parent',gca)
line([1-dx,1+dx],[y;y],'color','k','parent',gca) % screen
```

Combined injection and slug test in confined aquifer

In some cases, it could be useful to simulate a slug test conducted in a well that is at the same time extracted or injected. For a confined, homogeneous aquifer with fully penetrating well, the analytical solution is obtained through superposition of the Theis (1935) and the Cooper et al. (1967) solution given above. Using MAxSym, the solution is calculated directly by specifying a discharge as well as an initial head change for the ring representing the well.

Consider, for example, a confined aquifer with transmissivity of 1 m^2/d and storativity of 1e-5, in which a fully penetrating well of radius 0.03 m is present. Water is injected into the well at a constant rate of 0.5 m^3/d. After one day, an instantaneous head rise of 2 m is realized in the same well.

The axi-symmetric grid contains one layer, the inner grid radius is derived from the well radius, the outer grid radius is set 1e6 m, and 801 rings are considered including the first ring representing the well. The layer thickness is set 1 m, so transmissivity and storativity can be assigned directly to `m.par.kr` and `m.par.ss` respectively. The storage coefficient for the first ring is set to account for the wellbore storage. Finally, two stress periods of 1 day are defined, each of them subdivided into 300 time steps. For the first period, only a discharge of -0.5 m^3/d is set for the first ring, for the second period, an initial head change of 2 m is set additionally. The following statements to set up and run the model are implemented in script "..\Examples\Transient\slugTestPlusInjection.m":

```
 % create Model object
m = MAxSym.Model;

% set time steps
n = 300;
m.settime(repmat(diff(logspace(-5,0,n+1)),1,2),[n n]);
```

```
% set grid
rb = logspace(log10(0.03),6,801);
rb = [rb(1)^2/rb(2), rb];
m.setgrid(rb,1,true);

% set parameters
m.par.kr = 1;
m.par.ss = [m.grid.rb(2)^2*pi/m.grid.vol(1), ...
            1e-5*ones(1,m.grid.nr-1)];

% set stresses
m.stress(1).q = [-0.5, zeros(1,m.grid.nr-1)];
m.stress(2).q = m.stress(1).q;
m.stress(2).s0 = [1, zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,1);

% run model
m.run;

% analytical solution (Theis)
rw = m.grid.rb(2);
t = m.time.t;
u = rw^2./t * m.par.ss(end)/m.par.kr/4;
s = -m.stress(1).q(1)/4/pi/m.par.kr * expint(u);

% analytical solution (Cooper et al.)
F = @(y) (y.*besselj(0,y) - 2*m.par.ss(end)*besselj(1,y)).^2 + ...
         (y.*bessely(0,y) - 2*m.par.ss(end)*bessely(1,y)).^2;
G = @(y,t) exp(-m.par.kr*t/rw^2.*y.*y/m.par.ss(end)) ./y./F(y);
t = m.time.t(n+1:end)-m.time.t(n+1);
for i = 1:length(t)
    s(n+i) = s(n+i) + 8*m.stress(2).s0(1)*m.par.ss(end)/pi/pi * ...
    quadgk(@(y)G(y,t(i)),0,Inf);
end

% time-drawdown graph
figure
semilogx(t,squeeze(m.s(1,1,n+1:end))','k')
hold on
semilogx(t,s(n+1:end),'kx')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
xlim([1e-6 1])
```

The resulting plot comparing the MAxSym solution with the analytical solution is given in Figure 53. It shows the time-drawdown curve for the well starting from the moment the initial head change was realized.

Note that the system is still one-dimensional and linear, hence, it can be solved directly by setting the maximum number of iterations equal to 1. Total volumetric budget for each time step is `m.totbud(:,1)`, maximam total budget `max(abs(m.totbud(:,1)))` is 1.2165e-11m^3/d. Outer grid boundary conditions are verified by evaluating `min(m.s(1,end,:))`.

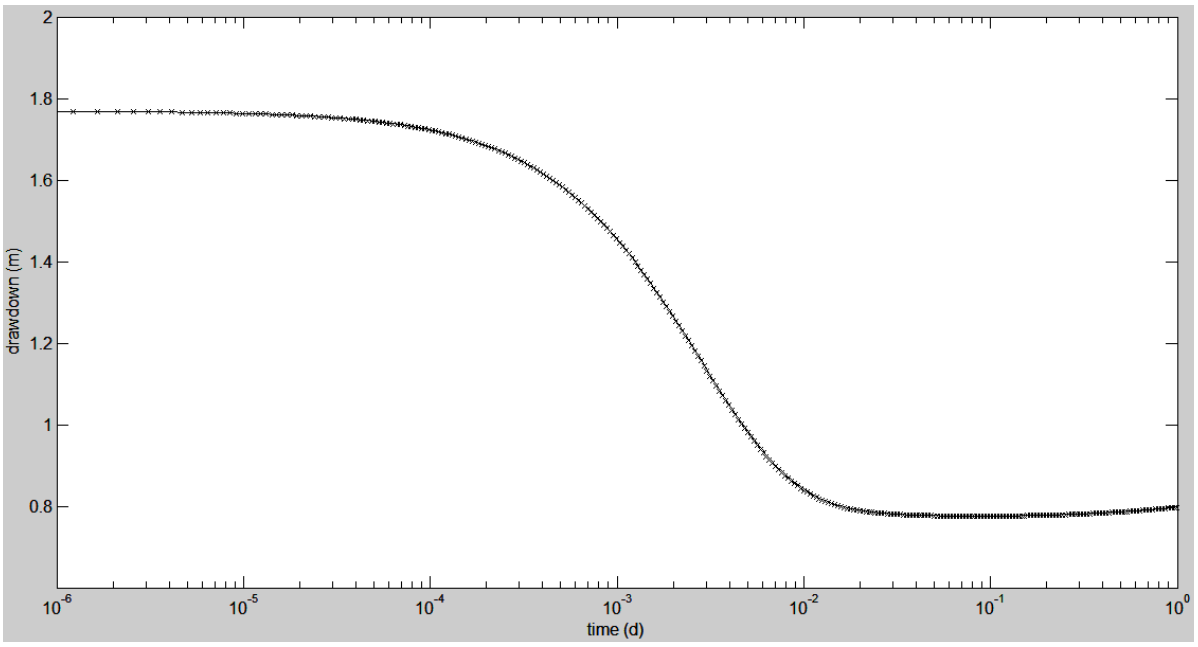


Figure 53. Time-drawdown graph for the combined injection and slug test in a confined aquifer. Results are calculated analytically (crosses) and using MAxSym (solid lines).

Hydraulic tomography in multi-layer system

Using MAxSym, it is straightforward to simulate and analyze multi-level aquifer tests such as multiple pumping tests and multi-level slug tests. As a first example, a hydraulic tomography performed in a multi-layer system is simulated. The test configuration consists of one extraction well and one or more observation wells, which are partitioned into several depth intervals using packers. Sequential short-time pumping tests are conducted at different intervals in the extraction well. During each test, drawdown is monitored at other intervals in the extraction well and at different intervals in the observation well(s).

To simulate a hydraulic tomography test using MAxSym, two approaches are possible. The sequential tests may be simulated during one model run, where different stress periods are defined for the different tests and the subsequent recovery phases. Alternatively, a model may be set up for each sequential test and the models are run sequentially using a `for` loop. The advantage of the latter approach is it is not required to consider the recovery phases. In this example, the first approach is applied and a `for` loop is used to specify the different stress periods that correspond to the sequential tests and recovery phases.

Consider a confined three-layer system where the layers have the following characteristics:

- upper layer: D = 1 m, K^r = 10 m/d, K^z = 5 m/d, S^s = 1e-4 m^{-1};
- middle layer: D = 1 m, K^r = 2 m/d, K^z = 0.5 m/d, S^s = 1e-5 m^{-1};
- lower layer: D = 1 m, K^r = 5 m/d, K^z = 2 m/d, S^s = 5e-5 m^{-1};

where D denotes thickness, K^r horizontal conductivity, K^z vertical conductivity, and S^s specific storage. Three sequential tests are conducted in an extraction well with radius of 0.05 m and skin of 0.02 m width. The conductivity of the skin is 50 m/d in both radial and vertical direction and its specific storage coefficient is 1e-6 m^{-1}. During the i-th test, the i-th layer is extracted at a rate of 10 m³/d. Each test lasts 20 minutes and is followed by a recovery phase of 100 minutes before the next test is started. Drawdown is measured at the extraction well and at an observation well at different depth intervals corresponding to the different layers.

The MAxSym model simulating this hydraulic tomography test has 6 stress periods defined, one for each pumping test and one for each recovery phase. Each stress period corresponding to one of the pumping tests is discretized into 100 time steps, the subsequent period of recovery is discretized into 500 time steps. The model grid consists of three layers. The inner model radius is derived from the well radius. The zone representing the well skin is discretized into 10 rings, and the zone representing the aquifer has an outer radius of 1000 m and is discretized into 240 rings. The parameter input adopts the hydraulic properties of aquifer layers and well skin. The storage coefficients for the first rings are set to account for the wellbore storage. The packers partitioning the extraction well are taken into account by specifying vertical resistances of large value between the first rings.

Script "..\Examples\Transient\hydraulicTomography.m" implements this example:

```
% create Model object
m = MAxSym.Model;

% set time steps
dtpump = diff(logspace(-6,log10(20/1440),101));
dtrecov = diff(logspace(-6,log10(100/1440),501));
m.settime(repmat([dtpump,dtrecov],1,3),repmat([100 500],1,3));

% set grid
rb = logspace(log10(0.05),log10(0.07),10);
rb = [rb(1)^2/rb(2), rb];
rb = [rb(1:end-1),logspace(log10(0.07),3,241)];
m.setgrid(rb,ones(3,1),true);

% set parameters
m.par.kr = repmat([10;2;5],1,m.grid.nr);
m.par.kr(:,1:10) = 50;
m.par.kz = repmat([5;0.5;2],1,m.grid.nr);
m.par.kz(:,1:10) = 50;
m.par.cz = sparse(m.grid.nz-1,m.grid.nr);
m.par.cz(:,1) = 1e5;
m.par.ss = repmat([1e-4;1e-5;5e-5],1,m.grid.nr);
m.par.ss(:,2:10) = 1e-6;
m.par.ss(:,1) = m.grid.rb(2)^2*pi./m.grid.vol(:,1);

% set stresses
for n = 1:2:m.time.nper
    m.stress(n).q = sparse(m.grid.nz,m.grid.nr);
    m.stress(n).q((n+1)/2,1) = 10;
end

% set solver
m.setsolver(1e-6,20,5);

% run model
m.run;
```

```
% time-drawdown graphs
figure
ir = [1 89];
for n = 1:length(ir)
    subplot(1,length(ir),n)
    plot(m.time.t*24,squeeze(m.s(:,ir(n),:))')
    set(gca,'fontsize',12)
    xlabel('time (h)')
    ylabel('drawdown (m)')
    if n == 1
        title('extraction well')
    else
        title(['obs.well at ',num2str(m.grid.r(ir(n)),'%.2f'),'m'])
    end
    legend(strcat('layer',num2str((1:m.grid.nz)')),...
           'location','southeast')
end
```

The resulting time-drawdown graphs for the different depth intervals at the extraction well and at an observation well at 1.6 m are shown in Figure 54. The maximum number of iterations the SIP solver needs to solve the system of equations is given by `max(m.niter)`. The maximum total budget is 4.2910e-4 m^3/d, which is obtained from `max(abs(m.totbud(:,1)))`. Outer grid boundary conditions are verified by checking `squeeze(m.s(:,end,:))'`.

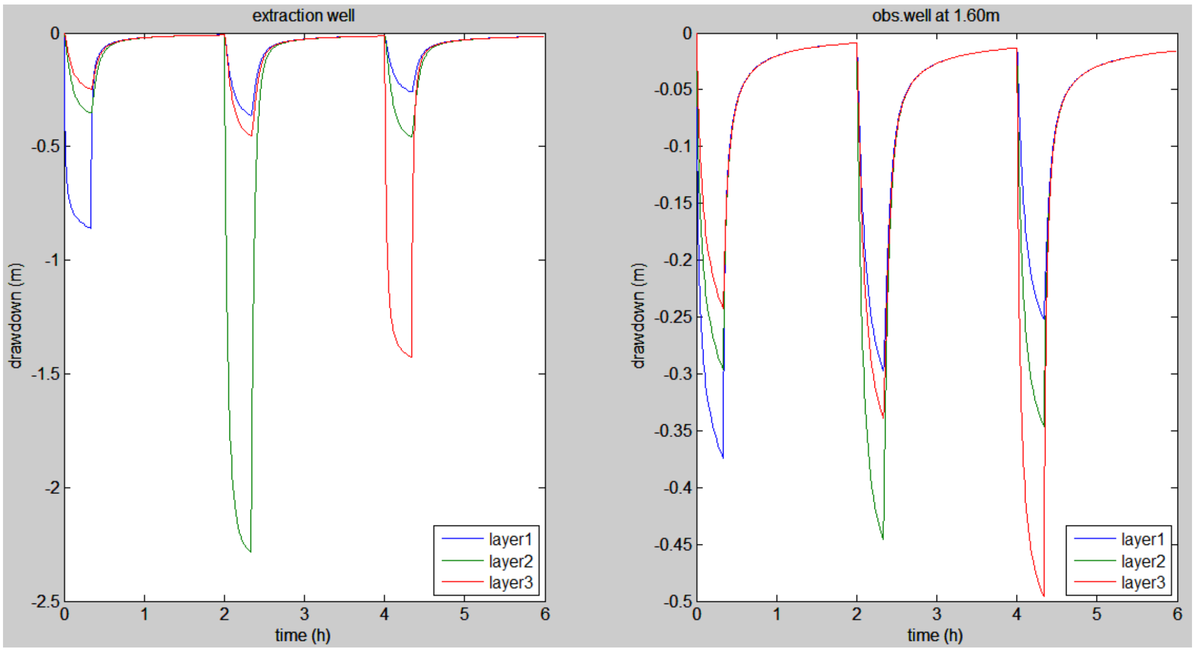


Figure 54. Time-drawdown graphs for the different depth intervals at the extraction well and at an observation well at 1.6 m for the hydraulic tomography test simulated using MAxSym.

Multi-level slug testing in multi-layer system

The idea behind multi-level slug testing is the same as for the hydraulic tomography test discussed in previous example. Simulating multi-level slug testing using MAxSym is also similar to the simulation of a hydraulic tomography test.

Consider, for instance, the confined three-layer aquifer described in previous example:

- upper layer: D = 1 m, K^r = 10 m/d, K^z = 5 m/d, S^s = 1e-4 m^{-1};
- middle layer: D = 1 m, K^r = 2 m/d, K^z = 0.5 m/d, S^s = 1e-5 m^{-1};
- lower layer: D = 1 m, K^r = 5 m/d, K^z = 2 m/d, S^s = 5e-5 m^{-1};

where D denotes thickness, K^r horizontal conductivity, K^z vertical conductivity, and S^s specific storage. Three sequential slug tests are conducted at a well with radius of 0.05 m and skin of 0.02 m width. The hydraulic properties of the skin are the same as in previous example: conductivity is 50 m/d in both radial and vertical direction and specific storage coefficient is 1e-6 m^{-1}. At the start of the *i*-th test, an amount of water is injected instantaneously into the well at the depth interval corresponding to the *i*-th layer, which causes an initial head change of 1 m. During 30 minutes, drawdown is measured at the extraction well and at an observation well at different depth intervals corresponding to the different layers. After 30 minutes, the next test is started.

The MAxSym model simulating this multi-level slug testing has 3 stress periods defined, one for each test. Each stress period is discretized into 500 time steps. The model grid consists of three layers. The inner model radius is derived from the well radius. The zone representing the well skin is discretized into 50 rings, and the zone representing the aquifer has an outer radius of 1000 m and is discretized into 950 rings. The parameter input adopts the hydraulic properties of aquifer layers and well skin. The storage coefficients for the first rings are set to account for the wellbore storage. The packers partitioning the extraction well are taken into account by specifying vertical resistances of large value between the first rings. Script "..\Examples\Transient\multilevelSlugTest.m" implements this example:

```
% create Model object
m = MAxSym.Model;

% set time steps
dt = diff(logspace(-6,log10(30/1440),501));
m.settime(repmat(dt,1,3),repmat(500,1,3));

% set grid
rb = logspace(log10(0.05),log10(0.07),50);
rb = [rb(1)^2/rb(2), rb];
rb = [rb(1:end-1),logspace(log10(0.07),3,951)];
m.setgrid(rb,ones(3,1),true);

% set parameters
m.par.kr = repmat([10;2;5],1,m.grid.nr);
m.par.kr(:,1:50) = 50;
m.par.kz = repmat([5;0.5;2],1,m.grid.nr);
m.par.kz(:,1:50) = 50;
m.par.cz = sparse(m.grid.nz-1,m.grid.nr);
m.par.cz(:,1) = 1e5;
m.par.ss = repmat([1e-4;1e-5;5e-5],1,m.grid.nr);
m.par.ss(:,2:50) = 1e-6;
m.par.ss(:,1) = m.grid.rb(2)^2*pi./m.grid.vol(:,1);

% set stresses
for n = 1:m.time.nper
    m.stress(n).s0 = sparse(m.grid.nz,m.grid.nr);
    m.stress(n).s0(n,1) = 1;
end
```

```
% set solver
m.setsolver(1e-7,20,5);

% run model
m.run;

% time-drawdown graphs
figure
ir = [1 361];
for n = 1:length(ir)
    subplot(1,length(ir),n)
    plot(m.time.t*24,squeeze(m.s(:,ir(n),:))')
    set(gca,'fontsize',12)
    xlabel('time (h)')
    ylabel('drawdown (m)')
    if n == 1
        title('slugged well')
    else
        title(['obs.well at ',num2str(m.grid.r(ir(n)),'%.2f'),'m'])
    end
    legend(strcat('layer',num2str((1:m.grid.nz)')),...
           'location','northeast')
end
```

The resulting time-drawdown graphs for the different depth intervals at the slugged well and at an observation well at 1.6 m are shown in Figure 55. The maximum number of iterations the SIP solver needs to solve the system of equations is given by `max(m.niter)`. The maximum total budget is 5.5981e-5 m^3/d, which is obtained from `max(abs(m.totbud(:,1)))`. Outer grid boundary conditions are verified by checking `squeeze(m.s(:,end,:))'`.

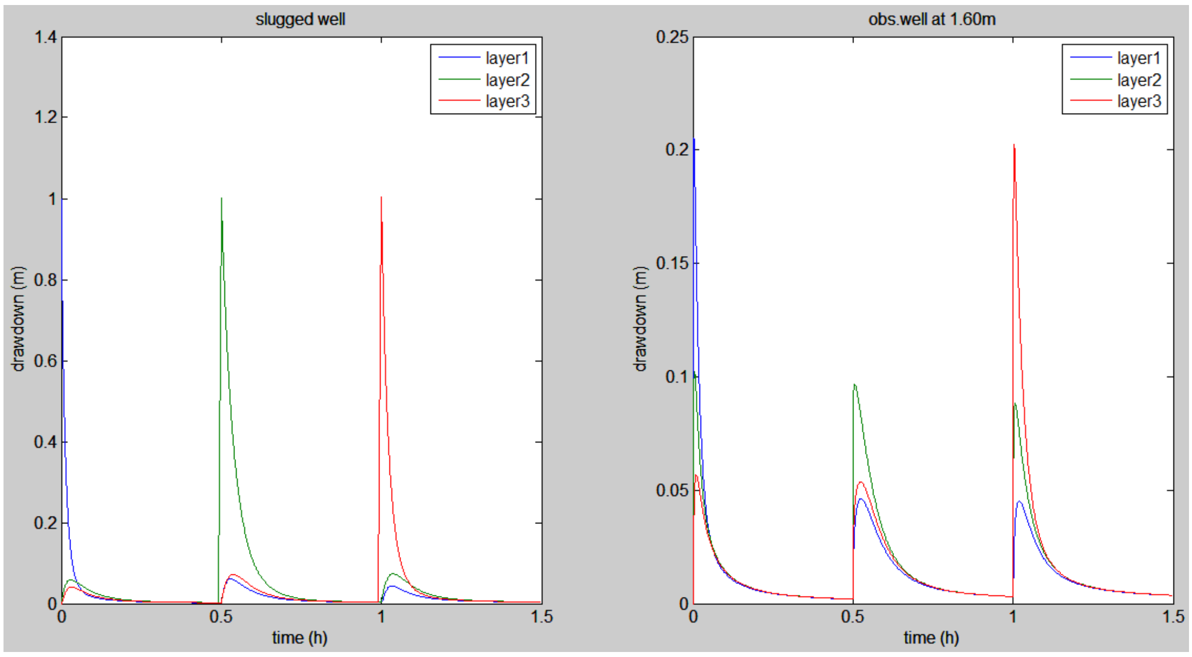


Figure 55. Time-drawdown graphs for the different depth intervals at the slugged well and at an observation well at 1.6 m for the multi-level slug test simulated using MAxSym.

Superposition models

All previous examples presented one- or two-dimensional axi-symmetric models to simulate flow towards or away from a well. By applying superposition, however, it is possible to construct simple three-dimensional models for multiple wells located in aquifer systems with horizontal, homogeneous layers of infinite lateral extent. An important difference between the two types of models concerns the coordinate system: the solution of an axi-symmetric model depends on a polar coordinate system, whereas the superposition model uses a Cartesian coordinate system.

If the model is steady state, the aquifer system must be bounded by a top and/or bottom layer with constant zero drawdown. If the model is transient, the aquifer system may be bounded by an impervious top and bottom layer. Building steady state models for multi-layer systems confined by two impervious layers is also possible, but the superposition of solutions is much more involved in this case, and therefore, out of the scope of this manual.

Theoretically, it is assumed each layer has a constant saturated thickness which does not vary as a function of time. In practice, a phreatic top layer is allowed if maximum drawdown in the top layer is much smaller than its minimum saturated thickness. Lateral anisotropy may be considered only if anisotropy properties are the same for all layers.

A three-dimensional model is obtained by superimposing the individual solutions for wells extracted or injected with constant discharge at different locations in the aquifer system. If the model is transient, the solution for a well with instantaneous head change is also allowed. If the model is steady state, superposition of head specified wells is possible, but much more involved, and therefore, out of the scope of this manual.

Mathematically, the superposition principle is expressed as (see, for example, Lebbe, 1999):

$$s(x,y,i,t) = \sum_{j=1}^{n} \Delta_j(x_p,y_p,k)\ s_u^k(r,\theta,i,t-t_p)$$

where *s(x,y,i,t)* is the drawdown at point *(x,y)* in layer *i* at time *t*, *n* is the number of superimposed wells with either a discharge change or a head change specified, $\Delta_j(x_p,y_p,k)$ is the discharge change or head change starting at time t_p for the *j*-th well located at point (x_p,y_p) in layer *k*, and $s_u^k(r,\vartheta,i,t)$ is the drawdown at point *(r, ϑ)* in layer *i* at time *t* according to the axi-symmetric model with the same layer discretization and parameters and a well located in layer *k* with unit discharge or initial head change. Note that each discharge change or head change for a certain well is conceptualized as a new well at the same location.

Mostly, distance *r* and time *t* do not coincide with nodal circles and simulation times defined for the axi-symmetric model, and therefore, bilinear interpolation is applied to calculate $s_u(r,\vartheta,i,t)$. If distance *r* lies inside the first nodal circle, drawdown at the latter is taken. In case of lateral anisotropy, the polar coordinates are converted first to apparent distances, which are used to derive drawdown from the corresponding axi-symmetric model with isotropic, effective conductivity. The polar coordinates *(r, ϑ)* are calculated as:

$$\begin{cases} r = \sqrt{(x - x_p)^2 + (y - y_p)^2} \\ \theta = atan2(y - y_p, x - x_p) \end{cases}$$

where *atan2* is the arctangent function that takes into account the quadrant, and that is evaluated in MATLAB using the function with the same name. The polar angle ϑ is not considered in case of a lateral isotropic aquifer system, because the flow properties have no angular variation.

It is assumed the wells have an infinitesimal small diameter, and flow in the axi-symmetric model(s) must be uniform, hence, finite-thickness well skins are not allowed. In practice, the well diameter should be small, and an infinitesimal skin and/or non-linear well loss is allowed since they cause a supplementary drawdown that is superimposed anyway.

Additionally, drawdown due to a circular infiltration area may be superimposed (Bakker and Strack, 2003), which is calculated by running a MAxSym model with areal infiltration only. If the center of the infiltration area has Cartesian coordinates (x_r,y_r), then conversion to polar coordinates is done as follows:

$$\begin{cases} r = \sqrt{(x - x_r)^2 + (y - y_r)^2} \\ \theta = atan2(y - y_r, x - x_r) \end{cases}$$

Finally, it is possible to consider lateral no-flow or constant-head boundaries by adding imaginary wells to the model. The use of imaginary wells is called the method of images and it can be applied if the aquifer system is laterally isotropic. In case of a laterally anisotropic system, the method is applicable only if the principal direction is parallel to the considered bounds.

The use of superposition and the method of images is explained in most hydrogeology text books (e.g. Haitjema, 1995; Domenico and Schwartz, 1998).

Forced gradient test in confined aquifer

A forced gradient test consists of two wells between which groundwater flow is induced by injecting water into one well while pumping the same amount of water per unit of time from the other well. If both wells are located in the same model layer, only one axi-symmetric model run is needed to build the three-dimensional model, and total drawdown is obtained by superimposing drawdown due to injection and extraction:

$$s(x,y,i,t) = Q\left[s_u^k(\sqrt{(x - x_p)^2 + (y - y_p)^2}, i, t) - s_u^k(\sqrt{(x - x_i)^2 + (y - y_i)^2}, i, t)\right]$$

where $s(x,y,i,t)$ is the drawdown at point (x,y) in layer i at time t after starting the test, and $s_u^k(r,i,t)$ is the drawdown at distance r according to the corresponding axi-symmetric model with unit discharge. The extraction well is located at (x_p,y_p) in layer k and the injection well is located at (x_i,y_i) in the same layer. The extraction and injection rates are constant and equal to Q and $-Q$ respectively. The aquifer system is assumed laterally isotropic.

As an example, consider a confined, homogeneous aquifer with thickness of 10 m, horizontal conductivity of 10 m/d, and specific storage coefficient of 1e-4 m^{-1}. Two fully penetrating wells are located in the aquifer and the distance between both wells is 50 m. A forced gradient test is performed for a period of 10 days. The injection and extraction rates are -100 m³/d and 100 m³/d respectively. First, a MAxSym model is run with unit discharge. The following statements are found in script "..\Examples\Superposition\forcedGradientTest.m":

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,1,61)));

% set grid
m.setgrid(logspace(-1,7,81),10,true);

% set parameters
m.par.kr = 10;
m.par.ss = 1e-4;

% set stresses
m.stress.q = [1, zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,1);

% run model
m.run;
```

The calculated drawdown stored in `m.s` is used now to simulate the forced gradient test by superimposing drawdown due to injection and extraction. For instance, drawdown within the extraction well as a function of time is calculated as follows:

```
% time-drawdown graph for extraction well
figure
sw = 100 * (m.s(1,1,:) - m.interp1r(1,50,[]));
semilogx(m.time.t,squeeze(sw))
set(gca,'fontsize',12)
xlabel('time (min)')
ylabel('drawdown (m)')
```

The resulting time-drawdown graph is shown in Figure 56. Method `m.interp1` is used to interpolate drawdown at a distance of 50 m. This method always returns a 3D array, and therefore, the output drawdown array must be 'squeezed' to a vector using function `squeeze` before it is passed to the `plot` function.

Drawdown within the injection well is easily obtained by multiplying drawdown within the extraction well by -1. This symmetry can be verified by making a contour plot of drawdown. Consider, for instance, a raster of 100 m x 100 m with origin (0,0) at the lower left corner. The *(x,y)* coordinate of the extraction well is (50,25) and the injection well is located at (50,75). The contour plot of drawdown at the end of the test is created as follows:

```
% 2D raster and well positions
[x,y] = meshgrid(1:100);
[xp,yp] = deal(50,25);
```

```
[xi,yi] = deal(50,75);

% interpolation
rp = sqrt((x-xp).^2+(y-yp).^2);
sp = squeeze(m.interp1r(1,rp(:),length(m.time.t)));
ri = sqrt((x-xi).^2+(y-yi).^2);
si = squeeze(m.interp1r(1,ri(:),length(m.time.t)));

% superposition
s = reshape(100*(sp-si),size(rp));

% contour plot
figure
contourf(x,y,s)
colorbar
axis equal
xlim([1 100])
ylim([1 100])
set(gca,'fontsize',12)
xlabel('x (m)')
ylabel('y (m)')
```

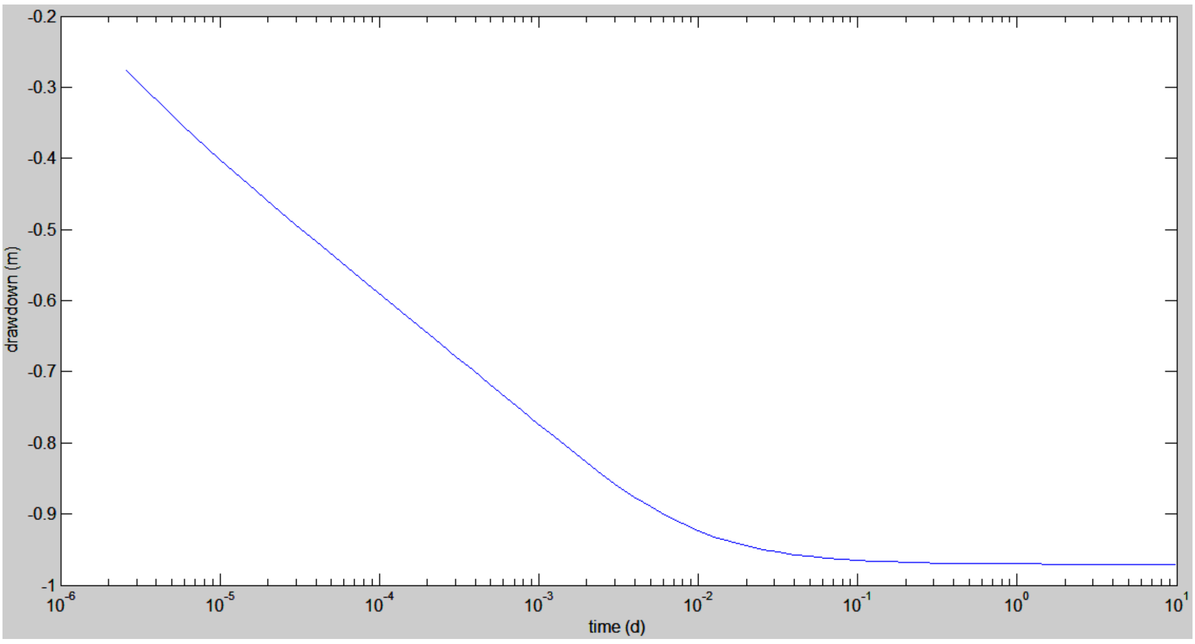


Figure 56. Time-drawdown graph for the extraction well used in the forced gradient test. Drawdown is simulated using MAxSym and applying superposition.

Method `m.interp1r` is used again to perform the linear interpolation. The method only accepts vectors as input arguments, hence, the colon operator `(:)` is used to resize input matrices `rp` and `ri` into column vectors. Afterwards, the interpolated drawdown arrays are reshaped using function `reshape`.

The resulting Figure 57 shows the resulting contour plot. From Figure 56 it is clear that the system has reached steady state conditions by this time. In between both wells, there is a zero drawdown boundary at all times. This clearly illustrates the idea behind the method of images: a constant-head boundary near a well with discharge *Q* is taken into account by superimposing the drawdown of an image well with discharge –*Q*. Similarly, an impervious boundary is obtained by considering an image well with the same discharge *Q* as the real well.

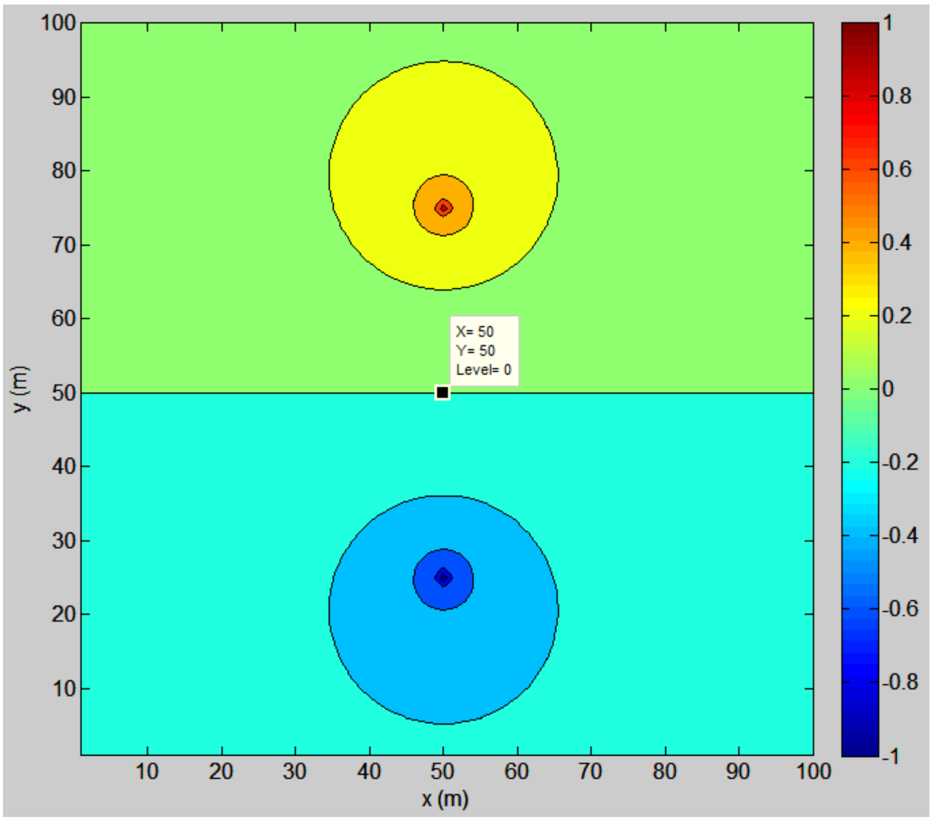


Figure 57. Contour plot of drawdown for the forced gradient test conducted in a confined aquifer. Drawdown is calculated using MAxSym and applying superposition at t = 10 days after the start of the test. The data cursor indicates the zero drawdown boundary.

Now we have calculated drawdown for each node *(i,j)* in the raster, it is straightforward to derive the flow velocity field. The x-component q_x of Darcian velocity between nodes *(i, j)* and *(i,j+1)* is defined as:

$$q_x(i,j+0.5) = -K^h \frac{s(i,j+1) - s(i,j)}{\Delta x}$$

where K^h is the horizontal conductivity of the aquifer and *Δx* is the constant distance between the nodes in the *x*-direction. Column indices are numbered from left to right. Similarly, the *y*-component q_y of Darcian velocity between nodes *(i,j)* and *(i+1,j)* is:

$$q_y(i+0.5,j) = -K^h \frac{s(i+1,j) - s(i,j)}{\Delta y}$$

where *Δy* is the constant distance between the nodes along the *y*-axis. Row indices are numbered from bottom to top. To obtain the velocity components for node *(i,j)*, linear interpolation is applied:

$$q_x(i,j) = [q_x(i,j-0.5) + q_x(i,j+0.5)]/2$$

$$q_y(i,j) = [q_y(i-0.5,j) + q_y(i+0.5,j)]/2$$

The effective velocity is equal to the Darcian velocity divided by the effective porosity of the aquifer. If effective porosity *n* is 0.3, the effective velocities are calculated as follows:

```
% effective velocities
n = 0.3;
[x,y] = deal(x(2:end-1,2:end-1),y(2:end-1,2:end-1));
qx = -m.par.kr/n*diff(s,1,2);
qy = -m.par.kr/n*diff(s,1,1);
qx = (qx(2:end-1,1:end-1) + qx(2:end-1,2:end))/2;
qy = (qy(1:end-1,2:end-1) + qy(2:end,2:end-1))/2;
ip = find(x==xp & y==yp);
ii = find(x==xi & y==yi);
di = [0 1 -1 98 -98];
qx([ip+di ii+di]) = 0;
qy([ip+di ii+di]) = 0;

% quiver plot
figure
for i = 1:2
    subplot(1,2,i)
    quiver(x,y,qx/2,qy/2,0)
    axis equal
    xlim([40 60])
    ylim([15 35]+(i-1)*50)
    set(gca,'fontsize',12)
    xlabel('x (m)')
    ylabel('y (m)')
end
```

The resulting velocity plot is shown in Figure 58. The velocities are large in the close vicinity of the wells and small elsewhere. Therefore, two subplots are created zooming in to both wells, and the velocities for the nodes close to the wells are ignored by setting their value zero. Additionally, the velocity values are rescaled by dividing them by two.

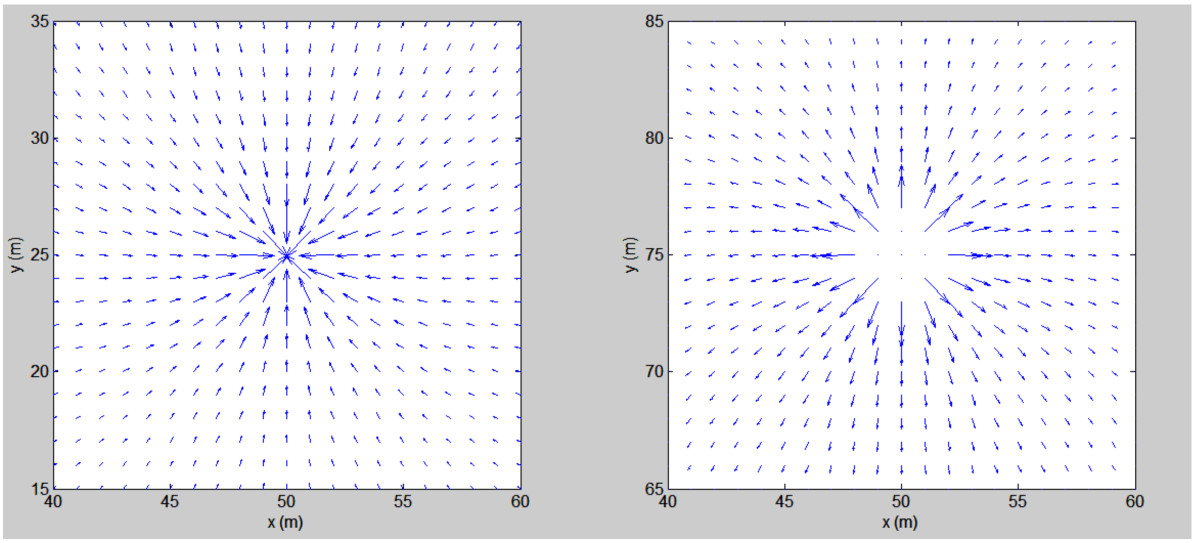


Figure 58. Quiver plots representing effective velocities around extraction well (left) and injection well (right) for the forced gradient test performed in a confined aquifer.

Forced gradient test in laterally anisotropic confined aquifer

Consider the same test as in previous example, but now, the horizontal conductivity of the aquifer is laterally anisotropic and characterized by K_{max} = 20 m/d in the direction characterized by angle α = 30°, and K_{min} = 5 m/d. Since the effective conductivity K_e is $(K_{max}K_{min})^{1/2}$ = 10 m/d, the same axi-symmetric model can be used as in previous example. The code implementing this example is found in script "..\Examples\Superposition\forcedGradientTestLatAnisotrop.m":

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,1,61)));

% set grid
m.setgrid(logspace(-1,7,81),10,true);

% set parameters
m.par.kr = 10;
m.par.ss = 1e-4;

% set stresses
m.stress.q = [1, zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,1);

% run model
m.run;
```

As in previous example, a contour plot is created for a raster of 100 m x 100 m with origin (0,0) at the lower left corner. Again, the forced gradient test lasts 10 days and is performed using an extraction well located at (x_p,y_p) = (50,25) and an injection well located at (x_i,y_i) = (50,75). The extraction and injection rates are Q = 100 m³/d and $-Q$ = -100 m³/d respectively.

The superposition for this case is expressed as:

$$s(x,y,t) = Q\left[s_u(r_p,t) - s_u(r_i,t)\right]$$

where $s(x,y,t)$ is the drawdown at point (x,y) and at time t after the start of the test, and $s_u(r_\alpha,t)$ is the drawdown according to the corresponding axi-symmetric model with effective conductivity and unit discharge. Recall that distance r_a is an apparent distance, respectively defined as:

$$\begin{cases} r_p = \sqrt{(x-x_p)^2 + (y-y_p)^2}\,\sqrt{\cos^2(\alpha-\theta_p)/a + \sin^2(\alpha-\theta_p).a} \\ \theta_p = atan2(y-y_p, x-x_p) \end{cases}$$

$$\begin{cases} r_i = \sqrt{(x-x_i)^2 + (y-y_i)^2}\,\sqrt{\cos^2(\alpha-\theta_i)/a + \sin^2(\alpha-\theta_i).a} \\ \theta_i = atan2(y-y_i, x-x_i) \end{cases}$$

where *a* is the anisotropy factor defined as *(K_{max}/K_{min})$^{1/2}$*, and *atan2* is the arctangent function that takes into account the quadrant. Calculation of drawdown corresponding to the apparent distances is done using linear interpolation:

```
% 2D raster
[x,y] = meshgrid(1:100);

% wells
[xp,yp] = deal(50,25);
[xi,yi] = deal(50,75);

% interpolation
alfa = pi/6;
a = sqrt(20/5);

r = sqrt((x-xp).^2+(y-yp).^2);
theta = atan2(y-yp,x-xp);
r = r .* sqrt(cos(alfa-theta).^2/a+sin(alfa-theta).^2*a);
sp = m.interp1r(1,r(:),length(m.time.t));

r = sqrt((x-xi).^2+(y-yi).^2);
theta = atan2(y-yi,x-xi);
r = r .* sqrt(cos(alfa-theta).^2/a+sin(alfa-theta).^2*a);
si = m.interp1r(1,r(:),length(m.time.t));

% superposition
s = reshape(100*(sp-si),size(r));

% contour plot
figure
contourf(x,y,s)
colorbar
axis equal
xlim([1 100])
ylim([1 100])
caxis([-1 1])
set(gca,'fontsize',12)
xlabel('x (m)')
ylabel('y (m)')
```

Method `m.interp1r` is used to perform the linear interpolation. Recall that the method only accepts vectors as input arguments, and the colon operator `(:)` is used to resize input matrix `r` into a column vector. Afterwards, the interpolated drawdown arrays are reshaped using function `reshape`.

The resulting contour plot is shown in Figure 59. It is noticed that the zero drawdown boundary in between both wells is parallel to the direction of the maximum conductivity. This explains why the method of images is applicable only if the principal direction is parallel to the lateral bounds in case of a laterally anisotropic system.

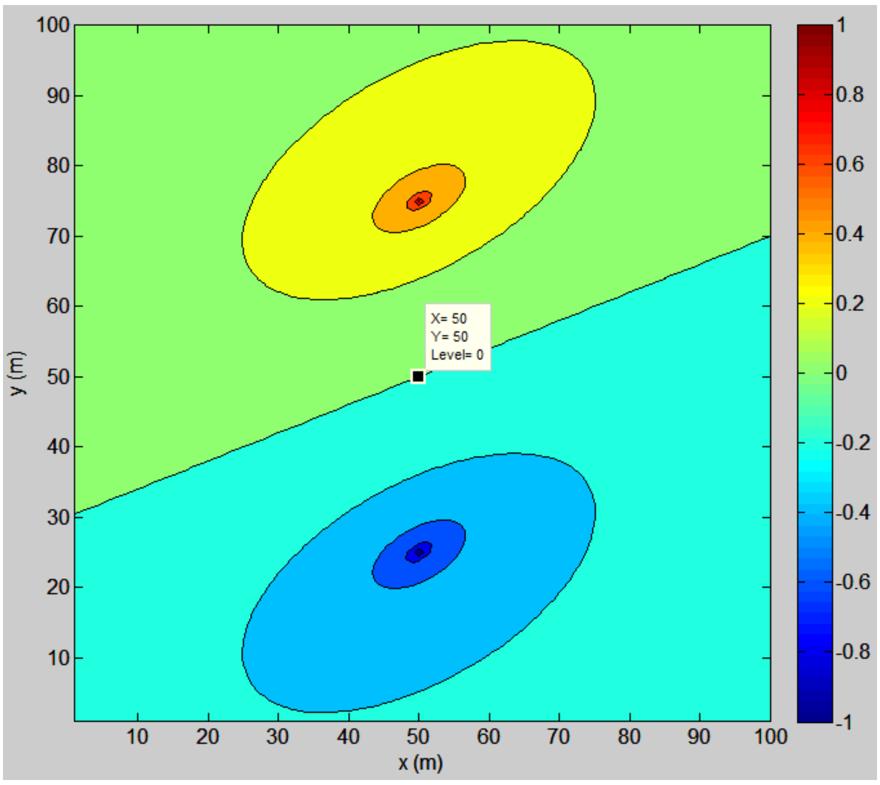


Figure 59. Contour plot of drawdown for the forced gradient test conducted in a confined, laterally anisotropic aquifer. Drawdown is calculated using MAxSym and applying superposition at t = 10 days after the start of the test. The data cursor indicates the zero drawdown boundary.

The flow velocity field is calculated the same way as in previous example, except for the horizontal conductivity, which is dependent on the angular direction in case of lateral anisotropy. Recall that the general expression for *K* as a function of *ϑ* is:

$$K_\theta = \frac{1}{\cos^2(\alpha-\theta)/K_{max} + \sin^2(\alpha-\theta)/K_{min}}$$

from which it follows that:

$$K_x = \frac{1}{\cos^2(\alpha)/K_{max} + \sin^2(\alpha)/K_{min}}$$

$$K_y = \frac{1}{\cos^2(\alpha-\pi/2)/K_{max} + \sin^2(\alpha-\pi/2)/K_{min}}$$

where K_x and K_y are the horizontal conductivity in the *x* and *y* direction respectively. The *x*-component q_x of Darcian velocity between nodes *(i, j)* and *(i,j+1)* is defined as:

$$q_x(i,j+0.5) = -K_x\frac{s(i,j+1)-s(i,j)}{\Delta x}$$

where *Δx* is the constant distance between the nodes in the *x*-direction. The *y*-component q_y of Darcian velocity between nodes *(i,j)* and *(i+1,j)* is:

$$q_y(i+0.5,j) = -K_y \frac{s(i+1,j) - s(i,j)}{\Delta y}$$

where *Δy* is the constant distance between the nodes in the *y*-direction. To obtain the velocity components for node *(i,j)*, linear interpolation is applied, as was explained in previous example. Recall that the effective velocity is obtained by dividing the Darcian velocity by effective porosity. If effective porosity *n* is 0.3, the effective velocities for the defined grid are calculated as follows:

```
% effective velocities
n = 0.3;
[x,y] = deal(x(2:end-1,2:end-1),y(2:end-1,2:end-1));

kx = 1/(cos(alfa)^2/20+sin(alfa)^2/5);
qx = -kx/n*diff(s,1,2);
qx = (qx(2:end-1,1:end-1) + qx(2:end-1,2:end))/2;

ky = 1/(cos(alfa-pi/2)^2/20+sin(alfa-pi/2)^2/5);
qy = -ky/n*diff(s,1,1);
qy = (qy(1:end-1,2:end-1) + qy(2:end,2:end-1))/2;

ip = find(x==xp & y==yp);
ii = find(x==xi & y==yi);
di = [0 1 -1 98 -98];
qx([ip+di ii+di]) = 0;
qy([ip+di ii+di]) = 0;

% quiver plot around wells
figure
for i = 1:2
    subplot(1,2,i)
    quiver(x,y,qx/2,qy/2,0)
    axis equal
    xlim([40 60])
    ylim([15 35]+(i-1)*50)
    set(gca,'fontsize',12)
    xlabel('x (m)')
    ylabel('y (m)')
end
```

The resulting velocity plot is shown in Figure 60. Two subplots are created zooming in to both wells and the large velocity values for the nodes close to the wells are ignored by setting them zero. The velocity values are rescaled by dividing them by two. Because the flow pattern is more difficult to understand than for the isotropic case (flow is not always perpendicular to drawdown contour lines in case of laterally anisotropy!), a second velocity plot for the entire grid was created, shown in Figure 61. Again, the large values around the wells are ignored:

```
% quiver plot entire grid
[ip,jp] = ind2sub(size(x),ip);
[ii,ji] = ind2sub(size(x),ii);
di = -5:5;
qx(ip+di,jp+di) = 0;
qx(ii+di,ji+di) = 0;
qy(ip+di,jp+di) = 0;
qy(ii+di,ji+di) = 0;
```

```
figure
quiver(x(1:2:end,1:2:end),y(1:2:end,1:2:end),...
       qx(1:2:end,1:2:end),qy(1:2:end,1:2:end),3)
axis equal
xlim([0 100])
ylim([0 100])
set(gca,'fontsize',12)
xlabel('x (m)')
ylabel('y (m)')
```

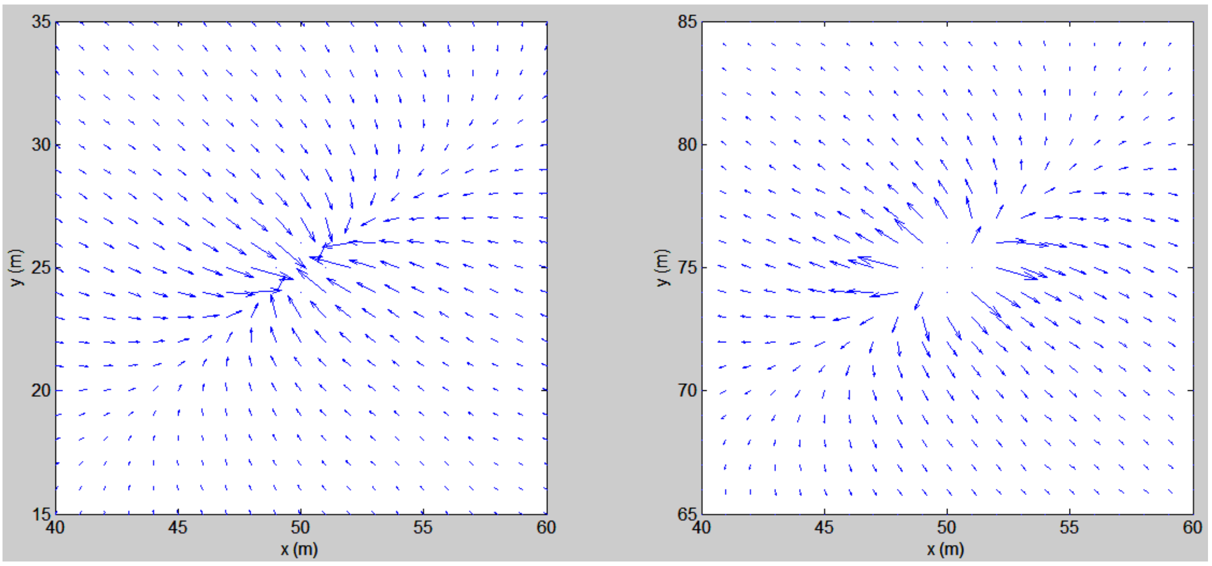


Figure 60. Quiver plots representing effective velocities around extraction well (left) and injection well (right) for the forced gradient test performed in a laterally anisotropic, confined aquifer.

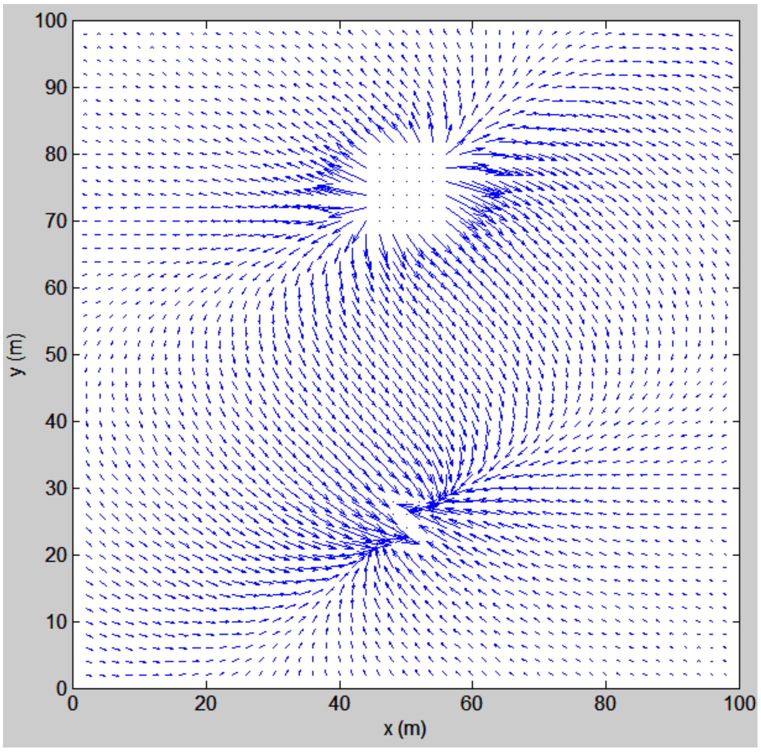


Figure 61. Quiver plot representing effective velocities for the forced gradient test performed in a laterally anisotropic, confined aquifer.

Simultaneous slug testing in confined aquifer

To illustrate the superposition of head specified wells, two slug tests are simulated, starting at the same time and conducted in fully penetrating wells located in a confined, homogeneous aquifer. If both wells have a diameter of r_w and if the distance between the well centers is *L*, then superposition is expressed as:

$$s_1(r_w,t) = H_1 s_u(r_w,t) + H_2 s_u(L,t)$$

$$s_2(r_w,t) = H_2 s_u(r_w,t) + H_1 s_u(L,t)$$

where $s_1(r_w,t)$ and $s_2(r_w,t)$ denote drawdown at time *t* at the face of well 1 and 2 respectively, H_1 and H_2 are the initial head changes within well 1 and 2 respectively, and $s_u(L,t)$ is the drawdown at distance *L* and at time *t* according to the corresponding axi-symmetric model with unit initial head change. Recall that the head change within the well is assumed equal to the drawdown at the face of the well screen.

Suppose the wells have a diameter of 0.03 m and the distance between the well centers is 5 m. The aquifer transmissivity is 1 m²/d and the aquifer storativity is 1e-5. The tests start at the same time and last 1 day. The initial head changes within the wells are 1.5 m and 2 m. The code implementing this example is found in script "..\Examples\Superposition\twoSlugTests.m":

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-6,1,251)));

% set grid
rb = logspace(log10(0.03),3,401);
rb = [rb(1)^2/rb(2),rb];
m.setgrid(rb,1,true);

% set parameters
m.par.kr = 1;
m.par.ss = [rb(2)^2*pi/m.grid.vol(1),1e-5*ones(1,m.grid.nr-1)];

% set stresses
m.stress.s0 = [1,zeros(1,m.grid.nr-1)];

% set solver
m.setsolver(1e-5,1);

% run model
m.run;

% drawdown within well
sw = squeeze(m.s(1,1,:));

% drawdown at 5 m
sr = squeeze(m.interp1r(1,5,[]));

% initial head changes
H0 = [1.5; 2];
```

```
% superposition
s = [sw,sr] * [H0,flipud(H0)];

% time-drawdown plot
figure
semilogx(m.time.t,s)
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
xlim([1e-6 1])
legend('well 1','well 2')
```

The resulting time-drawdown plot is shown in Figure 62.

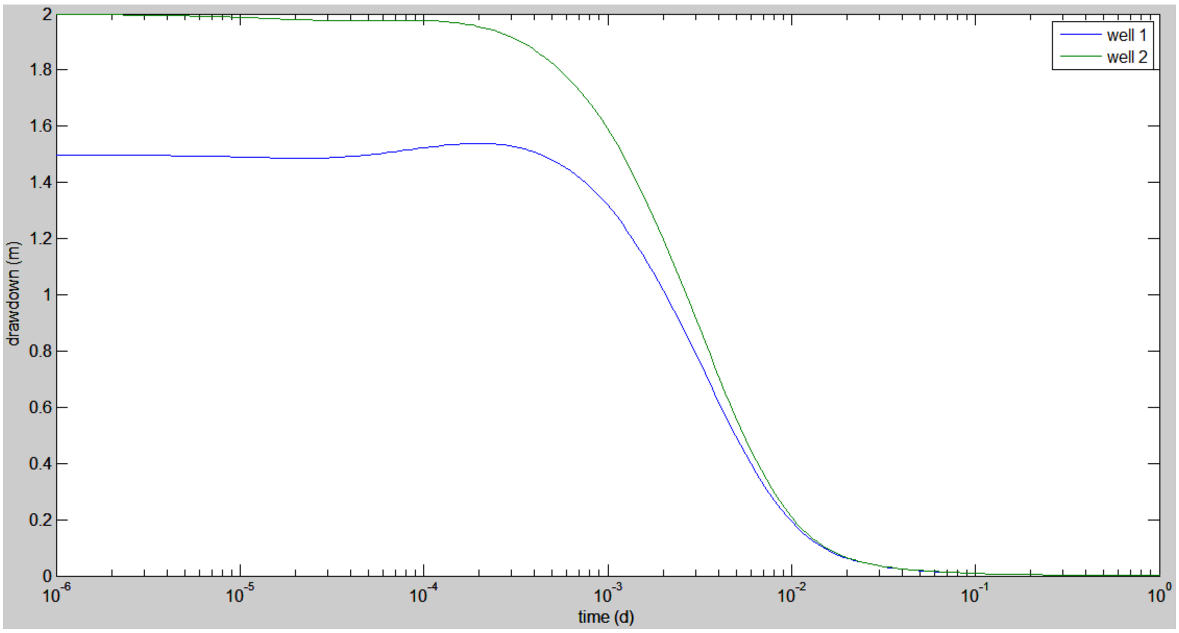


Figure 62. Time-drawdown graph for the two slug tests performed in a confined aquifer. Drawdown is calculated using MAxSym and applying superposition.

Permanent wells in leaky aquifer

Consider an aquifer with thickness of 6 m and horizontal conductivity of 10 m/d. The aquifer is bounded below by an aquiclude and above by an aquitard with constant head at the top. The thickness of the aquitard is 5 m, the horizontal conductivity is 0.01 m/d. Both aquifer and aquitard have vertical anisotropy of 10, meaning that the vertical conductivity is ten times smaller than the horizontal conductivity. Two permanent wells are located in the aquifer, both having a screen of 1 m. The top of the first screen is 3 m above the bottom of the aquifer, the top of the second screen is 1 m above the bottom of the aquifer. The horizontal distance between the wells is 5 m. The well discharges are constant and equal to 100 m^3/d and 250 m^3/d respectively.

To simulate drawdown due to these extractions, we first need to run two axi-symmetric models, because the well screens do not have the same vertical position. To have an accurate simulation of vertical flow, a fine vertical discretization is applied by subdividing the aquifer and aquitard into sub-layers of 0.1 m thickness. A constant-head layer is defined at the top of the aquitard and the former adopts the hydraulic properties of the latter. The wells are permanent and pumping rates are constant, thus steady state conditions are assumed. The following statements run the models and are found in script "..\Examples\Superposition\permanentWells.m":

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(0);

% set grid
m.setgrid(logspace(-1,7,81),0.1*ones(111,1),true);

% set parameters
m.par.constant = [true; false(110,1)];
m.par.kr = [0.01*ones(51,1); 10*ones(60,1)];
m.par.kz = m.par.kr/10;

% set solver
m.setsolver(1e-5,100,5);

% run models
su = [];
for n = [70 90]

    % discharge
    m.stress.q = zeros(m.grid.nz,m.grid.nr);
    m.stress.q(n+(1:10),1)  = 0.1;

    % run
    m.run;
    su = cat(3,su,m.s);

end
```

The screened parts of the aquifer are subdivided into 10 sub-layers and therefore, the unit discharge for each well is divided by 10. The non-screened parts of the wells are not taken into account by defining inactive rings, and neither is a small vertical resistance defined to obtain the same drawdown in each of the rings representing the wells. Fortunately, these effects are negligible because they are difficult or even impossible to account for when superposition is applied.

The unit discharge solutions for both wells are used to calculate drawdown for a vertical cross section intersecting the wells. A grid is defined with horizontal extent of 100 m and vertical extent of 11 m, which is the total thickness of the two layer system. The former is discretized into columns with a constant width of 0.1 m, the latter adopts the sub-layers of the axi-symmetric model. The origin is located at the lower center of the grid and the wells are located at x = -2.5 m and x = 2.5 m:

```
% cross section
x = -50:0.1:50;
z = [m.grid.z(:);m.grid.zb(end)];

% wells
[x1,q1] = deal(-2.5,100);
[x2,q2] = deal(2.5,250);
r1 = abs(x-x1);
```

```
r2 = abs(x-x2);

% interpolation
for n = 2:m.grid.nz
    s1(n,:) = interp1(log10(m.grid.r),su(n,:,1),log10(r1));
    s1(n,r1<m.grid.r(1)) = su(n,1,1);
    s2(n,:) = interp1(log10(m.grid.r),su(n,:,2),log10(r2));
    s2(n,r2<m.grid.r(1)) = su(n,1,2);
end

% superposition
s = q1*s1 + q2*s2;
s = [s; s(end,:)];

% contour plot
figure
[cs,h] = contourf(x,z,log10(-s),-1.4:0.1:1.3);
clabel(cs,h,-0.2:0.2:0.6)
colorbar
set(gca,'fontsize',12)
xlabel('x (m)')
ylabel('z (m)')
```

Because the vertical position of the nodes does not differ from the vertical position of the nodal circles in the axi-symmetric grid, linear interpolation is performed for the horizontal distances only. Only one `Model` object is used, and therefore MATLAB function `interp1` is used to perform the linear interpolation instead of method `m.interp1r`. To account for the effect of the lower impervious boundary, nodes are defined at the bottom of the aquifer. The drawdown for these nodes must be the same as the drawdown for the adjacent nodes at the center of the lower sub-layer to obtain a strictly horizontal flow near the boundary.

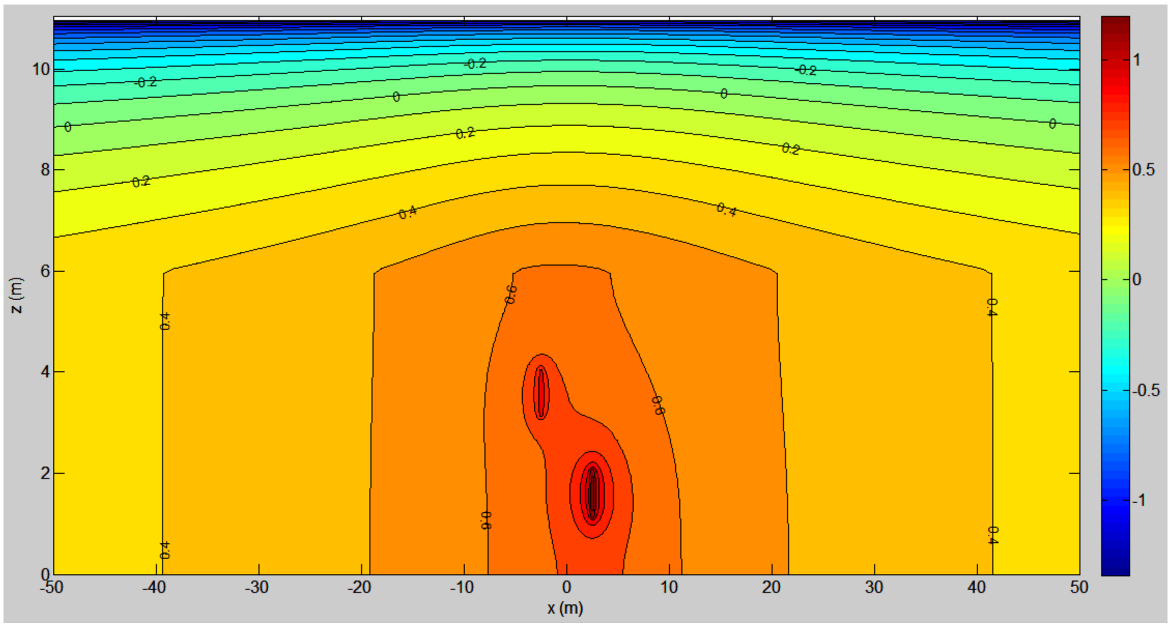


Figure 63. Cross section plot for two permanent wells in a leaky aquifer. Contour lines indicate the logarithm of drawdown calculated using MAxSym and applying superposition.

The resulting cross section plot is shown in Figure 63. Because of the large differences between drawdown near the wells and drawdown elsewhere in the aquifer or aquitard, the logarithm of drawdown is plotted. This explains the white border at the top of the aquitard, which corresponds to the zero drawdown boundary.

Finally, specific discharges are derived from drawdown calculated for the considered grid. Recall that the horizontal component q_x of the specific discharge between nodes *(i,j)* and *(i,j+1)* is calculated as:

$$q_x(i,j+0.5) = -K^h \frac{s(i,j+1) - s(i,j)}{\Delta x}$$

where K^h is the horizontal conductivity of the aquifer and *Δx* is the constant distance between the nodes in the *x*-direction. The column indices are numbered from left to right. The vertical component q_z of the specific discharge between nodes *(i,j)* and *(i+1,j)* is:

$$q_z(i+0.5,j) = \frac{s(i+1,j) - s(i,j)}{C^v(i+0.5)}$$

where $C^v(i+0.5)$ is the vertical resistance between layer *i* and *i+1*:

$$C^v(i+0.5) = \frac{D(i)}{2K^v(i)} + \frac{D(i+1)}{2K^v(i+1)}$$

with *D(i)* the thickness and $K^v(i)$ the vertical conductivity of layer *i*. Note that the layer indices are numbered from top to bottom. To obtain the specific discharge components for node *(i,j)*, linear interpolation is applied:

$$q_x(i,j) = [q_x(i,j-0.5) + q_x(i,j+0.5)]/2$$

$$q_z(i,j) = [q_z(i-0.5,j) + q_z(i+0.5,j)]/2$$

The following statements calculate the specific discharge components for the defined grid:

```
% specific discharges
[x,z] = deal(x(2:end-1),z(2:end-1));
qx = -repmat(m.par.kr([1:end,end]),1,size(s,2)-1) .* diff(s,1,2)/0.1;
cv = m.grid.D ./ m.par.kz / 2;
cv = [cv(1:end-1) + cv(2:end); cv(end)];
qz = diff(s,1,1) ./ repmat(cv,1,size(s,2));
qx = (qx(2:end-1,1:end-1) + qx(2:end-1,2:end))/2;
qz = (qz(1:end-1,2:end-1) + qz(2:end,2:end-1))/2;
```

Two plots are created showing respectively the horizontal and vertical Darcian velocities as a function of vertical distance *z* in the aquifer between the two extraction wells (Figure 64):

```
% specific discharge plot
figure

subplot(1,2,1)
iz = z<=6;
ix = -2:1:2;
plot(qx(iz,ismember(x,ix)),z(iz))
legend(strcat(num2str(ix'),'m'))
```

```
set(gca,'fontsize',12)
xlabel('q_x (m/d)')
ylabel('z (m)')

subplot(1,2,2)
plot(qz(iz,ismember(x,ix)),z(iz))
legend(strcat(num2str(ix'),'m'))
set(gca,'fontsize',12)
xlabel('q_z (m/d)')
ylabel('z (m)')
```

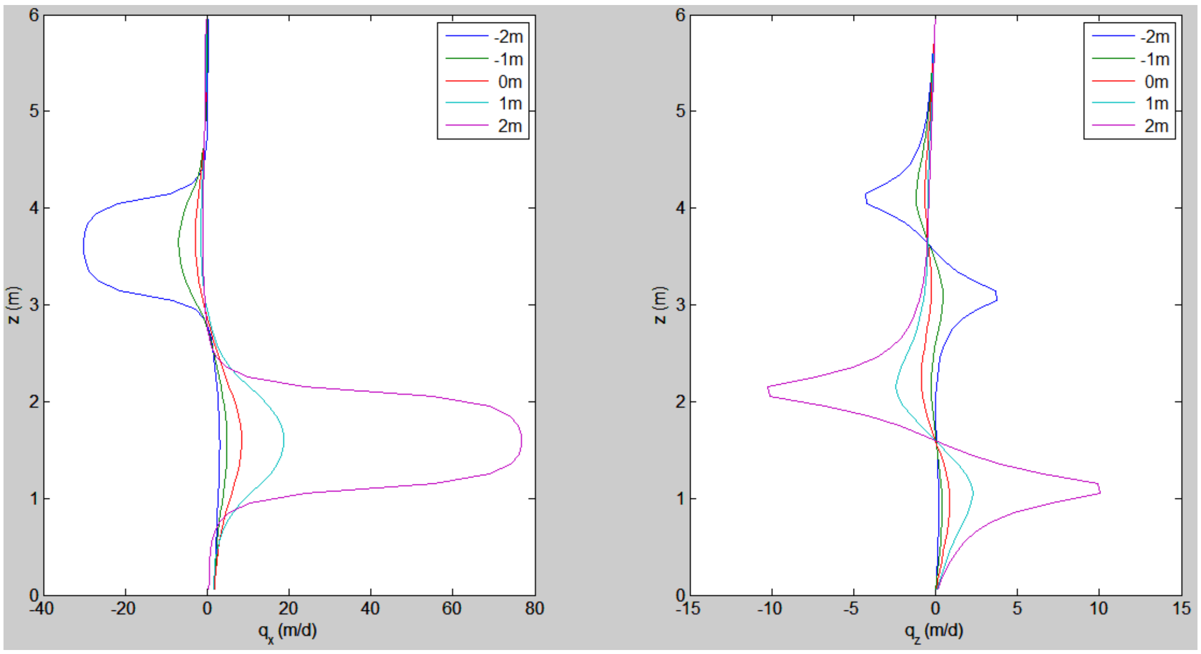


Figure 64. Plot of horizontal (left) and vertical (right) Darcian velocity as a function of vertical distance z in the leaky aquifer. The curves correspond to different horizontal positions between the two extraction wells located at x = -2.5 m and x = 2.5 m. The legend indicates the x-coordinates of these positions.

Multi-level extraction near constant-head and impervious boundary

Drawdown due to multi-level extraction near a constant-head and no-flow boundary is simulated to illustrate the application of the method of images. Consider a leaky two-aquifer system bounded below by an impervious layer and above by an aquitard with constant-head boundary at the top and vertical resistance of 1000 d. The two aquifers are separated by an aquitard with vertical resistance of 500 d. The thickness of both aquifers is 10 m, the transmissivity of the upper aquifer is 100 m^2/d, the transmissivity of the lower aquifer is 50 m^2/d. A permanent well is located in the aquifer system which extracts water from both aquifers, the discharge in the upper aquifer is 750 m^3/d, the discharge in the lower aquifer is 500 m^3/d. The aquifer system is bounded laterally by a no-flow boundary in the north and a constant-head boundary in the east. The distance to the well is 15 m for both lateral bounds.

First, a map is plotted to explain the method of images for this particular case. Consider a raster of 100 m x 100 m with origin at the lower left corner. The impervious boundary is situated at *y* = 100, the constant head boundary at *x* = 100. Consequently, the well is located at *(x,y)* = (85,85). The following statements create the map presented in Figure 65, and are found in script "..\Examples\Superposition\lateralBounds.m":

```
% coordinates and sign for well and image wells
[xp,yp,q] = deal([85 85 115 115],[85 115 85 115],[1 1 -1 -1]);

% map indicating well, image wells and lateral bounds
figure
plot([70 100; 100 100]',[100 100; 70 100]');
axis equal
xlim([70 130])
ylim([70 130])
set(gca,'fontsize',12)
xlabel('x (m)')
ylabel('y (m)')

hold on
for i = 1:4
    if i==1
        col = 'b';
    else
        col = 'r';
    end
    if q(i)>0
        sgn = '';
    else
        sgn = '-';
    end
    plot(xp(i),yp(i),strcat(col,'o'))
    text(xp(i)+1,yp(i),strcat(sgn,'Q'))
end

plot([100 130; 100 100]',[100 100; 100 130]','--');
legend('no flow','constant head','well','image well')
```

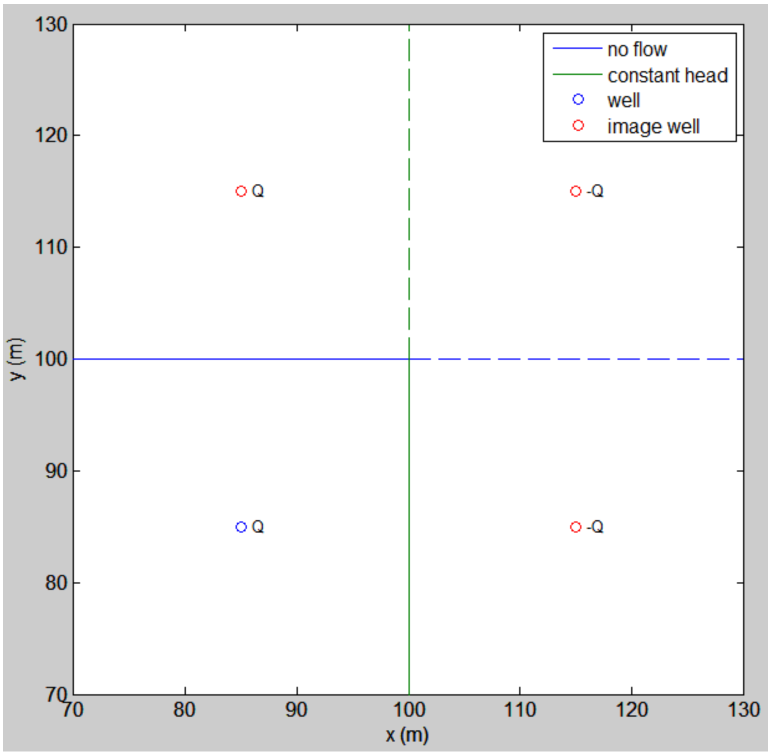


Figure 65. Map showing positions of well, image wells and lateral bounds for the multi-level extraction model.

The no-flow boundary is taken into account using an image multi-level extraction well with the same pumping rates as the real well, the constant-head boundary is accounted for using an image multi-level injection well with the opposite injection rates. Note that it is required to consider an additional image injection well to reflect the two image wells mentioned above.

The simulation of multi-level extraction near lateral bounds starts with running an axi-symmetric model simulating steady state drawdown in the absence of lateral bounds. Because the well screens have the same horizontal position and the extraction and injection rates of the image wells have the same magnitude, it is not required to run an axi-symmetric model with unit discharge for each aquifer extraction. Instead, we may simulate the simultaneous extraction with one model considering the given extraction rates:

```
% create Model object
m = MAxSym.Model;

% set time steps
m.settime(0);

% set grid
m.setgrid(logspace(-1,7,81),[1;10;10],true);

% set parameters
m.par.constant = [true;false(2,1)];
m.par.kr = [0;10;5];
m.par.cz = [1000;500];

% set stresses
m.stress.q = zeros(m.grid.nz,m.grid.nr);
m.stress.q(2:3,1) = [750;500];

% set solver
m.setsolver(1e-5,20,5);

% run model
m.run;
```

The effect of the lateral bounds is obtained by superposition of the solutions for the real well and the three image wells:

$$s_i(x,y) = \sum_{p=1}^{4} \operatorname{sgn}(Q_p)\, s_i\left(\sqrt{(x-x_p)^2+(y-y_p)^2}\right)$$

where $s_i(x,y)$ is the drawdown at point *(x,y)* in aquifer *i*, $s_i(r)$ is the drawdown in the axi-symmetric model at distance *r* in aquifer *i*, (x_p,y_p) is the coordinate of the *p*-th well with discharge Q_p, and *sgn* is the sign function which returns 1 if Q_p is positive and -1 if Q_p is negative. Applying superposition, drawdown is calculated in both aquifers for a 100 m x 100 m raster:

```
% 2D raster
[x,y] = meshgrid(0:100);

% interpolation and superposition
s = zeros([size(x),m.grid.nz-1]);
```

```
    for j = 1:length(xp)
        for i = 1:m.grid.nz-1
            r = sqrt((x-xp(j)).^2+(y-yp(j)).^2);
            tmp = reshape(m.interp1r(i+1,r(:)),size(r));
            s(:,:,i) = s(:,:,i) + q(j) * tmp;
        end
    end

    % contour plot
    figure
    for i = 1:2
        subplot(1,2,i)
        contourf(x,y,s(:,:,i))
        colorbar
        axis equal
        xlim([0 100])
        ylim([0 100])
        set(gca,'fontsize',12)
        xlabel('x (m)')
        ylabel('y (m)')
        title(['aquifer ',int2str(i)])
    end
```

The resulting contour plots are presented in Figure 66.

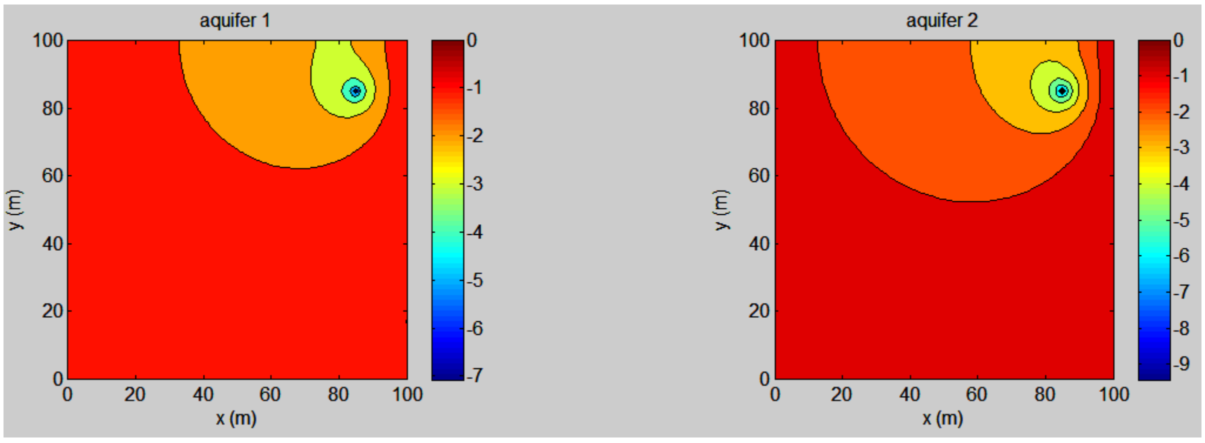


Figure 66. Contour plots of drawdown for the multi-level extraction in a leaky two-aquifer system bounded laterally by a no-flow boundary in the north and a constant-head boundary in the east. Drawdown is calculated using MAxSym and applying the method of images.

Finally, the flow velocity field is calculated in both aquifers for the considered grid, where it is assumed the effective porosity *n* is 0.3 for both aquifers:

```
    % effective velocities
    n = 0.3;
    [x,y] = deal(x(2:end-1,2:end-1),y(2:end-1,2:end-1));
    ip = find(x==xp(1) & y==yp(1));
    di = [0 1 -1 98:100 -100:-98];
    for i = 1:size(s,3)
        tmp = -m.par.kr(i+1)/n*diff(s(:,:,i),1,2);
        tmp = (tmp(2:end-1,1:end-1) + tmp(2:end-1,2:end))/2;
        tmp(ip+di) = 0;
        qx(:,:,i) = tmp;
        tmp = -m.par.kr(i+1)/n*diff(s(:,:,i),1,1);
        tmp = (tmp(1:end-1,2:end-1) + tmp(2:end,2:end-1))/2;
        tmp(ip+di) = 0;
        qy(:,:,i) = tmp;
    end

    % quiver plot
    figure
    for i = 1:2
        subplot(1,2,i)
        quiver(x,y,qx(:,:,i)/10,qy(:,:,i)/10,0)
        axis equal
        xlim([80 100])
        ylim([80 100])
        set(gca,'fontsize',12)
        xlabel('x (m)')
        ylabel('y (m)')
        title(['aquifer ',int2str(i)])
    end
```

The resulting plot is shown in Figure 67. Note that the flow components near the well are ignored.

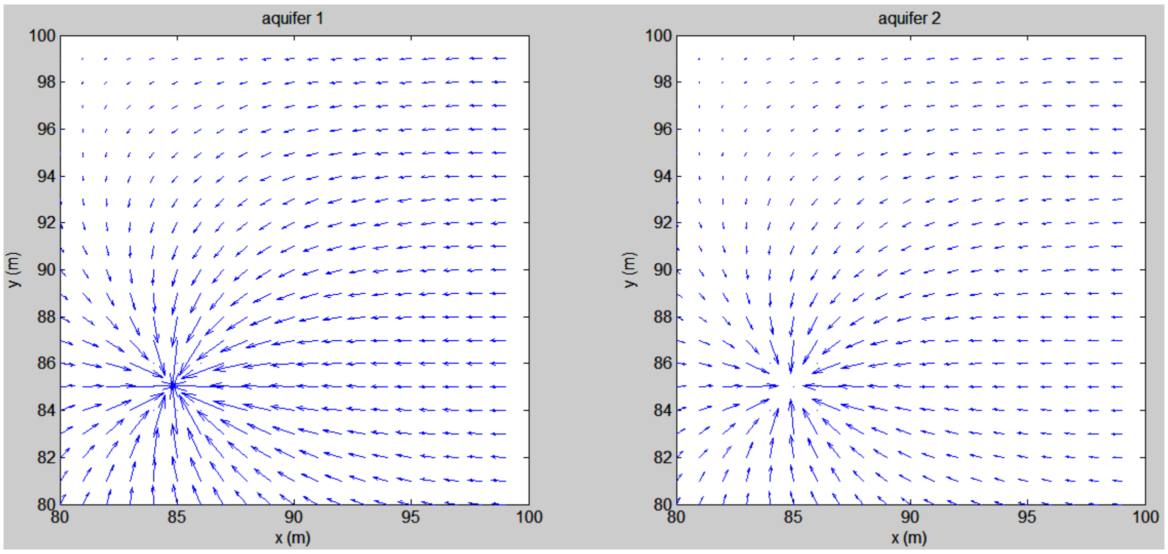


Figure 67. Quiver plots of effective velocity for the multi-level extraction in a leaky two-aquifer system bounded laterally by a no-flow boundary in the north and a constant-head boundary in the east.

Transient well field in multi-aquifer system with recharge

In this example, a transient well field in an unconfined two-aquifer system with recharge is simulated using MAxSym and applying superposition. The system is bounded below by an aquiclude and the aquifers are separated by an aquitard.

The characteristics of the aquifers and aquitard are:

- Upper aquifer: D = 20 m, K^h = 20 m/d, K^v = 10 m/d, S^s = 1e-4 m^{-1}, S^y = 0.2, N = 5e-4 m/d;
- Aquitard: D = 10 m, K^h = 0.1 m/d, K^v = 0.01 m/d, S^s = 1e-6 m^{-1};
- Lower aquifer: D = 30 m, K^h = 10 m/d, K^v = 1 m/d, S^s = 1e-5 m^{-1};

where D denotes thickness, K^h horizontal conductivity, K^v vertical conductivity, S^s specific storage coefficient, S^y specific yield, and N infiltration rate.

A well field is situated in the lower aquifer, consisting of 20 wells, each of them extracted with a constant rate of 250 m³/d. All wells are located on a straight line and the horizontal distance between the wells is 10 m. The effect of partial penetration is neglected, which means the wells are assumed to fully penetrate the lower aquifer. An extraction period of 1000 days is considered.

First, two MAxSym models are run, one considering the recharge and the other with a unit discharge well in the lower aquifer. The infiltration radius is derived from the total discharge of the well field, i.e. 20 x 250 m³/d. Script "..\Examples\Superposition\transientWellField.m" implements this example:

```
% MAXSYM MODELS

% recharge
N = 5e-4;
R = sqrt(250*20/pi/N);

% create Model object
m = MAxSym.Model;

% set time steps
m.settime(diff(logspace(-5,3,81)));

% set grid
rb = logspace(-5,log10(R),51);
rb = [rb(1:end-1) logspace(log10(R),7,41)];
m.setgrid(rb,[20;10;30],false);

% set parameters
m.par.kr = [20;0.1;10];
m.par.kz = [10;0.01;1];
m.par.ss = [1e-4;1e-6;1e-5];
m.par.sy = 0.2;

% set solver
m.setsolver(1e-7,50,5);

% run model with recharge
m.stress.q = zeros(m.grid.nz,m.grid.nr);
m.stress.q(1,2:50) = -N*m.grid.hs(2:50);
m.run;
sw = permute(m.s,[2 1 3]);

% run model with pumping
m.stress.q = zeros(m.grid.nz,m.grid.nr);
m.stress.q(3,1) = 1;
m.run
sp = permute(m.s,[2 1 3]);
```

Because the model grid and aquifer parameters are the same for both models, one `Model` object is defined and discharge matrix `m.stress.q` is reset before each model run. The calculated drawdown matrices for the recharge and the well model are stored in variables `sr` and `sp` respectively. Note that the dimensions for layers and rings are permuted, which is required for an efficient use of the `interp1` function.

Total drawdown may be calculated now according to:

$$s_i(x,y,t) = s_i^r\left(\sqrt{(x-x_r)^2+(y-y_r)^2},t\right) + \sum_{p=1}^{20} Q_p\, s_i^w\left(\sqrt{(x-x_p)^2+(y-y_p)^2},t\right)$$

where $s_i(x,y,t)$ is the total drawdown in layer *i* at point *(x,y)* and at time *t*, $s^r_i(r,t)$ is the drawdown in layer *i* at distance *r* and at time *t* corresponding to the model with recharge, $s^w_i(r,t)$ is the drawdown in layer *i* at distance *r* and at time *t* corresponding to the model with the unit discharge well, (x_r,y_r) is the coordinate of the center of the circular recharge area, and Q_p is the discharge of the *p*-th well with coordinate (x_p,y_p).

Applying superposition, a contour plot of drawdown in the lower aquifer is created for time steps 40 and 80. Therefore, a 1001 m x 1001 m raster is defined with origin (0,0) at the center. The center of the circular recharge area coincides with the center of the raster, and also the well field is located in the middle of the raster, with *(x,y)* coordatines of the two outer wells equal to (0,-95) and (0,95) respectively:

```
% CONTOUR PLOT

% wells
[xp,yp,q] = deal(zeros(20,1),-95:10:95,250*ones(20,1));

% 2D raster
[x,y] = meshgrid(-500:500);

% interpolation and superposition: pumping wells
ilay = 3;
it = [41 81];
s = zeros([size(x),length(it)]);
for j = 1:length(xp)
    r = sqrt((x(:)-xp(j)).^2+(y(:)-yp(j)).^2);
    tmp = interp1(log10(m.grid.r),squeeze(sp(:,ilay,it)),log10(r));
    b = r < m.grid.r(1);
    if any(b)
        tmp(b,:) = squeeze(repmat(sp(1,ilay,it),[sum(b) 1 1]));
    end
    s = s + q(j) * reshape(tmp,size(s));
end

% interpolation and superposition: recharge
r = sqrt(x(:).^2+y(:).^2);
tmp = interp1(log10(m.grid.r),squeeze(sr(:,ilay,it)),log10(r));
b = r < m.grid.r(1);
tmp(b,:) = squeeze(repmat(sr(1,ilay,it),[sum(b) 1 1]));
s = s + reshape(tmp,size(s));

% plot
figure
for i = 1:length(it)
    subplot(1,length(it),i)
    contourf(x,y,s(:,:,i))
    colorbar
    caxis([-10 0])
    axis equal
```

```
        xlim([-500 500])
        ylim([-500 500])
        set(gca,'fontsize',12)
        xlabel('x (m)')
        ylabel('y (m)')
        title([num2str(m.time.t(it(i)),'%.0e'),' days of pumping'])
    end
```

The resulting contour plots are shown in Figure 68. Linear interpolation is performed using MATLAB function `interp1` instead of method `m.interp1r`, because only one `Model` object is created and drawdown `m.s` was overwritten.

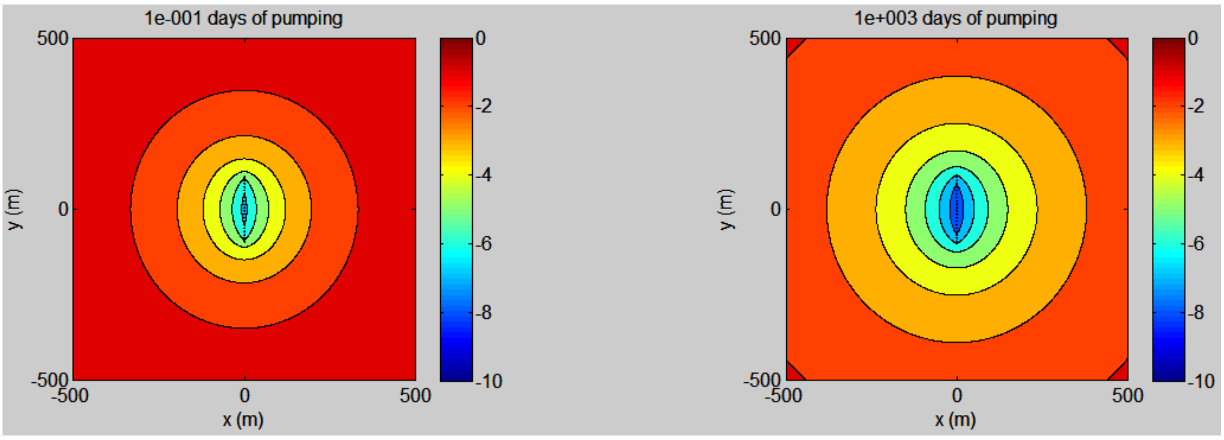


Figure 68. Contour plot of drawdown in the lower aquifer due to an extracted well field located in an unconfined two-layer aquifer system with recharge. Drawdown is calculated using MAxSym and applying superposition.

Finally, total drawdown as a function of time is calculated for point (0,0):

```
    % TIME-DRAWDOWN PLOT

    % point
    [x,y] = deal(0);

    % interpolation and superposition: pumping wells
    s = zeros(m.grid.nz,length(m.time.t));
    for j = 1:length(xp)
        r = sqrt((x-xp(j)).^2+(y-yp(j)).^2);
        if r > m.grid.r(1)
            tmp = interp1(log10(m.grid.r),sw,log10(r));
        else
            tmp = sw(1,:,:);
        end
        s = s + q(j) * squeeze(tmp);
    end

    % interpolation and superposition: recharge
    r = sqrt(x.^2+y.^2);
    if r > m.grid.r(1)
        tmp = interp1(log10(m.grid.r),sr,log10(r));
    else
        tmp = sr(1,:,:);
    end
    s = s + squeeze(tmp);
```

```
% plot
figure
semilogx(m.time.t,s')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
legend('upper aquifer','aquitard','lower aquifer')
```

The resulting time-drawdown plot is shown in Figure 69. Note that the drawdown in the upper aquifer is small compared to the aquifer thickness, which justifies the application of superposition for this particular case.

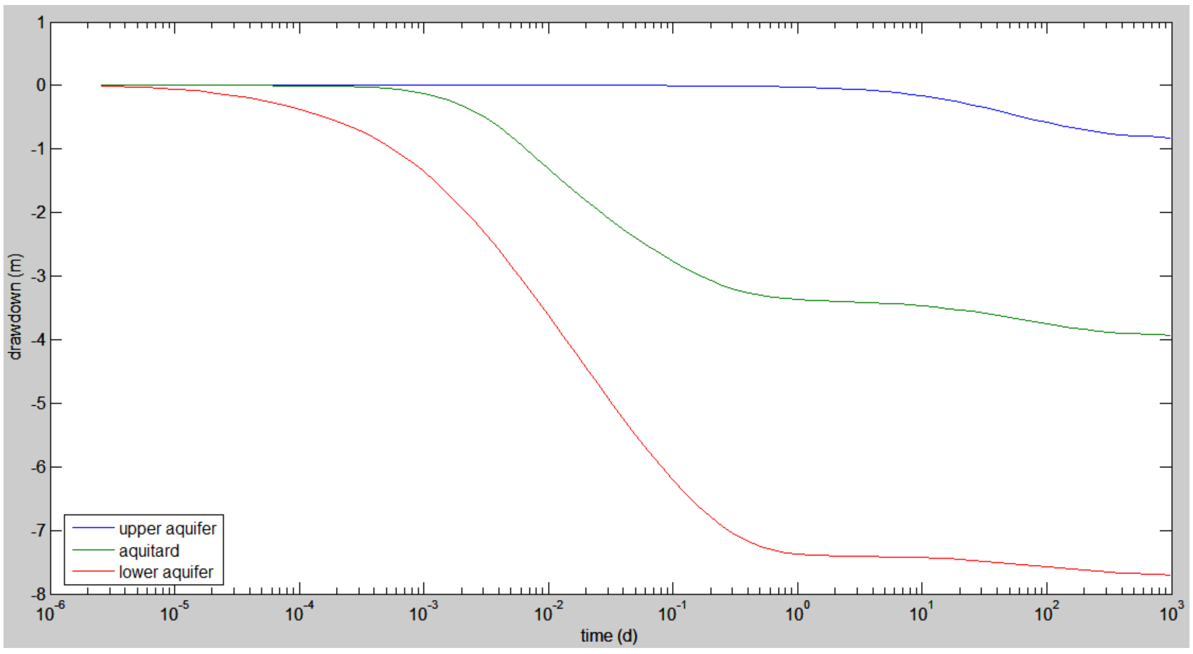


Figure 69. Drawdown as a function of time due to an extracted well field located in an unconfined two-layer aquifer system with recharge. Drawdown is calculated at the center of the circular recharge area using MAxSym and applying superposition.

Variable-discharge wells near infiltration pond

To illustrate the use of MAxSym for the analysis of groundwater management problems, an example is given of a two-aquifer system extracted by two wells near an infiltration pond. The latter is used to increase the recharge into the aquifer system and to protect it against over-exploitation.

The aquifer system is unconfined, has an impervious lower boundary, and consists of two aquifers separated by an aquitard. The hydraulic properties of the aquifers and aquitard are:

- upper aquifer: D = 20 m, K^h = 10 m/d, S^s = 1e-4 m^{-1}, S^y = 0.2;
- aquitard: C^v = 1000 d;
- lower aquifer: D = 30 m, K^h = 20 m/d, S^s = 1e-5 m^{-1};

where D denotes thickness, K^h horizontal conductivity, C^v vertical resistance, S^s specific storage coefficient, and S^y specific yield.

Two extraction wells are located in the lower aquifer. The distance between the wells is 30 m. The well discharges are time-varying and have a daily pattern: the first well is extracted at a rate of 500 m³/d for 8 hours and is shut down for the rest of the day, the second well is extracted at a rate of 1500 m³/d for 12 hours and at a rate of 500 m³/d for the next 12 hours. At the top of the aquifer system, a circular infiltration pond with radius of 10 m is constructed in between the two wells. The infiltration flux is constant and equal to 0.5 m/d. To predict the long-term effect, a period of 3 years is simulated.

Script "..\Examples\Superposition\infiltrationPond.m" implements this example. The following statements declare input variables and create a figure containing a map showing the location of extraction wells and infiltration pond and a plot of the daily pattern of well discharges (Figure 70):

```
% INPUT

% wells and infiltration pond
[x1,y1,Q1,d1] = deal(-15,0,500,[1 2]/3);
[x2,y2,Q2,d2,f2] = deal(15,0,1000,[1 1]/2,[1.5 0.5]);
N = 0.5;
R = 10;

% aquifer
D = [20;30];
kr = [10;20];
cz = 1000;
ss = [1e-4;1e-5];
sy = 0.2;

% number of days
days = (3*365)+1;

% map and discharge plot
figure
subplot(121)
plot(x1,y1,'ko',x2,y2,'ko')
text(x1,y1,'   Q1')
text(x2,y2,'   Q2')
hold on
a = linspace(0,2*pi,100);
plot(R*cos(a),R*sin(a),'b-')
axis equal
xlim([-20 20])
ylim([-20 20])
set(gca,'fontsize',12)
xlabel('x (m)')
ylabel('y (m)')

subplot(122)
h = plot([0 d1([1 1]) 1]*24,[Q1 Q1 0 0],...
         [0 d2([1 1]) 1]*24,Q2*f2([1 1 2 2]));
set(h,'linewidth',3)
xlim([0 24])
ylim([-500 2000])
grid minor
set(gca,'fontsize',12)
xlabel('time (h)')
ylabel('Q (m³/d)')
```

```
    legend('Q1','Q2')
```

Note that the *(x,y)* coordinate of the infiltration pond center is (0,0). The *(x,y)* coordinates of the wells are (-15,0) and (15,0) respectively. An extra day is added to the simulation time to facilitate linear interpolation of drawdown as a function of time.

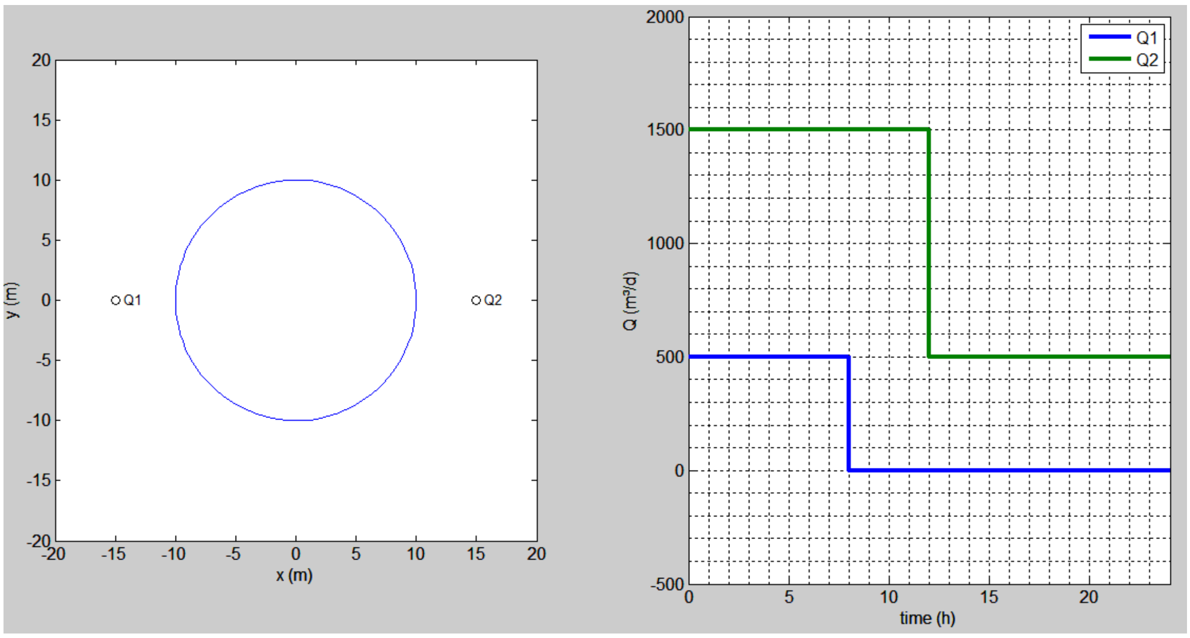


Figure 70. Left: map showing locations of the two extraction wells (small circles) and the infiltration pond (large circle). Right: graph showing the daily pattern of the well extraction rates. Wells are located in the lower aquifer of the unconfined two-aquifer system.

Theoretically, only two axi-symmetric models are required to apply superposition: one model simulating drawdown due to areal infiltration and one model simulating drawdown due to unit discharge in the lower aquifer. Time-varying discharge for each well is obtained by applying superposition as was explained earlier. If *n* discharge periods are considered for well *k* located at *(x_k,y_k)*, and if discharge is constant and equal to *Q(i)* during period *i* starting at $t^Q(i-1)$ and ending at $t^Q(i)$, then drawdown $s^w{}_k$ due to extraction of well *k* in layer *j* at point *(x,y)* and time *t* is calculated as:

$$s_k^W(x,y,j,t) = \sum_{i=1}^{n}[Q(i) - Q(i-1)]\ s_u(\sqrt{(x-x_k)^2 + (y-y_k)^2}, j, t - t_{(i-1)}^Q)$$

where both $t^Q(0)$ and *Q(0)* are zero, and $s_u(r,j,t)$ is the drawdown in layer *j* at distance *r* and time *t* due to unit discharge. Total drawdown *s* in layer *j* at point *(x,y)* and at time *t* is given by:

$$s(x,y,j,t) = s^r\left(\sqrt{(x-x_r)^2 + (y-y_r)^2}, j, t\right) + \sum_{i=1}^{2} s_k^W(x,y,j,t)$$

where $s^r(r,j,t)$ is the drawdown in layer *j* at distance *r* and at time *t* due to infiltration from the circular pond with center at *(x_r,y_r)*.

Considering the large number of discharge changes for both wells, it is, however, more convenient to run a MAxSym model for each well that takes into account the many discharge changes. Doing so, drawdown s^w_k due to extraction of well *k* in layer *j* at point *(x,y)* and at time *t* is easily obtained from:

$$s_k^w(x,y,j,t) = s_k^a(\sqrt{(x-x_k)^2+(y-y_k)^2},j,t)$$

where $s^a_k(r,j,t)$ is drawdown in layer *j* at distance *r* and at time *t* according to the MAxSym model simulating the *k*-th well. The following statements run the three MAxSym models:

```
% MAXSYM MODELS

% recharge model
mr = MAxSym.Model;
mr.settime(diff(logspace(-5,log10(days),81)));
rb = logspace(-1,log10(R),21);
rb = [rb(1:end-1),logspace(log10(R),7,61)];
mr.setgrid(rb,D,false);
mr.par.kr = kr;
mr.par.cz = cz;
mr.par.ss = ss;
mr.par.sy = sy;
mr.stress.q = sparse(mr.grid.nz,mr.grid.nr);
mr.stress.q(1,1:20) = -N*mr.grid.hs(1:20);
mr.setsolver(1e-7,50,5);
mr.run;

% well 1 model
m1 = MAxSym.Model;
t = 0:8:(24*days);
t(1) = 1e-5;
t(3:3:end) = [];
dt = [];
for i = 1:length(t)-1
    dt = [dt, diff(logspace(log10(t(i)),log10(t(i+1)),51))];
end
dt = dt/24;
m1.settime(dt,repmat([50 50],1,days));
m1.setgrid(logspace(-1,7,81),D,false);
m1.par.kr = kr;
m1.par.cz = cz;
m1.par.ss = ss;
m1.par.sy = sy;
for k = 1:2:m1.time.nper
    m1.stress(k).q = sparse(m1.grid.nz,m1.grid.nr);
    m1.stress(k).q(2,1) = Q1;
end
m1.setsolver(1e-7,50,5);
m1.run;

% well 2 model
m2 = MAxSym.Model;
t = 0:12:(24*days);
t(1) = 1e-5;
dt = [];
for i = 1:length(t)-1
    dt = [dt, diff(logspace(log10(t(i)),log10(t(i+1)),51))];
end
```

```
dt = dt/24;
m2.settime(dt,repmat([50 50],1,days));
m2.setgrid(logspace(-1,7,81),D,false);
m2.par.kr = kr;
m2.par.cz = cz;
m2.par.ss = ss;
m2.par.sy = sy;
for k = 1:2:m2.time.nper
    m2.stress(k).q = sparse(m2.grid.nz,m2.grid.nr);
    m2.stress(k).q(2,1) = f2(1)*Q2;
end
for k = 2:2:m2.time.nper
    m2.stress(k).q = sparse(m2.grid.nz,m2.grid.nr);
    m2.stress(k).q(2,1) = f2(2)*Q2;
end
m2.setsolver(1e-7,50,5);
m2.run;
```

Discretization of time and radial distance is different for each model, and therefore, a new `Model` object is instantiated for each model. As required for the application of superposition, layers and parameters must be the same. Note that only two model layers are considered, and the aquitard is taken into account by assigning its vertical resistance to property `cz`. To define stress periods and time steps, a `for` loop was used to minimize rounding errors. The following statements are an alternative and faster way to define the time steps for the first well model, and theoretically, they should give the same result:

```
>> dt = [diff(logspace(-5,log10(1/3),51)),...
         diff(logspace(-5,log10(2/3),51))];
>> dt = repmat(dt,1,days);
```

However, if we take the sum of all time steps, we notice a significant rounding error comparing the result to the input simulation time of 1096 days:

```
>> format long
>> sum(dt)

ans =

    1.095978080000056e+003
```

whereas the statements used in the script yield a more accurate result:

```
>> t = 0:8:(24*days);
>> t(1) = 1e-5;
>> t(3:3:end) = [];
>> dt = [];
>> for i = 1:length(t)-1
       dt = [dt, diff(logspace(log10(t(i)),log10(t(i+1)),51))];
   end
>> dt = dt/24;
>> sum(dt)

ans =

    1.095999999583325e+003
```

In general, one should always pay attention to rounding errors, especially when interpolation is applied to calculated results. A good advice is to always verify simulation times and distances before processing calculated drawdown. Using the command `format long`, one is able to see more digits than the default `short` format, which only displays 4 digits after the decimal point. See MATLAB help for more information on output formats.

Now the three models are run, superposition may be applied to calculate total drawdown as a function of time at any arbitrary point. Time-drawdown graphs are created at the 3 points corresponding to the horizontal position of the extractions wells and the center of the infiltration pond. Drawdown is calculated each hour for the entire simulation period of 3 years:

```
% TIME-DRAWDOWN GRAPHS

% input and initialization
[x,y] = deal([x1;x2;0],[y1;y2;0]);
[xb,yb] = deal(x,y);
t = (1:(24*(days-1)))/24;
s = zeros(2,3,length(t));
m = {m1; m2; mr};

% interpolation and superposition
for n = 1:length(m)
    r = sqrt((x-xb(n)).^2+(y-yb(n)).^2);
    s = s + m{n}.interp2([],r(:),t(:));
end
s = reshape(permute(s,[3 2 1]),[],6);

% plots
figure
b = 8:24:length(t);
plot(t(b),s(b,:))
hold on
b = 24:24:length(t);
plot(t(b),s(b,:))
xlim(t([1 end]))
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
str = sprintf('layer %1d: (%d,%d);',...
              [[ones(3,1);2*ones(3,1)],repmat([x(:) y(:)],2,1)]');
str = regexp(str,';','split');
legend(str(1:end-1),'location','northeastoutside');

figure
b = t >= days-2;
plot((t(b)-days+2)*24,s(b,:),'-x')
xlim([0 24])
set(gca,'fontsize',12)
xlabel('time (h)')
ylabel('drawdown (m)')
grid minor
legend(str(1:end-1),'location','northeastoutside');
```

Two time-drawdown plots are created: the first shows minimum and maximum drawdown as a function of time in the two layers at the specified points (Figure 71), and the second shows drawdown as a function of time for the last day only (Figure 72).

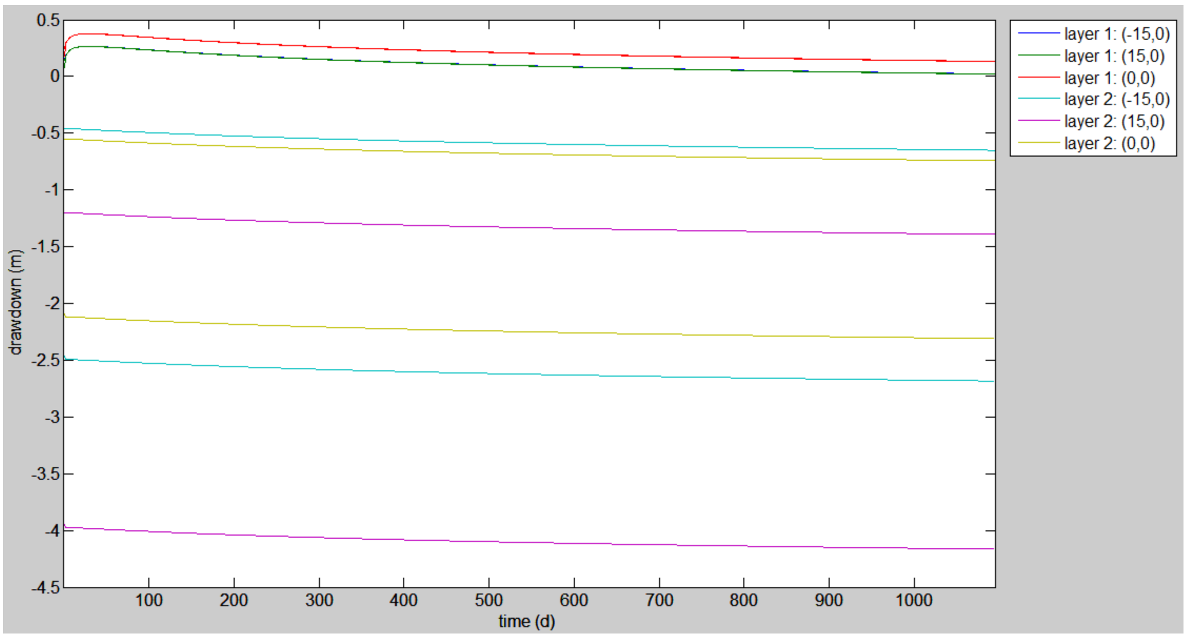


Figure 71. Minimum and maximum drawdown as a function of time at the positions of the extraction wells and the center of the infiltration pond. The results are calculated on an hourly basis using MAxSym and applying superposition.

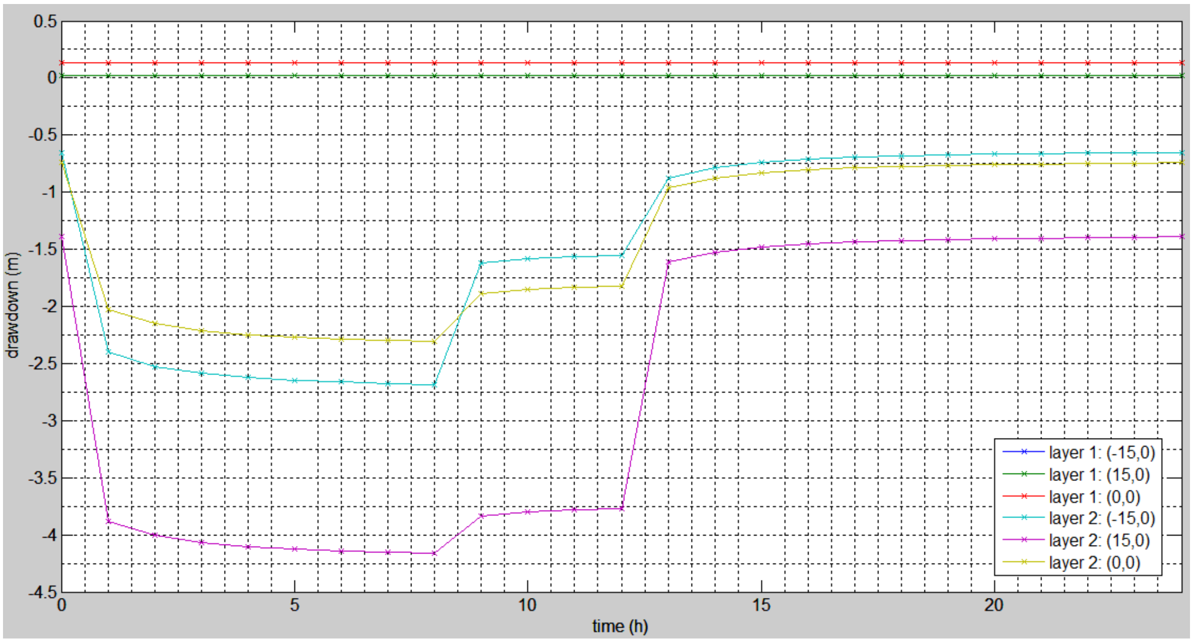


Figure 72. Drawdown as a function of time for the last simulation day at the positions of the extraction wells and the center of the infiltration pond. The results are calculated on an hourly basis using MAxSym and applying superposition.

Finally, a distance-drawdown plot is created for the last day of the simulation period, showing minimum and maximum drawdown in both aquifers. Therefore, a cross-section is defined at *y* = 0, extending from *x* = -1000 to *x* = 1000:

```
% DISTANCE-DRAWDOWN PLOT

% input and initialization
[x,y] = meshgrid(-1000:1000,0);
t = (days-1) - [d1(2) 0];
s = zeros(2,numel(x),2);

% interpolation and superposition
for n = 1:length(m)
    r = sqrt((x-xb(n)).^2+(y-yb(n)).^2);
    b = r > m{n}.grid.r(1);
    s = s + m{n}.interp2([],r(:),t(:));
end
s = reshape(permute(s,[2 3 1]),[size(x) 4]);

% plot
figure
plot(x,squeeze(s))
set(gca,'fontsize',12)
xlabel('x (m)')
ylabel('drawdown (m)')
str = sprintf('layer %1d: %d hours;',...
              [[1 1 2 2];repmat([d1(1) 1]*24,1,2)]);
str = regexp(str,';','split');
legend(str(1:end-1),'location','southeast');
```

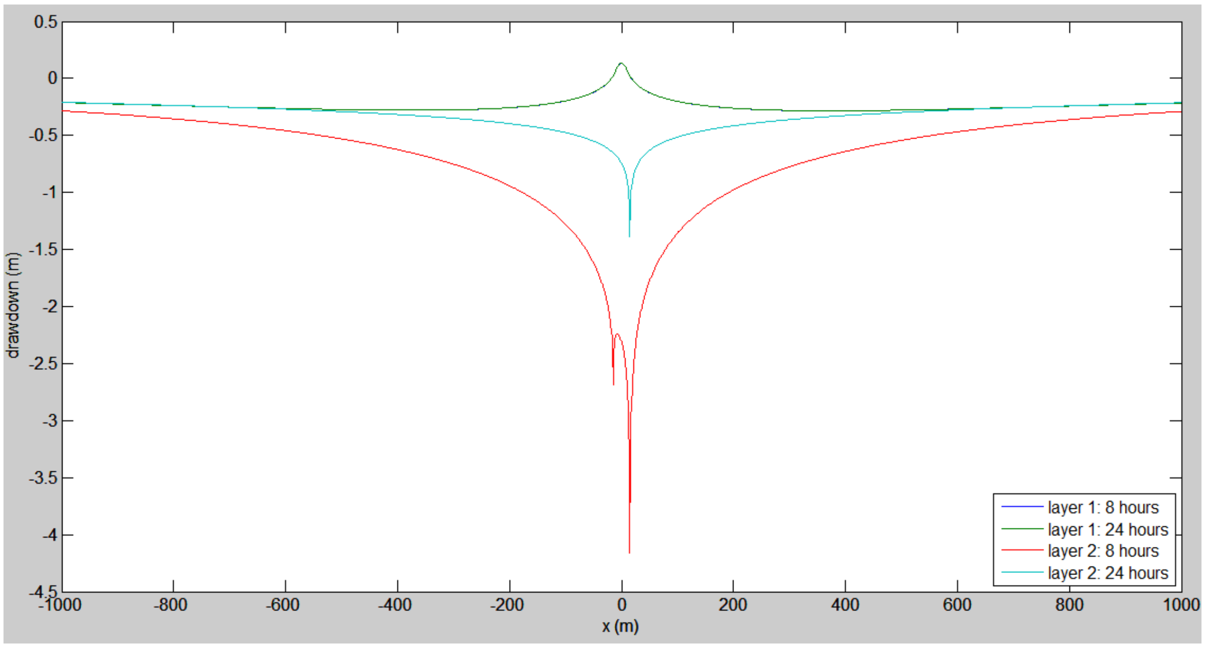


Figure 73. Drawdown as a function of distance at the last day of the simulation period for a cross section at y = 0 through the center of the infiltration pond and the extraction wells. Maximum drawdown at 8 hours and minimum drawdown at 24 hours is shown. Results are calculated using MAxSym and applying superposition.

The resulting plot is shown in Figure 73. Bilinear interpolation is applied using method `m.interp2`, because distances and times do not coincide with the radial distances and simulation times defined for the MAxSym models. A cell array is used to store the three `Model` objects. Because the `Model` class is a `handle` class, the cell array only contains a reference to the objects and data are not duplicated.

The combined system of extraction and infiltration may be optimized to maximize total extraction subject to a set of drawdown constraints. The latter could be defined, for instance, inside an ecologically valuable area close to the extraction wells. This optimization problem is solved by an inverse model that applies linear programming. The technique of linear programming is briefly discussed in the next section.

Inverse models

All models presented in previous examples are forward models. Applying these models, it is assumed model parameters and boundary conditions are known and data are derived from this information. Inverse models are applied to do the opposite: unknown parameters and/or boundary conditions are derived from data using a forward model coupled to an optimization algorithm.

Interpretation of aquifer tests is an example of inverse modeling. Observations measured during test performance are fitted using a mathematical model and a curve fitting method. In former days, the curve fitting was done using type curves. Nowadays, it is more convenient to apply an optimization algorithm, and an interactive computing environment such as MATLAB lends itself to the analysis of aquifer tests. In fact, the MathWorks offers an optimization toolbox containing many optimization algorithms implemented in easy-to-use functions. Unfortunately, this toolbox is not provided with the standard MATLAB environment and it must be purchased at extra cost. On the other hand, implementation of a standard non-linear regression algorithm is straightforward, and several more advanced optimization algorithms coded with MATLAB can be found on the internet and are downloadable free of charge.

This section explains how to combine MAxSym with a non-linear regression method for the interpretation of pumping and slug tests. MAxSym is used as forward model for the simulation of drawdowns corresponding to observed drawdowns. The non-linear regression method is applied to find the set of model parameters that gives the best fit of the observations. The Gauss-Newton and the related Levenberg-Marquardt methods (Levenberg, 1944; Marquardt, 1963) are discussed and applied for two examples illustrating the analysis of a pumping test and a slug test.

In general, the idea behind both regression methods is to approximate the non-linear model by a linear one, and to refine the parameters by successive iterations until the model fits best the observations in the least squares sense. This means the objective function to minimize is the sum of squared residuals defined as:

$$O = \sum_{i=1}^{n} \varepsilon_i^2$$

where n is the number of observations and ε_i is the residual of the i-th observation. In case of a pumping or slug test, the residual may be defined as:

$$\varepsilon_i = s_i^o - s_i^c$$

with s^o and s^c the observed and calculated drawdown respectively. Calculation of residuals and sum of squares requires one model run, performed at the start of the iteration process using the initial parameter estimates, and at the end of each iteration, after the parameters have been adjusted according to:

$$\bar{\beta} = (J^T J + \mu I)^{-1}\ (J^T \bar{\varepsilon})$$

where $\bar{\beta}$ is the *m x 1* vector containing the adjustment factors for the *m* parameters being optimized, *J* is the *n x m* sensitivity matrix which will be explained below, J^T is the transpose of the sensitivity matrix, *μ* is a damping factor, *I* is the identity matrix, and $\bar{\varepsilon}$ is an *n x 1* vector with residuals calculated at the end of the previous iteration. If the damping factor is zero, the above expression is simplified to the Gauss-Newton formulation, if it is greater than zero, the expression is referred to as the Levenberg-Marquardt algorithm.

Sensitivity matrix *J* is a finite-difference approximation of the Jacobian matrix and $J^T J$ is an approximation of the Hessian matrix. Sensitivity J_{ij} of the *i*-th drawdown s^c_i to the *j*-th parameter p_j is defined as:

$$J_{ij} = \frac{s_i^c(p_j + \Delta p) - s_i^c(p_j)}{\Delta p}$$

where *Δp* is the step size. In case of pumping and slug test interpretation, it is common practice to consider the logarithm of the aquifer parameters in the above expression. To calculate the sensitivity matrix, one needs to perform *m* model runs and sensitivity to the *j*-th parameter is calculated during the *j*-th run by augmenting its logarithmic value by *Δp*, whereas the other parameter values remain unaltered. Drawdown $s^c(p_j+\Delta p)$ is obtained from this run, while drawdown $s^c(p_j)$ has already been calculated when residuals were determined.

After the adjustment factors are calculated, parameter values are updated as $p_j + \bar{\beta}_j$ and a final model run is performed to calculate residuals and sum of squares. The latter is verified against the following criterion for convergence:

$$\left|\frac{O_{k-1} - O_k}{O_{k-1}}\right| \leq \delta$$

where *k* denotes the iteration index and *δ* is a small positive value specified by the user. If a damping factor is used, then it is common to adjust it according to the new sum of squared residuals O_k. The iteration process is terminated if the stop criterion is satisfied or when a specified maximum number of iterations is reached.

Summarizing, the non-linear regression algorithm consists of the following steps:

0. Calculate residuals and sum of squares considering initial parameter estimates
1. Calculate sensitivity matrix
2. Calculate adjustment factors and adjust parameter values
3. Calculate residuals and sum of squares considering adjusted parameter values

4. Check criterion for convergence:
    a. If criterion is not satisfied: go to step 1 to start next iteration
    b. If criterion is satisfied: optimal parameter estimates are found

If the maximum number of iterations is reached before the stop criterion is satisfied, the system has not converged to an optimal parameter set, which could indicate the problem is ill-posed. If the system is well-posed, then a unique and stable solution exists and an acceptable approximation to an optimal parameter set can be found that satisfies the criterion for convergence.

If a solution is obtained, it is recommended to verify whether the system is well-conditioned or not. This can be done by calculating the condition number κ of sensitivity matrix *J*:

$$\kappa = \frac{\sigma_{max}}{\sigma_{min}}$$

where σ_{max} and σ_{min} are maximal and minimal singular values of *J* respectively, which can be calculated using MATLAB function `svd`. The system is well-conditioned if the condition number is small, and ill-conditioned if the condition number is large. In the latter case, many local minima could exist and one may consider applying a global optimization algorithm to find the objective function's global minimum, since the presented regression methods are gradient-based and cannot distinguish the global minimum from local minima. On the other hand, ill-conditioned systems may be avoided by careful test design and sufficiently monitoring of the test.

After optimal values are estimated, it is also good practice to determine parameter uncertainties. Uncertainty is related to the residuals, which are a combination of measurement errors and errors arising from the mathematical model, but also the conditioning of the system plays an important role. A detailed discussion of uncertainty analysis is however out of the scope of this manual, as is an exposition on the application of different optimization algorithms with relation to the problem of non-uniqueness.

Before presenting the two examples, we would like to mention another type of inverse modeling. Superposition models can be coupled to a linear programming algorithm to maximize well discharges when a set of drawdown (or flux) constraints is given. Mathematically, the optimization is expressed as:

$$maximize\ w^T q$$

$$subject\ to\ Uq \leq c$$

$$and\ q \geq 0$$

where *q* is a vector with *n* well discharges, *w* is an *n x 1* weighting vector, *U* is the *m x n* matrix with drawdowns according to the unit discharge models, and *Uq* is the *m x 1* vector with superimposed drawdowns that must fulfill the constraints expressed by the *m x 1* vector *c*. Several algorithms are available to solve the system of equations, e.g. the simplex algorithm of Dantzig (1947). Some of them are provided with the MATLAB optimization toolbox. Using a linear programming function in combination with MAxSym makes the setup of simple groundwater management models with

MATLAB straightforward. Implementation of linear programming algorithms is, however, less straightforward, and therefore, out of the scope of this manual.

Interpretation of pumping test

A synthetic pumping test is simulated which is conducted in a fully penetrating well located in a homogeneous, confined aquifer with transmissivity of 100 m²/d and storativity of 1e-3. The well is extracted at a constant rate of 100 m³/d during a period of 1 day. Drawdown is measured each minute after starting the test in three fully penetrating observation wells at 1 m, 5 m, and 10 m from the well. The well diameter is small enough to neglect wellbore storage. A random error is superimposed on the simulated observations, where it is assumed the error is normally distributed with zero mean and standard deviation of 0.001 m. The MATLAB function `randn` is used to simulate the observation error. Script "..\Examples\Inverse\pumpingTest.m" implements this example:

```
% SYNTHETIC PUMPING TEST

% MAxSym model
m = MAxSym.Model;
m.settime(diff(logspace(-5,0.1,101)));
m.setgrid(logspace(-1,7,81),1,true);
m.par.kr = 100;
m.par.ss = 1e-3;
m.stress.q = [100, zeros(1,m.grid.nr-1)];
m.setsolver(1e-5,1);
m.run;

% observations
[tobs,robs] = deal((1:1440)/1440,[1 5 10]);
sobs = squeeze(m.interp2(1,robs,tobs))';
sobs = sobs + 0.001*randn(size(sobs));
```

Applying the Gauss-Newton algorithm, aquifer transmissivity and storativity are derived from these observations. The initial parameters are estimated as 10 m²/d and 1e-4 respectively. The regression parameters that must be specified by the modeler are the step size, which is set equal to 0.1, the stop criterion, which is set equal to 1e-5, and the maximum number of iterations, which is set equal to 50:

```
% INTERPRETATION

% regression parameters
dp = 10^0.1;
delta = 1e-5;
mni = 50;

% initial run
n = 0;
m.par.kr = 10;
m.par.ss = 1e-4;
m.run;
s = squeeze(m.interp2(1,robs,tobs))';
eta = sobs(:)-s(:);
ssr = eta' * eta;
dssr = delta + 1;
```

```
% echo
fprintf(1,'\nInitial run\n');
fprintf(1,' T = %e\n',m.par.kr)
fprintf(1,' S = %e\n',m.par.ss)
fprintf(1,' SSR = %e\n',ssr);

% iterations
while n < mni && dssr > delta

    % sensitivities
    m.par.kr = m.par.kr*dp;
    m.run;
    tmp = squeeze(m.interp2(1,robs,tobs))';
    J(:,1) = (tmp(:)-s(:))/log10(dp);
    m.par.kr = m.par.kr/dp;

    m.par.ss = m.par.ss*dp;
    m.run;
    tmp = squeeze(m.interp2(1,robs,tobs))';
    J(:,2) = (tmp(:)-s(:))/log10(dp);
    m.par.ss = m.par.ss/dp;

    % condition number
    [~,v] = svd(J);
    v = diag(v);
    k = max(v)/min(v);

    % adjusting parameters
    B = 10.^((J'*J)\(J'*eta));
    m.par.kr = m.par.kr*B(1);
    m.par.ss = m.par.ss*B(2);

    % residuals and sum of squares
    m.run;
    s = squeeze(m.interp2(1,robs,tobs))';
    eta = sobs(:)-s(:);
    dssr = eta' * eta;
    [dssr,ssr] = deal(abs((ssr-dssr)/ssr),dssr);

    % iteration index
    n = n + 1;

    % echo
    fprintf(1,'\nIteration %d\n',n)
    fprintf(1,' condition number = %.2f\n',k)
    fprintf(1,' T = %e\n',m.par.kr)
    fprintf(1,' S = %e\n',m.par.ss)
    fprintf(1,' SSR = %e\n',ssr);

end

% plot
figure
plot(tobs',sobs,'k-',tobs',s,'r-')
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('drawdown (m)')
```

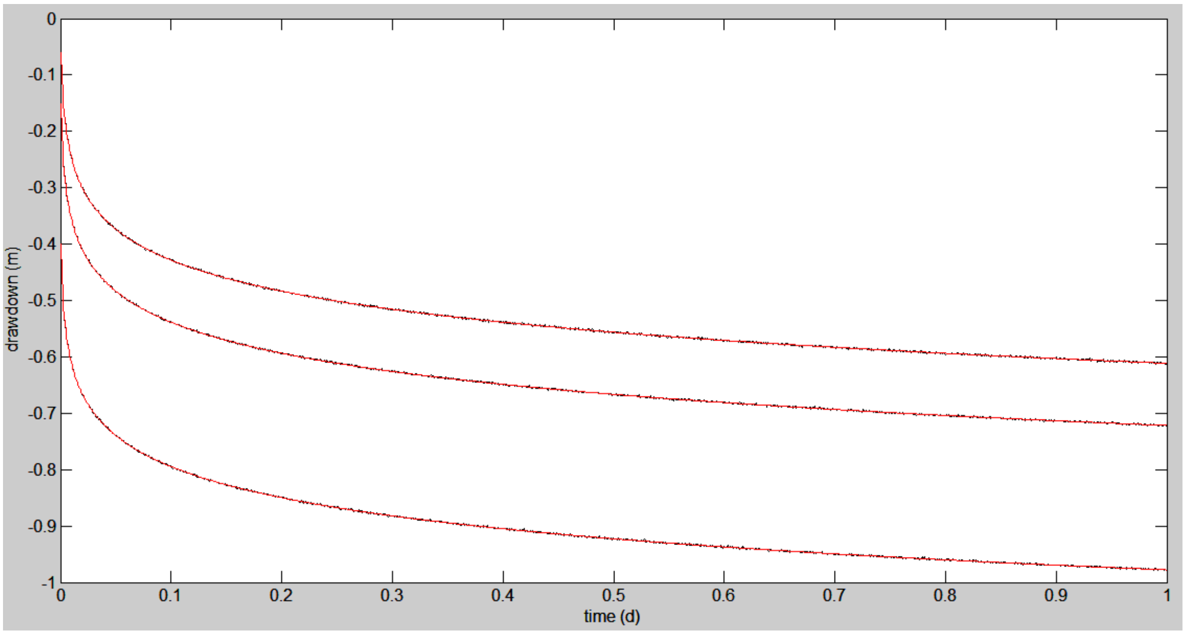


Figure 74. Time-drawdown graph for the interpreted pumping test comparing observed drawdowns (black) sampled from observation wells at 1 m, 5 m, and 10 m with corresponding drawdowns simulated using the optimal parameter estimates (red). The latter are derived applying MAxSym coupled to the Gauss-Newton algorithm.

The parameter estimates are printed each iteration, as is the sum of squared residuals and the condition number. The latter indicates the system is well-conditioned and the system only needs few iterations to find an optimal parameter set which is very close to the parameter set used for the simulation of the synthetic test. After termination of the regression algorithm, a time-drawdown graph is plotted comparing observed drawdowns to drawdowns simulated using the optimal parameter estimates. Figure 74 shows a very good fit is obtained.

Interpretation of slug test

A synthetic slug test is simulated which is conducted in a fully penetrating well with radius of 0.03 m and located in a homogeneous confined aquifer with transmissivity of 1 m²/d and storativity of 1e-5. The initial head change within the well is 1 m. Drawdown within the well is measured each second after starting the test for a period of 1e4 seconds. A random error is superimposed on the simulated observations using function `randn`. The error is assumed normally distributed with zero mean and standard deviation of 0.001 m. Script “..\Examples\Inverse\slugTest.m” implements this example:

```
% SYNTHETIC SLUG TEST

% MAxSym model
m = MAxSym.Model;
m.settime(diff(logspace(-6,1,251)));
rb = logspace(log10(0.03),3,401);
rb = [rb(1)^2/rb(2),rb];
m.setgrid(rb,1,true);
m.par.kr = 1;
m.par.ss = [rb(2)^2*pi/m.grid.vol(1),1e-5*ones(1,m.grid.nr-1)];
m.stress.s0 = [1,zeros(1,m.grid.nr-1)];
m.setsolver(1e-5,1);
m.run;
```

```
% observations
tobs = (1:1e4)/1440/60;
sobs = squeeze(m.interp1t(1,1,tobs));
sobs = sobs + 0.001*randn(size(sobs));
sobs(sobs<0) = 0;
```

Aquifer transmissivity and storativity are derived from these observations applying the Levenberg-Marquardt algorithm. The initial parameters are estimated as 10 m²/d and 1e-3 respectively. The regression parameters that must be specified by the modeler are the step size, which is set equal to 0.1, the stop criterion, which is set equal to 1e-5, the maximum number of iterations, which is set equal to 50, and the damping factor, which is equal to 10:

```
% INTERPRETATION

% regression parameters
dp = 10^0.1;
delta = 1e-5;
mni = 50;
mu = 10;

% initial run
n = 0;
m.par.kr = 10;
m.par.ss(2:end) = 1e-3;
m.run;
s = squeeze(m.interp1t(1,1,tobs));
eta = sobs(:)-s(:);
ssr = eta' * eta;
dssr = delta + 1;

% echo
fprintf(1,'\nInitial run\n');
fprintf(1,' T = %e\n',m.par.kr)
fprintf(1,' S = %e\n',m.par.ss(end))
fprintf(1,' SSR = %e\n',ssr);

% iterations
while n < mni && dssr > delta

    % sensitivities
    m.par.kr = m.par.kr*dp;
    m.run;
    tmp = squeeze(m.interp1t(1,1,tobs));
    J(:,1) = (tmp(:)-s(:))/log10(dp);
    m.par.kr = m.par.kr/dp;

    m.par.ss(2:end) = m.par.ss(2:end)*dp;
    m.run;
    tmp = squeeze(m.interp1t(1,1,tobs));
    J(:,2) = (tmp(:)-s(:))/log10(dp);
    m.par.ss(2:end) = m.par.ss(2:end)/dp;

    % condition number
    [~,v] = svd(J);
    v = diag(v);
    k = max(v)/min(v);
```

```
        % adjusting parameters
        H = J'*J;
        B = 10.^((H+mu*eye(size(H)))\(J'*eta));
        m.par.kr = m.par.kr*B(1);
        m.par.ss(2:end) = m.par.ss(2:end)*B(2);

        % residuals and sum of squares
        m.run;
        s = squeeze(m.interp1t(1,1,tobs));
        eta = sobs(:)-s(:);
        dssr = eta' * eta;
        decrease = dssr < ssr;
        [dssr,ssr] = deal(abs((ssr-dssr)/ssr),dssr);

        % iteration index
        n = n + 1;

        % echo
        fprintf(1,'\nIteration %d\n',n)
        fprintf(1,' condition number = %.2f\n',k)
        fprintf(1,' T = %e\n',m.par.kr)
        fprintf(1,' S = %e\n',m.par.ss(end))
        fprintf(1,' SSR = %e\n',ssr);
        fprintf(1,' damping factor = %e\n',mu);

        % damping factor
        if decrease
            mu = mu/2;
        else
            mu = mu*2;
        end

    end

    % plot
    figure
    semilogx(tobs,sobs,'k-',tobs,s,'r-')
    set(gca,'fontsize',12)
    xlabel('time (d)')
    ylabel('drawdown (m)')
```

The parameter estimates are printed each iteration, as is the sum of squared residuals, the condition number, and the damping factor. The latter is modified at the end of each iteration: if the sum of squares in current iteration is smaller than the sum of squares in previous iteration, the damping factor is divided by 2, else, the factor is multiplied by 2.

An optimal parameter set is found which is very close to the parameter set used for the simulation of the synthetic test. The time-drawdown graph comparing observed drawdowns to drawdowns simulated using the optimal parameter estimates confirms a very good fit is obtained (Figure 75). However, the large condition number indicates the system is ill-conditioned. From literature, we know sensitivity of drawdown within the well to aquifer storativity is small and a high correlation exists between sensitivity to storativity and sensitivity to transmissivity. This also explains why the Levenberg-Marquardt formulation is used instead of Gauss-Newton: in most cases, the latter fails in converging to a solution if both transmissivity and storativity are optimized.

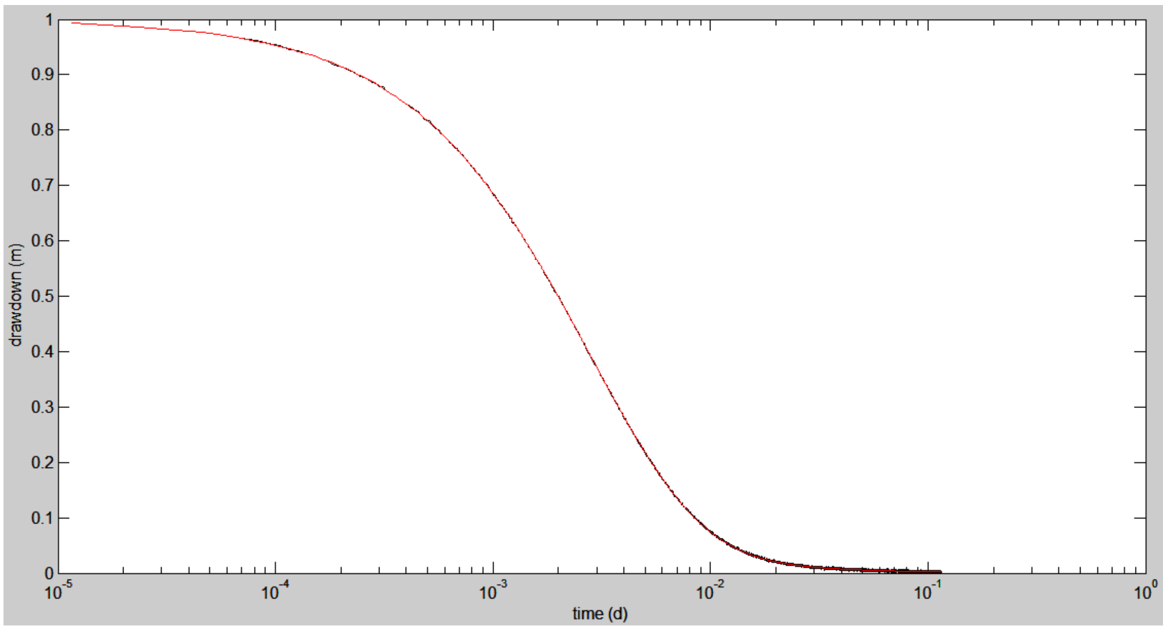


Figure 75. Time-drawdown graph for the interpreted slug test comparing observed drawdowns (black) with corresponding drawdowns simulated using the optimal parameter estimates (red). The latter are derived applying MAxSym coupled to the Levenberg-Marquardt algorithm.

The high correlation between drawdown sensitivities to transmissivity and storativity is illustrated by the sensitivity plot shown in Figure 76, which is created by the following statements:

```
% sensitivity plot
figure
semilogx(tobs,J)
set(gca,'fontsize',12)
xlabel('time (d)')
ylabel('sensitivity')
legend('T','S')
```

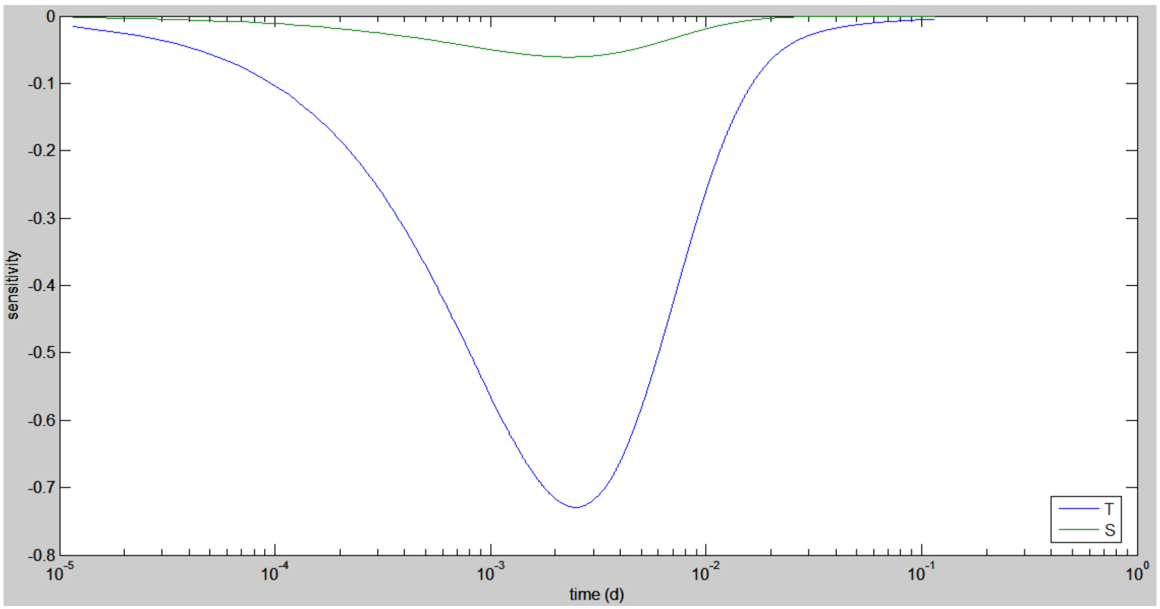


Figure 76. Plot of sensitivity of drawdown within the slugged well to aquifer transmissivity and aquifer storativity as a function of time for the interpreted slug test.

Summary

The examples illustrate the many applications covered by MAxSym. All kinds of axi-symmetric models can be simulated, and existing analytical solutions are approximated to great accuracy. Implementation is straightforward and limited to a dozen lines of code, computation time is small. Results can be visualized easily using the many visualization tools provided with the MATLAB environment.

Examples are given of steady state, axi-symmetric models which simulate flow due to extraction in confined, unconfined, and leaky, single- and multi-aquifer systems. Examples of transient, axi-symmetric models involve simulation of all kinds of aquifer tests in confined, unconfined, and leaky, single- and multi-aquifer systems of infinite lateral extent: constant-discharge pumping tests, variable-discharge pumping tests such as step-drawdown tests, push-pull tests, recovery tests considering recoverable or non-recoverable compression, slug tests, slug interference tests, multiple pumping tests such as hydraulic tomography, and multi-level slug tests. Most examples deal with laterally isotropic aquifer systems, but lateral anisotropy of hydraulic conductivity can be taken into account using apparent distances.

It is shown how to implement fully and partially penetrating wells of small and large diameter. Including wellbore storage is required for head-specified wells and large-diameter wells. An infinitesimally small skin as well as a finite-thickness skin may be considered, as also impervious well casing and packers within the well. The effect of non-linear well losses is demonstrated for pumping tests; the slug test examples are limited to over-damped cases.

If the system is unconfined, MAxSym takes into account the time-varying saturated thickness of the top layer. Applying the Dupuit (1863) assumption, however, results into a lower water table in the vicinity of pumped wells. Flow in the unsaturated zone is ignored, but it is possible to define areal infiltration in the top layer. The effect of the water table is simulated by considering specific yield. Care should be taken, however, since MAxSym only allows the upper layer to be phreatic. Consequently, the phreatic top layer cannot be discretized sufficiently to simulate accurately the vertical movement of the water table.

The principle of superposition is demonstrated and examples are given of simple, three-dimensional models for multiple wells in multi-aquifer systems of infinite lateral extent. Steady-state models only involve leaky aquifer systems, whereas transient models can be build for confined, leaky, and unconfined aquifer systems. The effect of areal infiltration may also be superimposed to the well solutions. Including lateral anisotropy is possible again using apparent distances. The method of images can be applied to include lateral no-flow and constant-head boundaries.

Finally, the theory behind inverse modelling is discussed briefly. Superposition models can be coupled to a linear programming algorithm to maximize well discharges subject to a set of drawdown constraints. Some algorithms are provided with the MATLAB optimization toolbox. Interpretation of aquifer tests can be done by coupling MAxSym to a non-linear regression algorithm. Gauss-Newton and Levenberg-Marquardt are discussed and two examples are given illustrating the analysis of a synthetic pumping test and a synthetic slug test.

Developers' guide

All examples in previous chapter were implemented using MATLAB scripts. This works fine for simple models which only require a dozen lines of code. It is, however, good programming practice to organize code when programs become large. In traditional procedural MATLAB programming, functions are used to structure programs. For instance, a generic plot function could be written to create a time-drawdown graph for any transient MAxSym model. This function would not only improve the readability of the example scripts, but it would also avoid copying lines of code. Another advantage of functions is they have a local workspace in which local variables are encapsulated and separated from variables created in the base workspace.

Encapsulation is also a key feature with object-oriented programming, but additional mechanisms such as inheritance and aggregation are available to structure code and increase reusability. For instance, a graph class could be defined which inherits from class `Model` and implements several methods to create drawdown plots. Alternatively, aggregation could be applied by defining a property in the graph class that contains a `Model` object. Discussing the fundamental concepts of object-oriented programming is, however, out of the scope of this manual. This chapter only aims to fully describe the design and implementation of MAxSym. In this way, developers should be able to easily extend the MAxSym code by applying the object-oriented mechanisms provided with MATLAB.

MAxSym package: overview

MAxSym's class diagram has already been presented in Figure 5. Figure 77 shows the complete class diagram, including MAxSym functions and superclasses `Check` and `handle`. The latter is the built-in MATLAB superclass for all handle classes. If an instance of a handle class is copied, MATLAB copies the reference to the object only, the so called handle, and both the original and copy will refer to the same instance. Hence, modifying data from the copied object only affects the original object. The `Check` class belongs to the MAxSym package and it is a static class provided with protected, static methods, used by other MAxSym classes to perform input checking.

As already mentioned in chapter Modelers' guide, the `Model` class acts as mediator, which accepts input data from the modeler, coordinates the calculation process, and stores the output data. The input data are stored in other classes, of which the sole function is to check and store data input by the modeler or other applications. Part of the calculations is taken care of by the `Model` class, but solving the system of equations applying SIP or ADI is done by several MATLAB and MEX functions, which are written in a procedural fashion.

The calculation process is started when the modeler calls the `Model` object's `run` method. In case of a confined model, the `Model` object calls the MATLAB function `solveconfined`, in case of an unconfined model, the function `solveunconfined` is called. The required ADI or SIP function is called then by these functions according to the `Solver` object's properties set by the modeler:

- `mexADIconf`: ADI MEX routine for confined systems (`nparm <= 0 & mex`)
- `ADIconf`: ADI MATLAB routine for confined systems (`nparm <= 0 & ~mex`)
- `mexSIPconf`: SIP MEX routine for confined systems (`nparm > 0 & mex`)
- `SIPconf`: SIP MATLAB routine for confined systems (`nparm > 0 & ~mex`)

- `mexADIunconf`: ADI MEX routine for unconfined systems (`nparm <= 0 & mex`)
- `ADIunconf`: ADI MATLAB routine for unconfined systems (`nparm <= 0 & ~mex`)
- `mexSIPunconf`: SIP MEX routine for unconfined systems (`nparm > 0 & mex`)
- `SIPunconf`: SIP MATLAB routine for unconfined systems (`nparm > 0 & ~mex`)

Classes and functions belonging to the MAxSym package are discussed in detail in the next sections.

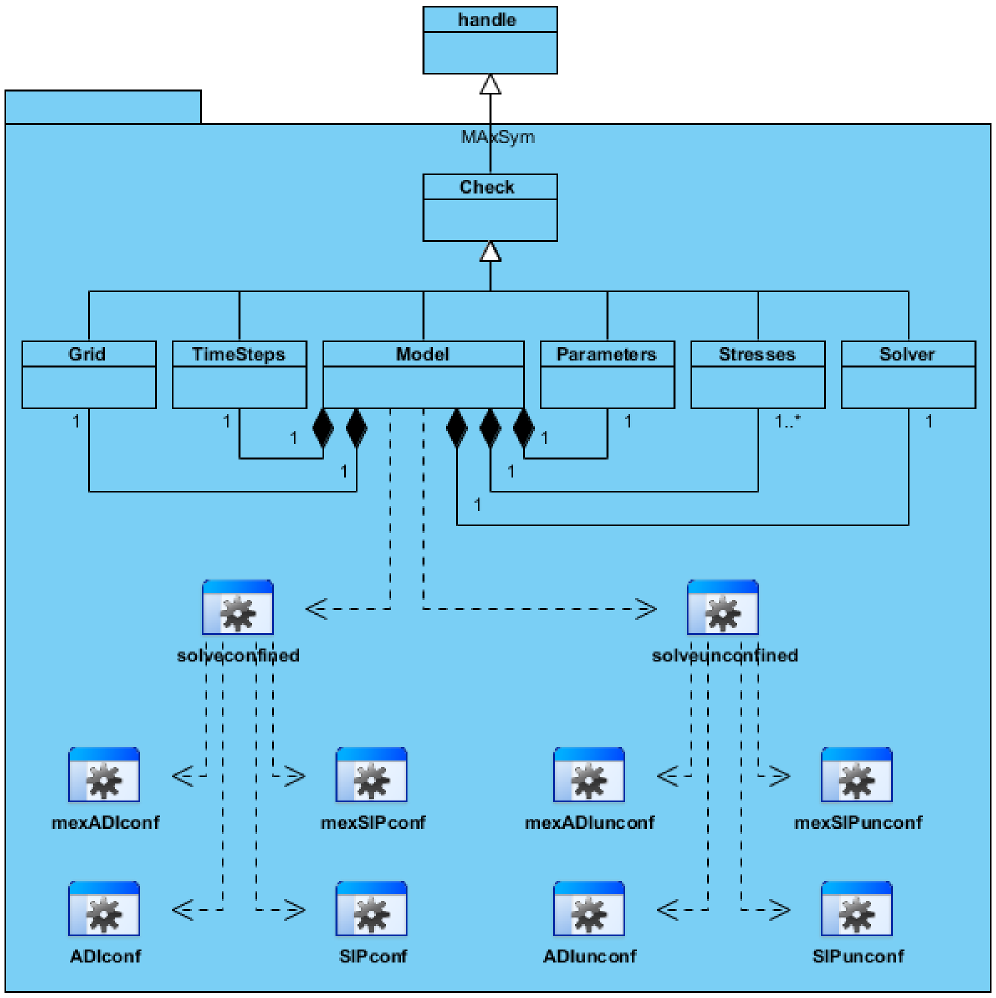
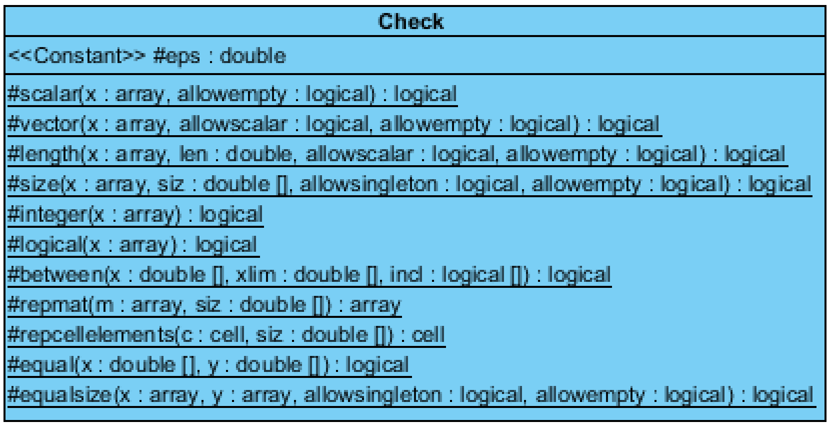


Figure 77. UML diagram presenting all classes and functions in the MAxSym package. Functions are indicated by a gear wheel symbol, dashed arrows indicate possible function calls.

Class Check

Most of the MAxSym input variables are numeric arrays of size determined by the model discretization. As a consequence, checking these input variables can be performed by generic functions. As is common in object-oriented designing, these functions are grouped into a single class. The `Check` class serves this purpose and contains the checking functions as static methods. Figure 78 shows the class diagram; Table 7 gives an overview. The `Check` class also has a constant property `eps`, which determines the maximum absolute difference used by method `equal` to test equality of floating point numbers. A full description of class `Check` is given in Appendix: members of class Check.



```
Check
<<Constant>> #eps : double
#scalar(x : array, allowempty : logical) : logical
#vector(x : array, allowscalar : logical, allowempty : logical) : logical
#length(x : array, len : double, allowscalar : logical, allowempty : logical) : logical
#size(x : array, siz : double [], allowsingleton : logical, allowempty : logical) : logical
#integer(x : array) : logical
#logical(x : array) : logical
#between(x : double [], xlim : double [], incl : logical []) : logical
#repmat(m : array, siz : double []) : array
#repcellelements(c : cell, siz : double []) : cell
#equal(x : double [], y : double []) : logical
#equalsize(x : array, y : array, allowsingleton : logical, allowempty : logical) : logical
```

Figure 78. UML class diagram for static class `Check`. Data types followed by `[]` indicate arrays, otherwise a scalar value is required. Datatype "array" refers to the MATLAB array, the object type that is the base for all other MATLAB data types. Hence, variables of type "array" may be of any data type.

All class members are static (or constant in case of property `eps`) and protected, hence, class `Check` itself may be considered as static and protected. As a consequence, the checking functions are accessible only to classes that inherit from class `Check`, which explains all other MAxSym classes are subclassing the latter. Mostly, the `Check` methods are called by the property setters of these subclasses. The technique of backing fields is applied to enable overriding of subclass setters and getters, as will be explained in the next section. The `Check` class also has two methods, `repmat` and `repcellelements`, called by the `Model` class to get full size input arrays.

Table 7. Overview of protected, static methods for class `Check`.

Method	**Description**
`scalar`	checks if input variable is scalar
`vector`	checks if input variable is vector
`length`	checks length of input vector
`size`	checks size of input array
`integer`	checks if input variable is numeric and contains integer elements only
`logical`	checks if input variable is logical
`between`	checks if elements of numeric input array lie between specified bounds
`repmat`	replicates input array and returns full size array
`repcellelements`	replicates array elements of input cell array and returns cell containing full size array elements
`equal`	checks equality of corresponding elements of input arrays using property `eps`
`equalsize`	checks if input arrays have equal size

Backing fields

In C#, a private or protected field that stores the data exposed by a public property is called a *backing store* or *backing field*. In MATLAB, no distinction is made between fields and properties for class members storing data. However, it is possible to define backing fields for public properties by using dependent properties. This is a powerful mechanism, since it allows overriding of setters and getters for inherited public properties.

The idiom to define a public property `x` with backing field `xback` for a parent class `Parent` is:

```
classdef Parent < handle

    properties (Dependent)
        x
    end

    properties (Access = private)
        xback
    end

    methods
        function x = get.x(obj)
            x = obj.getx;
        end
        function set.x(obj,x)
            obj.setx(x)
        end
    end

    methods (Access = protected)
        function x = getx(obj)
            <statements>
            x = obj.xback;
            <statements>
        end
        function setx(obj,x)
            <statements>
            obj.xback = x;
            <statements>
        end
    end

end
```

The public property `x` is defined `Dependent`, the backing field `xback` is defined `private`. Because property `x` is dependent, its getter `get.x` must be defined, and if it is not read-only, its setter `set.x` must be defined too. See MATLAB help for more information about dependent properties and property set and get access methods. The getter and setter cannot by overridden by child classes. Therefore, getter `get.x` calls the protected method `getx`, which can be overridden by a subclass. Similarly, setter `set.x` calls the protected method `setx`. The getter and setter statements are written in the protected methods, including getting the `x` value from the backing field or storing the `x` value into the backing field. The parent class inherits from `handle`, but this is not required. However, if the parent class is a value class, then getters, setters, and protected methods should be defined accordingly.

The idiom for a child class `Child` inheriting from class `Parent` and overriding the getter and setter for `x` is:

```
classdef Child < Parent

    methods (Access = protected)
        function setx(obj,x)
            <statements>
            obj.setx@Parent(x)
            <statements>
        end
```

```
            function x = getx(obj)
                <statements>
                x = obj.getx@Parent;
                <statements>
            end
        end

    end
```

The backing field `xback` may also have protected accessibility. In this case, the backing field is accessible by the child class and the overridden get and set methods do not have to call the parent class' get and set methods respectively. Sometimes, the value of a read-only property is dependent on the value of another property in the same class, and defining a backing field for the former would be redundant. In this case, both properties share the same backing field. As an example, class `Child` is subclassed by class `GrandChild`, which defines property `y` that is dependent on inherited property `x`:

```
    classdef GrandChild < Child

        properties (Dependent)
            y
        end

        methods
            function y = get.y(obj)
                y = gety(obj);
            end
        end

        methods (Access = protected)
            function y = gety(obj)
                <statements>
                x = obj.x;
                <statements>
                y = <expression>;
            end
        end

    end
```

If backing field `xback` in the `Parent` class is defined protected, it can be accessed directly by `gety`, which may be an advantage if `getx` performs a lot of calculations to return the `x` value.

Backing fields are defined for public properties in MAxSym classes storing input data. The naming convention applied in the above idiom for defining the parent class is also adopted:

- the backing field gets the name of the public property followed by the suffix 'back';
- the protected get method gets the name of the public property preceded by the prefix 'get';
- the protected set method gets the name of the public property preceded by the prefix 'set'.

The MAxSym backing fields are always protected, hence, they may be redefined by subclasses inheriting from MAxSym classes. An overview of MAxSym properties with backing fields is given in the next paragraphs.

Class TimeSteps

The class diagram for class TimeSteps is shown in Figure 8, and properties are summarized in Table 1. All properties are read-only, so only protected getters are implemented, applying the backing field technique (Table 8). Properties t and ndt have protected backing fields tback and ndtback respectively. Other properties are dependent on these properties directly or indirectly.

The backing fields are set by the TimeSteps constructor, which calls inherited Check methods to do the necessary checking on input dt and ndt. After object construction, the backing fields cannot be modified anymore. If time steps and stress periods are re-defined, a new TimeSteps object must be created. Recall that the settime method of the Model object calls the TimeSteps constructor. The TimeSteps class has no other methods defined but the constructor and property getters. A full description of class TimeSteps is given in Appendix: members of class TimeSteps.

Table 8. Properties for class TimeSteps. It is indicated whether protected getters and setters are implemented or not, and whether a backing field is present or not. Properties without backing field depend on other properties. The get statement indicates on which property or backing field properties are dependent.

Property	Getter	Setter	Backing field	Get statement
steady	yes	no	-	length(tback) == 1
t	yes	no	tback	tback
dt	yes	no	-	diff(t)
ndt	yes	no	ndtback	ndtback
tper	yes	no	-	t(cumsum([1;obj.ndt]))
dtper	yes	no	-	diff(obj.tper)

nper	yes	no	-	if transient: length(obj.ndt)
				if steady state: 1

Class Grid

The class diagram for class Grid is shown in Figure 9, and properties are summarized in Table 2. All properties are read-only, so only protected getters are implemented, applying the backing field technique (Table 9). Properties confined, rb, and zb have protected backing fields confinedback, rbback, and zbback respectively. Other properties are dependent on these properties directly or indirectly.

The backing fields are set by the Grid constructor, which calls inherited Check methods to do the necessary checking on input rb, D, and confined. Note that vector zb is derived from input D. After object construction, the backing fields cannot be modified anymore. If the axi-symmetric grid is re-defined, a new Grid object must be created. Recall that the setgrid method of the Model object calls the Grid constructor. The Grid class has no other methods defined but the constructor and the property getters. A full description of class Grid is given in Appendix: members of class Grid.

Table 9. Properties for class `Grid`. It is indicated whether protected getters and setters are implemented or not, and whether a backing field is present or not. Properties without backing field depend on other properties. The get statement indicates on which property or backing field properties are dependent.

Property	Getter	Setter	Backing field	Get statement
`confined`	yes	no	`confinedback`	`true` or `false`
`rb`	yes	no	`rbback`	`rbback`
`r`	yes	no	-	`sqrt(rb(1:end-1).*rb(2:end))`
`dr`	yes	no	-	`diff(rb)`
`nr`	yes	no	-	`length(rb)-1`
`zb`	yes	no	`zbback`	`zbback`
`z`	yes	no	-	`(zb(1:end-1)+zb(2:end))/2`
`D`	yes	no	-	`-diff(zb)`
`nz`	yes	no	-	`length(zb)-1`
`hs`	yes	no	-	`pi*(rb(2:end).^2-rb(1:end-1).^2)`
`vol`	yes	no	-	`D*hs`

Class Parameters

The class diagram for class `Parameters` is shown in Figure 10, and properties are summarized in Table 3. Class `Parameters` has four read-only properties without backing fields: `confined`, `steady`, `nz`, and `nr`. These properties are set by the class constructor, which is called by the protected `setpar` method of the `Model` object, as will be explained below. They are copies of the properties with the same name in the `Grid` and the `TimeSteps` object, and cannot be modified.

All other properties are public and have a backing field and protected getters and setters (Table 10). These properties are set by the modeler or another application after object creation and they may be modified at any time. The property setters call inherited `Check` methods and use the four read-only properties to do the necessary checking on the input parameter arrays.

Table 10. Properties for class `Parameters`. The first four properties are read-only and have attribute `SetAccess = protected`. All other properties are public and have a backing field, a getter and a setter. The get statement for the getter simply gets the backing field contents.

Property	Getter	Setter	Backing field	Get statement
`confined`	no	no	-	-
`steady`	no	no	-	-
`nz`	no	no	-	-
`nr`	no	no	-	-
`inactive`	yes	yes	`inactiveback`	`inactiveback`
`constant`	yes	yes	`constantback`	`constantback`
`kr`	yes	yes	`krback`	`krback`
`cr`	yes	yes	`crback`	`crback`
`kz`	yes	yes	`kzback`	`kzback`
`cz`	yes	yes	`czback`	`czback`
`ss`	yes	yes	`ssback`	`ssback`
`sy`	yes	yes	`syback`	`syback`

Besides the constructor and property getters and setters, class `Parameters` also has a public method `isdefined`, which is called by the `Model` object to verify if all required properties are set. A full description of class `Parameters` is given in Appendix: members of class Parameters.

Class Stresses

The class diagram for class `Stresses` is shown in Figure 11, and properties are summarized in Table 4. Class `Stresses` has two read-only properties without backing fields: `nz` and `nr`. These properties are set by the class constructor, which is called by the protected `setstress` method of the `Model` object, as will be explained below. They are copies of the properties with the same name in the `Grid` object, and cannot be modified.

All other properties are public and have a backing field and protected getters and setters (Table 11). These properties are set by the modeler or another application after object creation and they may be modified at any time. The property setters call inherited `Check` methods and use the two read-only properties to do the necessary checking on the input parameter arrays.

Besides the constructor and property getters and setters, class `Stresses` also has a public method `isdefined`, which is called by the `Model` object to verify if any stress property is set. A full description of class `Stresses` is given in Appendix: members of class Stresses.

Table 11. Properties for class `Stresses`. The first two properties are read-only and have attribute `SetAccess = protected`. The two other properties are public and have a backing field, a getter and a setter. The get statement for the getter simply gets the backing field contents.

Property	Getter	Setter	Backing field	Get statement
`nz`	no	no	-	-
`nr`	no	no	-	-
`q`	yes	yes	`qback`	`qback`
`s0`	yes	yes	`s0back`	`s0back`

Class Solver

The class diagram for class `Solver` is shown in Figure 12, and properties are summarized in Table 5. All properties are public and have a backing field and protected getters and setters (Table 12).

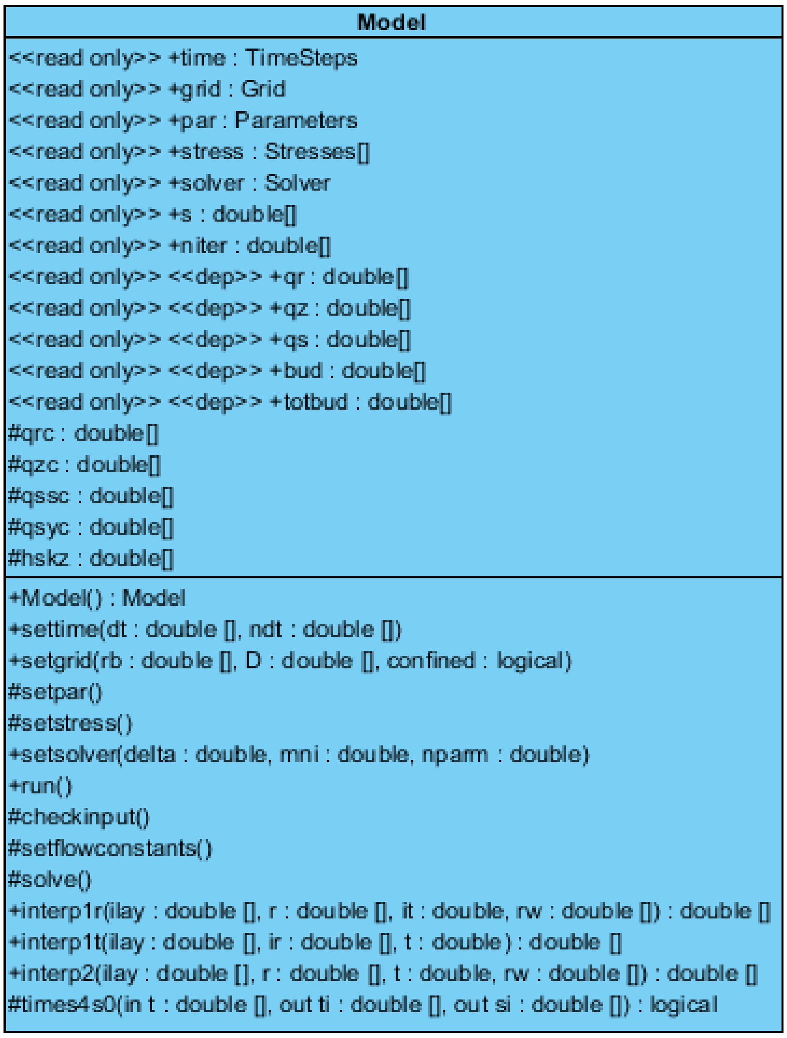
Table 12. Properties for class `Solver`. All properties have a backing field, a getter and a setter. The get statement for the getter simply gets the backing field contents. Some properties have a default value, which is assigned to the backing field.

Property	Getter	Setter	Backing field	Get statement	Default backing field
`delta`	yes	yes	`deltaback`	`deltaback`	-
`mni`	yes	yes	`mniback`	`mniback`	-
`nparm`	yes	yes	`nparmback`	`nparmback`	`[]`
`wseed`	yes	yes	`wseedback`	`wseedback`	`-1`
`accl`	yes	yes	`acclback`	`acclback`	`1`
`mex`	yes	yes	`mexback`	`mexback`	`true`

Recall that the `setsolver` method of the `Model` object calls the `Solver` constructor to create a `Solver` object. The constructor sets the backing fields for properties `delta` and `mni`, and if `nparm` is passed, the backing field for `nparm` is also set. The backing fields for `wseed`, `accl`, and `mex` are assigned a default value. After object creation, all `Solver` properties may be modified by the modeler. The `Solver` class has no other methods defined but the constructor and property getters and setters. A full description of class `Solver` is given in Appendix: members of class Solver.

Class Model

The class diagram for class `Model` is shown in Figure 7, and public read-only properties are summarized in Table 6. These properties do not have backing fields because the corresponding get methods are not likely to be overwritten. Properties `s` and `niter` have attribute `SetAccess = protected`. The same holds for properties `time`, `grid`, `par`, `stress`, and `solver`. The other public properties have attribute `Dependent = true`, and a get method is implemented for each of them, as is requested for dependent read-only properties.



```
Model
<<read only>> +time : TimeSteps
<<read only>> +grid : Grid
<<read only>> +par : Parameters
<<read only>> +stress : Stresses[]
<<read only>> +solver : Solver
<<read only>> +s : double[]
<<read only>> +niter : double[]
<<read only>> <<dep>> +qr : double[]
<<read only>> <<dep>> +qz : double[]
<<read only>> <<dep>> +qs : double[]
<<read only>> <<dep>> +bud : double[]
<<read only>> <<dep>> +totbud : double[]
#qrc : double[]
#qzc : double[]
#qssc : double[]
#qsyc : double[]
#hskz : double[]
+Model() : Model
+settime(dt : double [], ndt : double [])
+setgrid(rb : double [], D : double [], confined : logical)
#setpar()
#setstress()
+setsolver(delta : double, mni : double, nparm : double)
+run()
#checkinput()
#setflowconstants()
#solve()
+interp1r(ilay : double [], r : double [], it : double, rw : double []) : double []
+interp1t(ilay : double [], ir : double [], t : double) : double []
+interp2(ilay : double [], r : double [], t : double, rw : double []) : double []
#times4s0(in t : double [], out ti : double [], out si : double []) : logical
```

Figure 79. UML class diagram presenting public (+) and protected (#) class members in class `Model`*. Data types followed by* `[]` *indicate arrays, otherwise a scalar value is required. Input arguments preceded by keyword 'in' are input arguments, whereas keyword 'out' indicates output arguments. Keyword 'in' is omitted when the parameter list only contains input arguments. The return argument is always the first output argument.*

The idiom to define a dependent read-only property `x` with getter `get.x` is:

```
classdef classname < handle

    properties (Dependent)
        x
    end

    methods
        function x = get.x(obj)
            <statements>
            x = <expression>;
        end
    end

end
```

Each time property `x` is accessed, method `get.x` is invoked to calculate the value of `x`. All dependent properties for class `Model` are dependent directly or indirectly on property `s`, which stores the drawdown array. The expressions to calculate the dependent property arrays are given in the chapter discussing the Finite-difference approach. Using dependent properties has the advantage of saving memory.

Class `Model` also has protected members which are not visible to the modeler or other classes but subclasses. Figure 79 shows the entire class diagram for `Model` including protected class members. A full description of class `Model` is given in Appendix: members of class Model.

A set method is implemented for each of the five properties holding input class instances. The set methods for `time`, `grid`, and `solver` are public, the set methods for `par` and `stress` are protected. The idiom to define a public or protected set method for a read-only input property is:

```
classdef classname < handle

    properties (SetAccess = protected)
        x
        y
    end

    methods
        function setx(obj,varargin)
            <statements>
            obj.x = xclass(varargin{:});
            <statements>
        end
    end

    methods (Access = protected)
        function sety(obj)
            <statements>
            obj.y = yclass(<parameterlist>);
            <statements>
        end
    end

end
```

Property `x` has a public set method and holds an object of input class `xclass`, and property `y` has a protected set method and holds an object of input class `yclass`. The public set method `setx` accepts a parameter list, which is passed to the constructor `xclass` using the MATLAB statement `varargin` (see MATLAB help). The protected set method `sety` has no parameter list, whereas constructor `yclass` accepts a parameter list of input variables copied from other properties. Applied to the properties of class `Model`, we summarize as follows:

- `time`: public set method `settime(varargin)` calls `TimeSteps(varargin{:})`
- `grid`: public set method `setgrid(varargin)` calls `Grid(varargin{:})`
- `solver`: public set method `setsolver(varargin)` calls `Solver(varargin{:})`
- `par`: protected set method `setpar()` calls `Parameters(nz,nr,confined,steady)`
- `stress`: protected set method `setstress()` calls `Stresses(nz,nr)` for each stress period

The public set methods are called by the modeler or a MATLAB application that uses MAxSym. Methods `setpar` and `setstress` are invoked by method `settime` or `setgrid`, depending on the order in which the `TimeSteps` and the `Grid` objects are created, because the former methods can be invoked only if time steps and grid are defined. Figure 80 shows the different paths that can be followed to reach the `Model` state where all input properties are set. Figure 6, for instance, shows the modeler invokes `settime` before `setgrid`, hence, the latter calls `setpar` and `setstress` after calling the `Grid` class constructor. If `setgrid` is called first, `settime` calls `setpar` and `setstress` after creating a `TimeSteps` object.

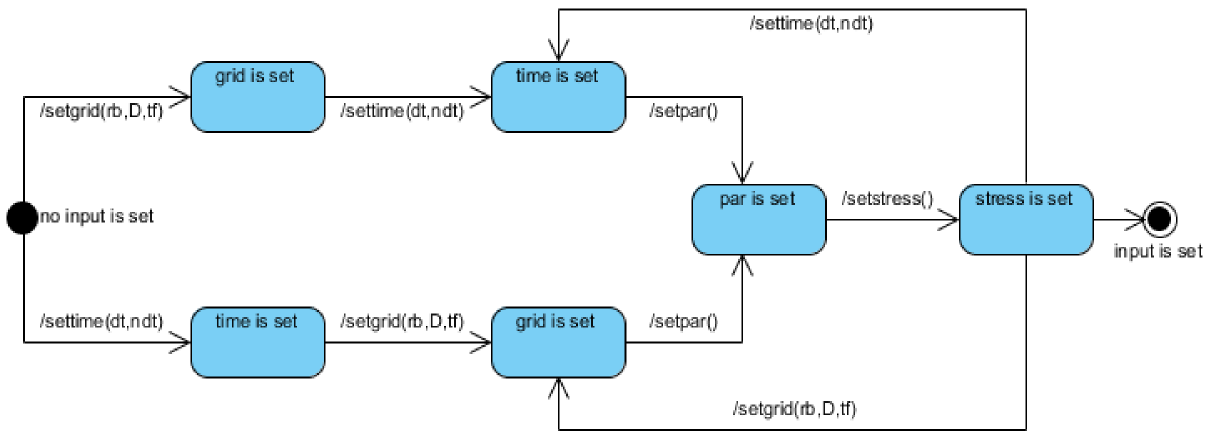


Figure 80. UML state diagram presenting the internal state of the `Model` object when the modeler is setting model input except for solver input. Method `setsolver` to set property `solver` may be invoked anytime by the modeler. The same holds for `settime` and `setgrid`, as is indicated by the diagram.

This means `settime` checks if property `grid` is set. If the latter is set, `setpar` and `setstress` are called, except for the following cases:

- Method `setpar` is not called if property `par` is already set and the simulation state (steady or unsteady) is not modified or `par.steady == time.steady`.
- Method `setstress` is not called if property `stress` is already set and the number of stress periods is not modified or `length(stress) == time.nper`.

It is clear these cases occur when time steps are re-defined. Similarly, `setgrid` checks if `time` is set, and `setpar` and `setstress` are called if `time` is not empty, except for the following cases:

- Method `setpar` is not called if property `time` is already set, and the grid size (number of layers and rings) and the flow type (confined or unconfined) are not modified or `par.nz == grid.nz & par.nr == grid.nr & par.confined == grid.confined`.
- Method `setstress` is not called if property `stress` is already set and the grid size (number of layers and rings) is not modified or `par.nz == grid.nz & par.nr == grid.nr`.

Protected methods `checkinput`, `setflowconstants`, and `solve` are invoked by the `run` method in this order (Figure 81):

1. Method `checkinput` is called to verify if all model input is given and consistent.
2. Method `setflowconstants` is called to calculate the flow equation constants which are stored in protected properties `qrc`, `qzc`, `qssc`, `qsyc`, and `hskz`.
3. Method `solve` is called, which, on its turn, calls the function `solveconfined` or `solveunconfined` to solve the system of equations. The flow equation constants are input arguments for the latter functions; the drawdown array is output and is assigned to property `s` by method `solve`. The next section discusses the solver functions in more detail.

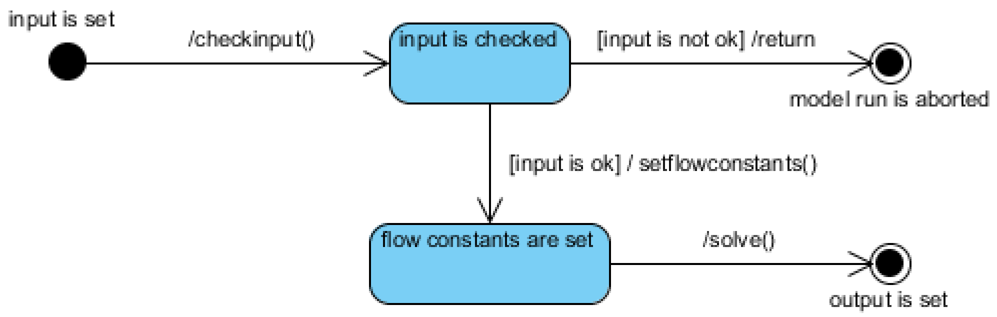


Figure 81. UML state diagram presenting the internal state of the `Model` object when method `run` is active.

The expressions to calculate property arrays `qrc`, `qzc`, `qssc`, and `qsyc` are given in section Flow equations. Property `hskz` is calculated as:

```
hskz = 2 * grid.hs * par.kz(1,:)
```

and it is used to correct the vertical resistance between rings in the two top layers in case of unconfined flow (see equation [30]). This correction is made after each iteration if `kz` is set by the modeler. If `cz` is given, no correction is made because vertical distance between nodal circles in the two layers is not known in this case.

Finally, protected method `times4s0` is called by interpolation methods `interp1t` and `interp2`. The former selects times for which interpolated drawdown must be corrected for instantaneous head changes (see equation [75]).

Solver functions

Method `solve` gets property `grid.confined` to verify whether the model is confined or unconfined. If the model is confined, function `solveconfined` is called, otherwise, function `solveunconfined` is called. Method `solve` passes the necessary model input to these functions:

- grid size;
- time steps and stress periods;
- flow equation constants;
- stresses;
- cell properties;
- solver parameters.

Based on the contents of properties `nparm` and `mex`, functions `solveconfined` and `solveunconfined` select the appropriate solver function to solve the system of equations. Figure 82 shows the decision tree. Solver functions prefixed by 'mex' are C routines that must be compiled to MEX funcions (see Installation of MAxSym). The other functions are MATLAB functions. A full syntax description of the solver functions is given in Appendix: solver functions. Command `help <solverfunctionname>` also displays the syntax description and requires the documentation for the MEX functions is written in an m-file with the same name as the c-file containing the source code.

Different input arguments are passed to each of these solver functions, and converting the model input into a valid parameter list is the task of functions `solveconfined` and `solveunconfined`. Some modifications to the input arrays are needed to enhance the efficiency of the solver algorithm. All matrices containing the flow equation constants must have n_z rows and n_r columns. If this is not the case, an extra column or row is added to the original matrix. These additional entries are zero and correspond to no-flow model boundaries. In case of a steady state simulation, all entries of matrix `qssc` (and `qsyc` if the model is unconfined) must be zero and `dt` must be assigned a non-zero value to avoid `NaN` results.

The `constant` and `inactive` matrices are converted to a single `ibound` matrix, which is defined as:

- `ibound(i,j)` > 0: ring `(i,j)` is active and has variable head;
- `ibound(i,j)` < 0: ring `(i,j)` is active and has constant head;
- `ibound(i,j)` = 0: ring `(i,j)` is inactive.

Stress matrices `q` and `s0` are concatenated to cells when passed to a MATLAB function and converted to a list with linear indices when passed to a MEX function. Note that zero-based indexing is used in C instead of one-based indexing. This explains the different definitions for input argument `nt` which contains the indices of the last time step in each stress period: in case of MATLAB functions, the first element of `nt` is 1, whereas the first element is 0 in case of MEX functions.

All MEX function input arguments are numeric yet 'data-type sensitive', which means the required numeric data type must be followed: all arguments are of type `double`, except for arguments `idq`, `ids0`, and `nt` containing indices, which are `int32` integers, and the `ibound` matrix, which is of type int8. Conversion from double, the default numeric data type, into int32 or int8 is done using MATLAB functions with the same name. Unlike the MATLAB functions, the MEX functions do not accept empty arrays and a value must be assigned to each parameter in the input list, even if the parameter is not relevant.

The output arguments s and niter are always double arrays. The calculation process for unconfined models may have been abrupted when the water table drawdown was larger than the thickness of the top layer. In this case, the niter parameter returned by the unconfined solver functions will contain zeros. Function solveunconfined checks niter for zero iteration time steps, sets the corresponding drawdowns NaN, and displays a message telling that the water table is below the top layer.

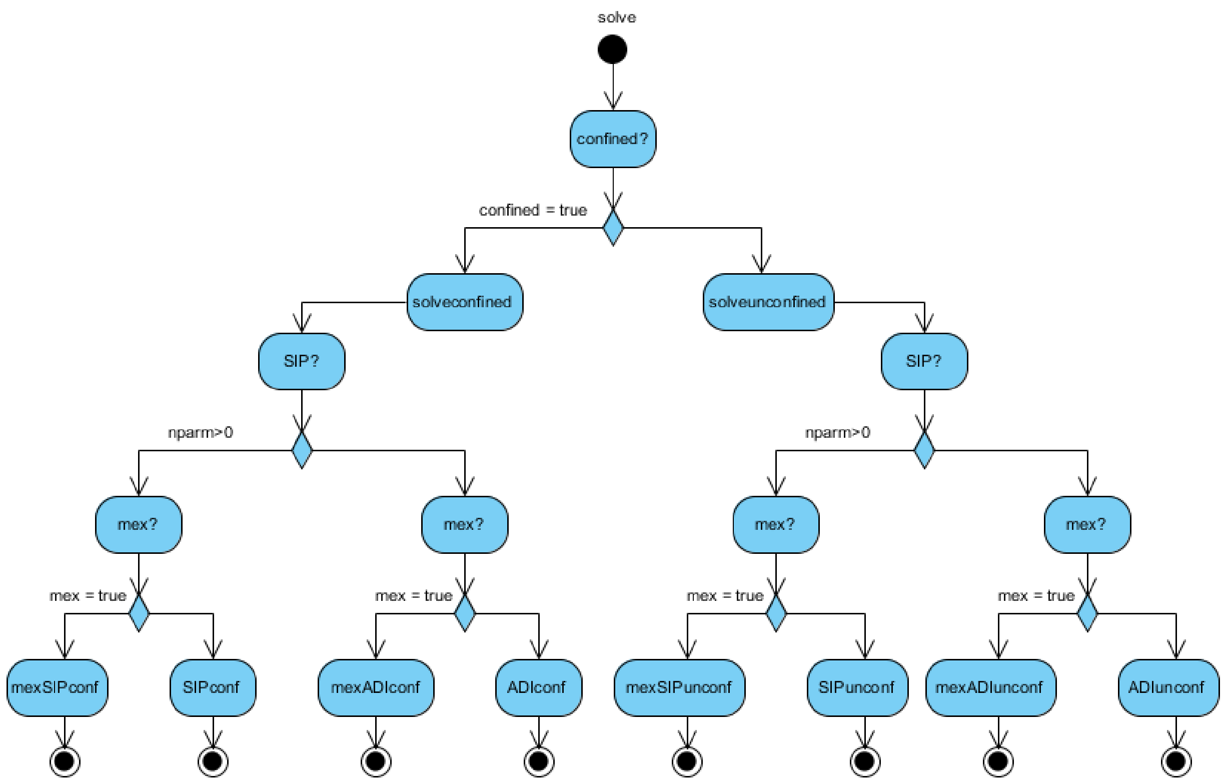


Figure 82. UML activity diagram showing the decision tree for calling one of the 8 solver functions. At the initial node, method solve is called by method run.

Summary

The design of MAxSym is object-oriented. Class Model acts as mediator and a Model object contains instances of classes TimeSteps, Grid, Parameters, Stresses, and Solver. These instances have protected backing fields that store the input data exposed by public properties. The technique of backing fields allows overriding of get and set methods of these properties. All MAxSym classes inherit from MATLAB class handle. The constructor of a handle class returns a reference to the object created, and when the handle object is assigned to multiple variables, MATLAB does not make a copy of the original object. MAxSym classes also inherit from static class Check, and use its inherited, protected methods to check input data.

The `Model` object also coordinates the calculation process and stores the calculated results. Dependent properties are used to save memory. Solving the system of finite-difference equations is done by one of the solver functions, which implement the ADI and SIP algorithms for confined and unconfined models. A MATLAB function as well as a MEX function is written for each algorithm. Invoking the appropriate solver function is done by functions `solveconfined` and `solveunconfined`, which are called by the `Model` object. The required flow equation constants are calculated by the latter.

The appendices contain a full syntax description of all MAxSym classes and functions.

Conclusion

This document presents MAxSym, a MATLAB tool for the simulation of two-dimensional, non-uniform, axi-symmetric groundwater flow in saturated, porous media. MAxSym solves the governing partial differential equation numerically by applying the finite-difference method. This requires discretization of space and time: an axi-symmetric grid consisting of layers and rings is constructed, and time steps are specified, which are grouped into stress periods. Grid boundaries are defined using constant-head and inactive rings, and hydraulic parameters are assigned to each ring in the grid. Initial and time-dependent boundary conditions are stated by specifying initial drawdown and discharge for each ring and for each stress period. If the system is linear and one-dimensional, it can be solved directly using the Thomas algorithm. Otherwise, the system must be solved iteratively using one of the two solvers provided with MAxSym: the iterative Alternating Direction Implicit method (ADI) or the Strongly Implicit Procedure (SIP). It is recommended to apply the SIP method to solve large models containing many sources and sinks.

The design of MAxSym is object-oriented, and it reflects the steps that are involved in building and running a MAxSym model. First, the model discretization is defined by specifying time steps, stress periods, and model grid. Then, grid boundaries, hydraulic parameters, and stresses are defined. Before running the model, the solver is chosen and its parameters are set. After the model is run, calculated drawdown can be interpolated bilinearly at arbitrary distances and arbitrary times. Setting up a MAxSym model is straightforward and only a dozen lines of code are required. The highly interactive MATLAB environment offers many tools to visualize and process the calculated results. If programs become large, however, it is good programming practice to organize code using object-oriented mechanisms such as inheritance or aggregation. Object-oriented design and implementation of MAxSym is explained in detail, enabling developers to easily extend MAxSym. Additionally, a full syntax description is given for each MAxSym class and function.

A large number of examples is given, which illustrate the many applications covered by MAxSym. Simple model cases are compared with existing analytical solutions, and it is concluded MAxSym approximates these solutions to great accuracy, while computation time remains small. However, the added value of MAxSym becomes more obvious in case of complex models, for which an analytical solution is much more involved or even not existing. In general, the examples simulate transient, axi-symmetric flow in laterally isotropic, multi-layer systems. Steady-state flow and one-dimensional flow in a single layer are considered special cases of the generic, time-dependent, two-dimensional case. Both confined and leaky systems may be defined, but also unconfined systems can be treaded by specifying a specific yield parameter. Care should be taken, however, since MAxSym only allows the upper layer to be phreatic. Flow in the unsaturated zone is ignored, but it is possible to define areal infiltration in the top layer. Using apparent distances, it is also possible to consider laterally anistropic systems.

The main advantage of the finite-difference solution is that it is able to simulate the variation of the saturated thickness in unconfined top layers. In addition, it is straightforward to include radial variation of hydraulic parameters, which is useful to simulate the effect of a finite-thickness skin. Head-specified or discharge-specified wells of small or large diameter can be implemented, which are fully or partially penetrating. Including wellbore storage is required for head-specified wells and large-diameter wells. Impervious well casing and packers within the well can also be taken into account. Simulating the effect of non-linear well losses is possible for pumping tests. The slug test models are limited to over-damped cases.

The many examples prove that MAxSym is a useful tool to analyze complex field situations and to simulate all kinds of advanced aquifer tests such as multiple pumping tests and multi-level slug tests. For the interpretation of aquifer tests, it is recommended to couple the MAxSym code to an optimization algorithm to make the parameter estimation process more advanced and more reliable. With MATLAB, this coupling is straightforward and an optimization toolbox is even available, which offers a large number of optimization algorithms. However, implementation of a standard non-linear regression method is easy, as is demonstrated in this document. The principle of superposition is also discussed and examples are given of simple, three-dimensional, steady and unsteady state models for multiple wells in multi-aquifer systems. The effect of areal infiltration may also be superimposed to the well solutions, and the method of images can be applied to include lateral no-flow and constant-head boundaries. Superposition models can be coupled to a linear programming algorithm to maximize well discharges subject to a set of drawdown constraints. Some algorithms are provided with the MATLAB optimization toolbox.

Appendix: members of class Model

Property `time`

Description
Stress periods and time steps

Attributes
`SetAccess = protected`

Set method
`settime`

Size and Class
1x1 `MAxSym.TimeSteps`

Property `grid`

Description
Axi-symmetric model grid

Attributes
`SetAccess = protected`

Set method
`setgrid`

Size and Class
1x1 `MAxSym.Grid`

Property `par`

Description
Hydraulic parameters

Attributes
`SetAccess = protected`

Set method
`setpar`

Size and Class
1x1 `MAxSym.Parameters`

Property `stress`

Description
Stresses

Attributes
`SetAccess = protected`

Set method
`setstress`

Size and Class
1 x n_{per} `MAxSym.Stresses`
with n_{per} the number of stress periods

Property `solver`

Description
Solver

Attributes
`SetAccess = protected`

Set method
`setsolver`

Size and Class
1x1 `MAxSym.Solver`

Property `s`

Description
Calculated drawdown *s* [L]

Attributes
`SetAccess = protected`

Size and Class
n_z x n_r x n_t `double`

Property `niter`

Description
Number of iterations for each time step

Attributes
`SetAccess = protected`

Size and Class
n_{dt} x *1* `double`

Property `qr`

Description
Radial discharge Q^r [L^3/T]

Attributes
`Dependent = true`

Get method
`get.qr`

Size and Class
n_z x (n_r+1) x n_t `double`

Property qz

Description
Vertical discharge Q^z [L^3/T]

Attributes
`Dependent = true`

Get method
`get.qz`

Size and Class
(n_z+1) x n_r x n_t `double`
`[]` if one layer only

Property qs

Description
Storage change Q^s [L^3/T]

Attributes
`Dependent = true`

Get method
`get.qs`

Size and Class
n_z x n_r x n_{dt} `double`
`[]` if steady state

Property bud

Description
Volumetric budget [L^3/T] at the end of each time step

Attributes
`Dependent = true`

Get method
`get.bud`

Size and Class
n_z x n_r x n_{dt} `double`

Property totbud

Description
Volumetric budget [L^3/T] for the entire model at the end of each time step
column 1 gives total volumetric budget for variable-head rings
column 2 gives total volumetric budget for constant-head rings

Attributes
`Dependent = true`

Get method
`get.totbud`

Size and Class
n_{dt} x 2 `double`

Property qrc

Description
Radial conductance Q^{rc} [L²/T]

Attributes
`Access = protected`

Size and Class
n_z x (n_r-1) `double`

Property qzc

Description
Radial conductance Q^{zc} [L²/T]

Attributes
`Access = protected`

Size and Class
(n_z-1) x n_r `double`
`[]` if one layer only

Property qssc

Description
Storage change term Q^{ssc} [L²]

Attributes
`Access = protected`

Size and Class
n_z x n_r `double`
`[]` if steady-state

Property qsyc

Description
Specific yield term Q^{syc} [L²]

Attributes
`Access = protected`

Size and Class
1 x n_r `double`
`[]` if steady-state or confined

Property hskz

Description
Vertical flow correction terms for the two top layers if top layer is phreatic
Vertical flow is not corrected in ring `j` if `hskz(j)<=0`

Attributes
`Access = protected`

Size and Class
1 x n_r `double`
`[]` if confined or one layer only

Method `settime`

Description
Sets time steps and stress periods (see property `time`)

Attributes
`Access = public`

Syntax
transient: `settime(obj,dt,ndt)`
steady state: `settime(obj,0)`

Input	**Description**
`obj`	`MAxSym.Model` object
`dt`	vector with time steps Δt [T]
`ndt`	vector with number of time steps for each stress period `length(dt)` must be equal to `sum(ndt)` only required if more than 1 stress period

Method `setgrid`

Description
Sets axi-symmetric grid (see property `grid`)

Attributes
`Access = public`

Syntax
`setgrid(obj,rb,D,confined)`

Input	**Description**
`obj`	`MAxSym.Model` object
`rb`	vector with radii r_b of ring boundaries [L] rings are numbered from inner to outer boundary
`D`	vector with layer thicknesses D [L] layers are numbered from top to bottom
`confined`	logical indicating whether the aquifer system is confined (`true`) or not (`false`)

Method `setpar`

Description
Sets hydraulic parameters (see property `par`)
Invoked by `settime` or `setgrid`

Attributes
`Access = protected`

Syntax
`setpar(obj)`

Input	**Description**
`obj`	`MAxSym.Model` object

Method `setstress`

Description
Sets stresses (see property `stress`)
Invoked by `settime` or `setgrid`

Attributes
`Access = protected`

Syntax
`setstress(obj)`

Input	**Description**
`obj`	`MAxSym.Model` object

Method `setsolver`

Description
Sets solver (see property `solver`)

Attributes
`Access = public`

Syntax
`setgrid(obj,delta,mni,nparm)`

Input	**Description**
`obj`	`MAxSym.Model` object
`delta`	criterion for convergence δ [L]
`mni`	maximum number of iterations for one time step iteration
`nparm`	number of SIP iteration parameters n_{parm} if `nparm` is given and greater than 0, SIP is used, otherwise, ADI is used

Method `run`

Description
Runs model

Attributes
`Access = public`

Syntax
`run(obj)`

Input	Description
`obj`	`MAxSym.Model` object

Method `checkinput`

Description
Checks model input
Invoked by `run`

Attributes
`Access = protected`

Syntax
`checkinput(obj)`

Input	Description
`obj`	`MAxSym.Model` object

Method `setflowconstants`

Description
Sets properties `qrc`, `qzc`, `qssc`, `qsyc`, and `hskz`
Invoked by `run`

Attributes
`Access = protected`

Syntax
`setflowconstants(obj)`

Input	Description
`obj`	`MAxSym.Model` object

Method `solve`

Description
Calls appropriate solver function: `solveconfined` or `solveunconfined`
Invoked by `run`

Attributes
`Access = protected`

Syntax
`checkinput(obj)`

Input	Description
`obj`	`MAxSym.Model` object

Method `interp1r`

Description
Interpolates drawdown in dimension of *log(r)*

Attributes
`Access = public`

Syntax
transient: `s = interp1r(obj,ilay,r,it,rw)`
steady state: `s = interp1r(obj,ilay,r,rw)`

Input	**Description**
`obj`	`MAxSym.Model` object
`ilay`	vector with indices of considered layers if `ilay` is empty, all layers are considered
`r`	vector with radial distances at which drawdown *s* is interpolated
`it`	vector with indices of considered simulation times (only if transient) if `it` is empty, all simulation times are considered
`rw`	vector of `length(ilay)` with well radius for each layer drawdown *s* at distances smaller than or equal to `rw` is not interpolated, but equal to the drawdown in the first ring if `rw` is empty or not given, the well radius coincides with the first nodal circle if `rw` is a scalar, all layers have the same well radius

Output	**Description**
`s`	interpolated drawdown matrix transient: `size(s) = [length(ilay),length(r),length(it)]` steady state: `size(s) = [length(ilay),length(r)]`

Method `interp1t`

Description
Interpolates drawdown in dimension of *log(t)*

Attributes
`Access = public`

Syntax
`s = interp1t(obj,ilay,ir,t)`

Input	**Description**
`obj`	`MAxSym.Model` object
`ilay`	vector with indices of considered layers if `ilay` is empty, all layers are considered
`ir`	vector with indices of considered rings if `ir` is empty, all rings are considered
`t`	vector with times at which drawdown *s* is interpolated drawdown *s* at times smaller than *t(2)* is not interpolated, but equal to initial drawdown s_0

Output	**Description**
`s`	interpolated drawdown matrix of size `[length(ilay),length(ir),length(t)]`

Method `interp2`

Description
Interpolates drawdown in dimension of *log(r)* and dimension of *log(t)*

Attributes
`Access = public`

Syntax
`s = interp1t(obj,ilay,r,t,rw)`

Input	**Description**
`obj`	`MAxSym.Model` object
`ilay`	vector with indices of considered layers if `ilay` is empty, all layers are considered
`r`	vector with radial distances at which drawdown *s* is interpolated
`t`	vector with times at which drawdown *s* is interpolated drawdown *s* at times smaller than *t(2)* is not interpolated, but equal to initial drawdown s_0
`rw`	vector of `length(ilay)` with well radius for each layer drawdown *s* at distances smaller than or equal to `rw` is not interpolated, but equal to the drawdown in the first ring if `rw` is empty or not given, the well radius coincides with the first nodal circle if `rw` is a scalar, all layers have the same well radius

Output	**Description**
`s`	interpolated drawdown matrix of size `[length(ilay),length(r),length(t)]`

Method `times4s0`

Description
Selects times for which interpolated drawdown must be corrected for initial drawdown s_0

Attributes
`Access = protected`

Syntax
`[b,ti,si] = times4s0(obj,t)`

Input	**Description**
`obj`	`MAxSym.Model` object
`t`	vector with times at which drawdown *s* is interpolated

Output	**Description**
`b`	indicates which times `t` are inside time steps ending with initial drawdown
`ti`	vector with relevant simulation times
`si`	n_z x n_r x `length(ti)` array with relevant drawdowns corrected for instantaneous head changes

Appendix: members of class TimeSteps

Property `steady`

Description
Indicates whether the simulation is steady state (`true`) or transient (`false`)

Attributes
`Dependent = true`

Get method
`get.steady`
`getsteady`

Size and Class
1 x 1 `logical`

Property `t`

Description
Simulation times *t* [T] – zero if steady state

Attributes
`Dependent = true`

Backing field
`tback`

Get method
`get.t`
`gett`

Size and Class
n_t x 1 `double`

Property `dt`

Description
Time steps *Δt* [T] – empty if steady state

Attributes
`Dependent = true`

Get method
`get.dt`
`getdt`

Size and Class
n_{dt} x 1 `double`

Property `ndt`

Description
Number of time steps in each stress period – empty if steady state

Attributes
`Dependent = true`

Backing field
`ndtback`

Get method
`get.ndt`
`getndt`

Size and Class
n_{per} x 1 `double`, with n_{per} the number of stress periods

Property `tper`

Description
Stress period times [T] – zero if steady state

Attributes
`Dependent = true`

Get method
`get.tper`
`gettper`

Size and Class
(n_{per}+1) x 1 `double`, with n_{per} the number of stress periods

Property `dtper`

Description
Stress period lengths [T] – empty if steady state

Attributes
`Dependent = true`

Get method
`get.dtper`
`getdtper`

Size and Class
n_{per} x 1 `double`, with n_{per} the number of stress periods

Property `nper`

Description
number of stress periods n_{per} – one if steady state

Attributes
`Dependent = true`

Get method
`get.nper`
`getnper`

Size and Class
1 x 1 `double`

Method `TimeSteps`

Description
Creates `TimeSteps` object

Attributes
`Access = public`

Syntax
transient: `obj = MAxSym.TimeSteps(dt,ndt)`
steady state: `obj = MAxSym.TimeSteps(0)`

Input	**Description**
`dt`	vector with time steps Δt [T]
`ndt`	vector with number of time steps for each stress period `length(dt)` must be equal to `sum(ndt)` only required if more than 1 stress period

Output	**Description**
`obj`	`TimeSteps` object

Appendix: members of class Grid

Property `confined`

Description
Indicates whether the top layer is confined (`true`) or unconfined (`false`)

Attributes
`Dependent = true`

Backing field
`confinedback`

Get method
`get.confined`
`getconfined`

Size and Class
1 x 1 `logical`

Property `rb`

Description
Radii r_b of ring boundaries [L] – from inner to outer model boundary

Attributes
`Dependent = true`

Backing field
`rbback`

Get method
`get.rb`
`getrb`

Size and Class
1 x (n_r+1) `double`

Property `r`

Description
Radii r of nodal circles [L] – from inner to outer model boundary

Attributes
`Dependent = true`

Get method
`get.r`
`getr`

Size and Class
1 x n_r `double`

Property `dr`

Description
Ring widths [L] – from inner to outer model boundary

Attributes
`Dependent = true`

Get method
`get.dr`
`getdr`

Size and Class
1 x n_r `double`

Property `nr`

Description
Number of rings n_r

Attributes
`Dependent = true`

Get method
`get.nr`
`getnr`

Size and Class
1 x 1 `double`

Property `zb`

Description
Levels z_b of layer boundaries [L] – from top to bottom boundary

Attributes
`Dependent = true`

Backing field
`zbback`

Get method
`get.zb`
`getzb`

Size and Class
(n_z+1) x 1 `double`

Property `z`

Description
Levels z of layer middles [L] – from top to bottom boundary

Attributes
`Dependent = true`

Get method
`get.z`
`getz`

Size and Class
n_z x 1 `double`

Property `D`

Description
Layer thicknesses *D* [L] – from top to bottom boundary

Attributes
`Dependent = true`

Get method
`get.D`
`getD`

Size and Class
n_z x 1 `double`

Property `nz`

Description
Number of layers n_z

Attributes
`Dependent = true`

Get method
`get.nz`
`getnz`

Size and Class
1 x 1 `double`

Property `hs`

Description
Horizontal surface area [L^2] of rings

Attributes
`Dependent = true`

Get method
`get.hs`
`geths`

Size and Class
1 x n_r `double`

Property `vol`

Description
Volume [L^3] of rings

Attributes
`Dependent = true`

Get method
`get.vol`
`getvol`

Size and Class
n_z x n_r `double`

Method `Grid`

Description
Creates `Grid` object

Attributes
`Access = public`

Syntax
`obj = Grid(rb,D,confined)`

Input	**Description**
`rb`	vector with radii r_b of ring boundaries [L] rings are numbered from inner to outer boundary
`D`	vector with layer thicknesses D [L] layers are numbered from top to bottom
`confined`	logical indicating whether the aquifer system is confined (`true`) or not (`false`)

Output	**Description**
`obj`	`Grid` object

Appendix: members of class Parameters

Property `confined`

Description
Indicates whether the top layer is confined (`true`) or unconfined (`false`)

Attributes
`SetAccess = protected`

Size and Class
1 x 1 `logical`

Property `steady`

Description
Indicates whether the simulation is steady state (`true`) or transient (`false`)

Attributes
`SetAccess = protected`

Size and Class
1 x 1 `logical`

Property `nz`

Description
Number of layers n_z

Attributes
`SetAccess = protected`

Size and Class
1 x 1 `double`

Property `nr`

Description
Number of rings n_r

Attributes
`SetAccess = protected`

Size and Class
1 x 1 `double`

Property `inactive`

Description
Inactive rings – all rings are active by default

Attributes
`Dependent = true`

Backing field
`inactiveback`

Get method
`get.inactive`
`getinactive`

Set method
`set.inactive`
`setinactive`

Size and Class
n_z x n_r `logical`

Property `constant`

Description
Constant-head rings – all rings have variable head by default

Attributes
`Dependent = true`

Backing field
`constantback`

Get method
`get.constant`
`getconstant`

Set method
`set.constant`
`setconstant`

Size and Class
n_z x n_r `logical`

Property `kr`

Description
Radial component of hydraulic conductivity K^r [L/T]
Required if some elements of `cr` are zero
`NaN` values are ignored

Attributes
`Dependent = true`

Backing field
`krback`

Get method
`get.kr`
`getkr`

Set method
`set.kr`
`setkr`

Size and Class
n_z x n_r `double`

Property `cr`

Description
Radial resistance C^r [T] between rings
Required if `kr` is not set
`NaN` values and values ≤ 0 are ignored

Attributes
`Dependent = true`

Backing field
`crback`

Get method
`get.cr`
`getcr`

Set method
`set.cr`
`setcr`

Size and Class
n_z x (n_r-1) `double`

Property `kz`

Description
Vertical component of hydraulic conductivity K^z [L/T]
Required if $n_z > 1$ and some elements of `cz` are zero
`NaN` values are ignored

Attributes
`Dependent = true`

Backing field
`kzback`

Get method
`get.kz`
`getkz`

Set method
`set.kz`
`setkz`

Size and Class
n_z x n_r `double`

Property `cz`

Description
Radial resistance C^z [T] between rings

Required if `kz` is not set and $n_z > 1$
`NaN` values and values ≤ 0 are ignored

Attributes
`Dependent = true`

Backing field
`czback`

Get method
`get.cz`
`getcz`

Set method
`set.cz`
`setcz`

Size and Class
$(n_z - 1) \times n_r$ `double`

Property ss

Description
Specific elastic storage coefficient S^s [L^{-1}]
Required in case of transient simulation
`NaN` values are ignored

Attributes
`Dependent = true`

Backing field
`ssback`

Get method
`get.ss`
`gets`

Set method
`set.ss`
`sets`

Size and Class
$n_z \times n_r$ `double`

Property sy

Description
Specific yield S^y [-]
Required if model is transient and unconfined
`NaN` values are ignored

Attributes
`Dependent = true`

Backing field
`syback`

Get method
`get.sy`
`getsy`

Set method
`set.sy`
`setsy`

Size and Class
1 x n_r `double`

Method `Parameters`

Description
Creates `Parameters` object

Attributes
`Access = public`

Syntax
`obj = Parameters(nz,nr,confined,steady)`

Input	**Description**
`nz`	number of layers n_z
`nr`	number of rings n_r
`confined`	logical indicating whether the top layer is confined (`true`) or unconfined (`false`)
`steady`	logical Indicating whether the simulation is steady state (`true`) or transient (`false`)

Output	**Description**
`obj`	`Parameters` object

Method `isdefined`

Description
Checks if all required parameters are defined

Attributes
`Access = public`

Syntax
`tf = isdefined(obj)`

Input	**Description**
`obj`	`Parameters` object

Output	**Description**
`tf`	logical indicating whether all parameters are defined (`true`) or not (`false`)

Appendix: members of class Stresses

Property `nz`

Description
Number of layers n_z

Attributes
`SetAccess = protected`

Size and Class
1 x 1 `double`

Property `nr`

Description
Number of rings n_r

Attributes
`SetAccess = protected`

Size and Class
1 x 1 `double`

Property `q`

Description
Ring discharges Q [L^3/T]
A positive entry means that water is extracted from the corresponding ring
All discharges are zero by default

Attributes
`Dependent = true`

Backing field
`qback`

Get method
`get.q`
`getq`

Set method
`set.q`
`setq`

Size and Class
n_z x n_r `double`

Property `s0`

Description
Initial drawdown s_0 [L]
A positive entry means that an instantaneous head rise is realized in the corresponding ring
All initial drawdowns are zero by default

Attributes
`Dependent = true`

Backing field
`s0back`

Get method
`get.s0`
`gets0`

Set method
`set.s0`
`sets0`

Size and Class
n_z x n_r `double`

Method `Stresses`

Description
Creates `Stresses` object

Attributes
`Access = public`

Syntax
`obj = Stresses(nz,nr)`

Input	**Description**
`nz`	number of layers n_z
`nr`	number of rings n_r

Output	**Description**
`obj`	`Stresses` object

Method `isdefined`

Description
Checks if any stress is defined, i.e. any non-zero entry in matrix `q` or `s0`

Attributes
`Access = public`

Syntax
`tf = isdefined(obj)`

Input	**Description**
`obj`	`Stresses` object

Output	**Description**
`tf`	logical indicating whether any stress is defined (`true`) or not (`false`)

Appendix: members of class Solver

Property `delta`

Description
Criterion for convergence δ [L]

Attributes
`Dependent = true`

Backing field
`deltaback`

Get method
`get.delta`
`getdelta`

Set method
`set.delta`
`setdelta`

Size and Class
1 x 1 `double`

Property `mni`

Description
Maximum number of iterations during one time step

Attributes
`Dependent = true`

Backing field
`mniback`

Get method
`get.mni`
`getmni`

Set method
`set.mni`
`setmni`

Size and Class
1 x 1 `double`

Property `nparm`

Description
Number of SIP iteration parameters n_{parm}
If greater than zero, SIP is used, otherwise, ADI is used

Attributes
`Dependent = true`

Backing field
`nparmback`

Get method
`get.nparm`
`getnparm`

Set method
`set.nparm`
`setnparm`

Size and Class
1 x 1 `double`

Property `wseed`

Description
Iteration parameter seed σ for calculaton of SIP relaxation parameter ω
If `wseed` is empty or smaller than zero, it is derived from the conductance matrices (default)

Attributes
`Dependent = true`

Backing field
`wseedback`

Get method
`get.wseed`
`getwseed`

Set method
`set.wseed`
`setwseed`

Size and Class
1 x 1 `double`

Property `accl`

Description
SIP accelerator α, which is a value between 0 and 1
If `accl` is empty, an accelerator of 1 is used (default)

Attributes
`Dependent = true`

Backing field
`acclback`

Get method
`get.accl`
`getaccl`

Set method
`set.accl`
`setaccl`

Size and Class
1 x 1 `double`

Property `mex`

Description
Indicates whether MEX function (`true`, default) or MATLAB function (`false`) is called

Attributes
`Dependent = true`

Backing field
`mexback`

Get method
`get.mex`
`getmex`

Set method
`set.mex`
`setmex`

Size and Class
1 x 1 `logical`

Method `Solver`

Description
Creates `Solver` object

Attributes
`Access = public`

Syntax
`obj = Solver(delta,mni,nparm)`

Input	**Description**
`delta`	criterion for convergence δ [L]
`mni`	maximum number of iterations for one time step iteration
`nparm`	number of SIP iteration parameters n_{parm}
	if `nparm` is given and greater than 0, SIP is used, otherwise, ADI is used

Output	**Description**
`obj`	`MAxSym.Solver` object

Appendix: solver functions

Function `solveconfined`

Description
Solves finite-difference equations for 2D, axi-symmetric, confined model using ADI or SIP

Syntax
```
[s,niter] = MAxSym.solveconfined(nz,nr,nper,ndt,dt,...
                                 qrc,qzc,qssc,...
                                 q,s0,inactive,constant,...
                                 delta,mni,nparm,wseed,accl,mx)
```

Input	**Description**
`nz`	number of layers n_z
`nr`	number of rings n_r
`nper`	number of stress periods
`ndt`	number of time steps in each stress period `length(ndt) == nper` and `sum(ndt)` = n_{dt} if the simulation is steady state, `ndt` is empty or smaller than 1
`dt`	vector with n_{dt} time steps Δt [T] `length(dt) == sum(ndt)` if the simulation is steady state, `dt` is ignored
`qrc`	n_z x (n_r-1) matrix with radial conductances Q^{rc} [L^2/T]
`qzc`	(n_z -1) x n_r matrix with vertical conductances Q^{zc} [L^2/T] if the model has one layer only (`nz==1`), `qzc` is ignored
`qssc`	n_z x n_r matrix with storage change terms Q^{ssc} [L^2] if the simulation is steady state, `qssc` is ignored

`q`	`cell` where `q{i}` is n_z x n_r matrix with ring discharges Q [L^3/T] for stress period `i`
	`length(q) == nper`
	a positive discharge value means that water is withdrawn
	`q` is empty if all discharges are zero throughout the simulation
	`q{i}` is empty if all discharges are zero for stress period `i`

`s0`	`cell` where `s0{i}` is n_z x n_r matrix with initial drawdowns s_0 [L] at the start of stress period `i` (starting heads in case of a steady state simulation)
	`length(s0) == nper`
	a positive initial drawdown value means that there is an instantaneous head rise
	`s0` is empty if all initial drawdowns are zero throughout the simulation
	`s0{i}` is empty if all initial drawdowns are zero for stress period `i`

`inactive`	n_z x n_r logical matrix indicating inactive rings if `inactive` is empty, there are no inactive rings
`constant`	n_z x n_r logical matrix indicating constant-head rings if `constant` is empty, there are no constant-head rings
`delta`	criterion for convergence δ [L]
`mni`	maximum number of iterations for one time step iteration
`nparm`	number of SIP iteration parameters n_{parm} if `nparm` is empty or smaller than 1, ADI is used
`wseed`	iteration parameter seed σ for calculaton of SIP relaxation parameter ω if `wseed` is empty or smaller than zero, it is derived from the conductance matrices
`accl`	SIP accelerator α, which is a value between 0 and 1 if `accl` is empty, an accelerator of 1 is used
`mx`	logical indicating whether MEX function (`true`) or MATLAB function (`false`) is called

Output	Description
`s`	n_z x n_r x n_t matrix with drawdowns *s* [L] a positive drawdown value means that head has risen since the start of the simulation recall that `s(:,:,1) = s0{1}` if the simulation is steady state, `s` is n_z x n_r matrix
`niter`	vector with number of iterations performed during each time step `length(niter) == sum(ndt)` if the simulation is steady state, `niter` is scalar

Function `solveunconfined`

Description
Solves finite-difference equations for 2D, axi-symmetric, unconfined model using ADI or SIP

Syntax
```
[s,niter] = MAxSym.solveunconfined(nz,nr,nper,ndt,dt,...
                                   qrc,qzc,qssc,qsyc,D1,hskz,...
                                   q,s0,inactive,constant,...
                                   delta,mni,nparm,wseed,accl,mx)
```

Input	Description
`nz`	number of layers n_z
`nr`	number of rings n_r
`nper`	number of stress periods
`ndt`	number of time steps in each stress period `length(ndt) == nper` and `sum(ndt)` = n_{dt} if the simulation is steady state, `ndt` is empty or smaller than 1
`dt`	vector with n_{dt} time steps Δt [T] `length(dt) == sum(ndt)` if the simulation is steady state, `dt` is ignored
`qrc`	n_z x (n_r-1) matrix with radial conductances Q^{rc} [L²/T]
`qzc`	(n_z -1) x n_r matrix with vertical conductances Q^{zc} [L²/T] if the model has one layer only (`nz==1`), `qzc` is ignored
`qssc`	n_z x n_r matrix with storage change terms Q^{ssc} [L²] if the simulation is steady state, `qssc` is ignored
`qsyc`	`1` x n_r vector with specific yield terms Q^{syc} [L²] if the simulation is steady state, the `qsyc` terms are ignored
`D1`	saturated thickness *D(i)* [L] of upper phreatic layer before simulation starts
`hskz`	`1` x n_r vector with vertical flow correction terms for the top layers vertical flow is not corrected in ring `j` if `hskz(j)<=0` if the model has one layer only (`nz==1`), `hskz` is ignored

`q`	`cell` where `q{i}` is n_z x n_r matrix with ring discharges *Q* [L³/T] for stress period `i`
	`length(q) == nper`
	a positive discharge value means that water is withdrawn
	`q` is empty if all discharges are zero throughout the simulation
	`q{i}` is empty if all discharges are zero for stress period `i`

`s0`	`cell` where `s0{i}` is n_z x n_r matrix with initial drawdowns s_0 [L] at the start of stress period `i` (starting heads in case of a steady state simulation)
	`length(s0) == nper`
	a positive initial drawdown value means that there is an instantaneous head rise
	`s0` is empty if all initial drawdowns are zero throughout the simulation
	`s0{i}` is empty if all initial drawdowns are zero for stress period `i`

`inactive`	n_z x n_r logical matrix indicating inactive rings if `inactive` is empty, there are no inactive rings
`constant`	n_z x n_r logical matrix indicating constant-head rings if `constant` is empty, there are no constant-head rings
`delta`	criterion for convergence δ [L]
`mni`	maximum number of iterations for one time step iteration
`nparm`	number of SIP iteration parameters n_{parm} if `nparm` is empty or smaller than 1, ADI is used
`wseed`	iteration parameter seed σ for calculaton of SIP relaxation parameter ω if `wseed` is empty or smaller than zero, it is derived from the conductance matrices
`accl`	SIP accelerator α, which is a value between 0 and 1 if `accl` is empty, an accelerator of 1 is used
`mx`	logical indicating whether MEX function (`true`) or MATLAB function (`false`) is called

Output	**Description**
`s`	n_z x n_r x n_t matrix with drawdowns s [L] a positive drawdown value means that head has risen since the start of the simulation recall that `s(:,:,1) = s0{1}` if the simulation is steady state, `s` is n_z x n_r matrix
`niter`	vector with number of iterations performed during each time step `length(niter) == sum(ndt)` if the simulation is steady state, `niter` is scalar

Function `mexADIconf`

Description

ANSI C ADI routine to solve finite-difference equations for 2D, axi-symmetric, confined model

Syntax

```
[s,niter] = MAxSym.mexADIconf(nz,nr,nper,nt,dt,...
                              qrc,qzc,qrzc,qssc,...
                              nq,idq,q,ns0,ids0,s0,ibound,...
                              delta,mni)
```

Input	**Description**
`nz`	number of layers n_z (`double`)
`nr`	number of rings n_r (`double`)
`nper`	number of stress periods (`double`)
`nt`	`int32` vector with zero based time step indices indicating the end of each stress period `length(nt) == nper+1` `nt(1)` = 0 indicates the initial drawdown condition at the start of the simulation the number of time steps for each stress period is `diff(nt)` if the simulation is steady state, `nt = [0 1]`
`dt`	`double` vector with time steps Δt [T] `length(dt) == nt(end)` if the simulation is steady state, `dt` is ignored, but may not be empty or zero
`qrc`	`nz` x `nr` `double` matrix with radial conductances Q^{rc} [L^2/T] `qrc(:,1) = 0` corresponds to the inner model boundary
`qzc`	`nz` x `nr` `double` matrix with vertical conductances Q^{zc} [L^2/T] `qzc(1,:) = 0` corresponds to the upper model boundary if the model has one layer (`nz==1`), `qzc` is ignored, but may not be empty

`qrzc`	`nz` x `nr` `double` matrix with sum of conductances [L^2/T] i.e. sum of upper and lower vertical conductances and inner and outer conductances
`qssc`	`nz` x `nr` `double` matrix with storage change terms Q^{ssc} [L^2] if the simulation is steady state, all `qssc` entries must be zero
`nq`	total number of non-zero discharges (`double`)
`idq`	`nq` x 2 `int32` matrix with zero based stress period indices and grid indices of discharged rings or `idq = [iper(:),igrid(:)]` where `iper` is stress period index and `igrid` is linear grid index equal to `nr*iring+ilay` with `iring` and `ilay` the ring and layer indices respectively if `nq == 0`, `idq` is ignored, but may not be empty
`q`	`double` vector of length `nq` with discharges Q [L^3/T] corresponding to `idq` a positive discharge value means that water is withdrawn if `nq == 0`, `q` is ignored, but may not be empty
`ns0`	total number of non-zero initial drawdowns (`double`)
`ids0`	`ns0` x 2 `int32` matrix with zero based stress period indices and grid indices of rings with non-zero instantaneous head change or `ids0 = [iper(:),igrid(:)]` where `iper` is stress period index and `igrid` is linear grid index equal to `nr*iring+ilay` with `iring` and `ilay` the ring and layer indices respectively if `ns0 == 0`, `ids0` is ignored, but may not be empty
`s0`	`double` vector of length `ns0` with initial drawdowns s_0 [L] corresponding to `ids0` a positive initial drawdown value means that there is an instantaneous head rise if `ns0 == 0`, `s0` is ignored, but may not be empty
`ibound`	`nz` x `nr` `int8` matrix indicating the head property for each ring: `ibound` > 0 indicates an active, variable-head ring `ibound` = 0 indicates an inactive, impervious ring `ibound` < 0 indicates an active, constant-head ring
`delta`	criterion for convergence δ [L] (`double`)
`mni`	maximum number of iterations for one time step iteration (`double`)

Output	**Description**
`s`	`nz` x `nr` x (`nt(end)`+1) `double` matrix with drawdowns s [L] a positive drawdown value means that head has risen since the start of the simulation recall that `s(:,:,1) = s0{1}` if the simulation is steady state, `s` is `nz` x `nr` matrix
`niter`	`double` vector with number of iterations performed during each time step `length(niter) == nt(end)` if the simulation is steady state, `niter` is scalar

Function `ADIconf`

Description

MATLAB ADI routine to solve finite-difference equations for 2D, axi-symmetric, confined model

Syntax

```
[s,niter] = MAxSym.ADIconf(nz,nr,nper,nt,dt,...
                           qrc,qzc,qrzc,qssc,...
                           q,s0,ibound,...
                           delta,mni)
```

Input	Description
`nz`	number of layers n_z
`nr`	number of rings n_r
`nper`	number of stress periods
`nt`	vector with time step indices indicating the end of each stress period
	`length(nt) == nper+1`
	`nt(1)` = 1 indicates the initial drawdown condition at the start of the simulation
	the number of time steps for each stress period is `diff(nt)`
	if the simulation is steady state, `nt = [1 2]`
`dt`	vector with time steps *Δt* [T]
	`length(dt) == nt(end)-1`
	if the simulation is steady state, `dt` is ignored, but may not be empty or zero
`qrc`	`nz` x `nr` matrix with radial conductances Q^{rc} [L^2/T]
	`qrc(:,1) = 0` corresponds to the inner model boundary
`qzc`	`nz` x `nr` matrix with vertical conductances Q^{zc} [L^2/T]
	`qzc(1,:) = 0` corresponds to the upper model boundary
	if the model has one layer (`nz==1`), `qzc` is ignored
`qrzc`	`nz` x `nr` matrix with sum of conductances [L^2/T]
	i.e. sum of upper and lower vertical conductances and inner and outer conductances
`qssc`	`nz` x `nr` matrix with storage change terms Q^{ssc} [L^2]
	if the simulation is steady state, all `qssc` entries must be zero
`q`	`cell` where `q{i}` is `nz` x `nr` matrix with ring discharges *Q* [L^3/T] for stress period `i`
	`length(q) == nper`
	a positive discharge value means that water is withdrawn
	`q` is empty if all discharges are zero throughout the simulation
	`q{i}` is empty if all discharges are zero for stress period `i`
`s0`	`cell` where `s0{i}` is `nz` x `nr` matrix with initial drawdowns s_0 [L] at the start of stress period `i` (starting heads in case of a steady state simulation)
	`length(s0) == nper`
	a positive initial drawdown value means that there is an instantaneous head rise
	`s0` is empty if all initial drawdowns are zero throughout the simulation
	`s0{i}` is empty if all initial drawdowns are zero for stress period `i`
`ibound`	`nz` x `nr` matrix indicating the head property for each ring:
	`ibound` > 0 indicates an active, variable-head ring
	`ibound` = 0 indicates an inactive, impervious ring
	`ibound` < 0 indicates an active, constant-head ring
`delta`	criterion for convergence *δ* [L]
`mni`	maximum number of iterations for one time step iteration

Output	Description
`s`	`nz` x `nr` x `nt(end)` matrix with drawdowns *s* [L]
	a positive drawdown value means that head has risen since the start of the simulation
	recall that `s(:,:,1) = s0{1}`
	if the simulation is steady state, `s` is `nz` x `nr` matrix
`niter`	vector with number of iterations performed during each time step
	`length(niter) == nt(end)-1`
	if the simulation is steady state, `niter` is scalar

Function mexSIPconf

Description

ANSI C SIP routine to solve finite-difference equations for 2D, axi-symmetric, confined model

Syntax

```
[s,niter] = MAxSym.mexSIPconf(nz,nr,nper,nt,dt,...
                              qrc,qzc,qrzc,qssc,...
                              nq,idq,q,ns0,ids0,s0,ibound,...
                              delta,mni,nparm,wseed,accl)
```

Input	**Description**
`nz`	number of layers n_z (`double`)
`nr`	number of rings n_r (`double`)
`nper`	number of stress periods (`double`)
`nt`	`int32` vector with zero based time step indices indicating the end of each stress period `length(nt) == nper+1` `nt(1)` = 0 indicates the initial drawdown condition at the start of the simulation the number of time steps for each stress period is `diff(nt)` if the simulation is steady state, `nt = [0 1]`
`dt`	`double` vector with time steps Δt [T] `length(dt) == nt(end)` if the simulation is steady state, `dt` is ignored, but may not be empty or zero
`qrc`	`nz` x `nr` `double` matrix with radial conductances Q^{rc} [L^2/T] `qrc(:,1) = 0` corresponds to the inner model boundary
`qzc`	`nz` x `nr` `double` matrix with vertical conductances Q^{zc} [L^2/T] `qzc(1,:) = 0` corresponds to the upper model boundary if the model has one layer (`nz==1`), `qzc` is ignored, but may not be empty
`qrzc`	`nz` x `nr` `double` matrix with sum of conductances [L^2/T] i.e. sum of upper and lower vertical conductances and inner and outer conductances
`qssc`	`nz` x `nr` `double` matrix with storage change terms Q^{ssc} [L^2] if the simulation is steady state, all `qssc` entries must be zero
`nq`	total number of non-zero discharges (`double`)
`idq`	`nq` x 2 `int32` matrix with zero based stress period indices and grid indices of discharged rings or `idq = [iper(:),igrid(:)]` where `iper` is stress period index and `igrid` is linear grid index equal to `nr*iring+ilay` with `iring` and `ilay` the ring and layer indices respectively if `nq == 0`, `idq` is ignored, but may not be empty
`q`	`double` vector of length `nq` with discharges Q [L^3/T] corresponding to `idq` a positive discharge value means that water is withdrawn if `nq == 0`, `q` is ignored, but may not be empty
`ns0`	total number of non-zero initial drawdowns (`double`)
`ids0`	`ns0` x 2 `int32` matrix with zero based stress period indices and grid indices of rings with non-zero instantaneous head change or `ids0 = [iper(:),igrid(:)]` where `iper` is stress period index and `igrid` is linear grid index equal to `nr*iring+ilay` with `iring` and `ilay` the ring and layer indices respectively if `ns0 == 0`, `ids0` is ignored, but may not be empty
`s0`	`double` vector of length `ns0` with initial drawdowns s_0 [L] corresponding to `ids0` a positive initial drawdown value means that there is an instantaneous head rise

	if `ns0 == 0`, `s0` is ignored, but may not be empty
`ibound`	`nz` x `nr` `int8` matrix indicating the head property for each ring: `ibound` > 0 indicates an active, variable-head ring `ibound` = 0 indicates an inactive, impervious ring `ibound` < 0 indicates an active, constant-head ring
`delta`	criterion for convergence δ [L] (`double`)
`mni`	maximum number of iterations for one time step iteration (`double`)
`nparm`	number of SIP iteration parameters n_{parm} (`double`), which is a strictly positive value
`wseed`	iteration parameter seed σ for calculaton of SIP relaxation parameter ω (`double`)
`accl`	SIP accelerator α (`double`), which is a value between 0 and 1

Output	**Description**
`s`	`nz` x `nr` x (`nt(end)`+1) `double` matrix with drawdowns *s* [L] a positive drawdown value means that head has risen since the start of the simulation recall that `s(:,:,1) = s0{1}` if the simulation is steady state, `s` is `nz` x `nr` matrix
`niter`	`double` vector with number of iterations performed during each time step `length(niter) == nt(end)` if the simulation is steady state, `niter` is scalar

Function SIPconf

Description
MATLAB SIP routine to solve finite-difference equations for 2D, axi-symmetric, confined model

Syntax
```
[s,niter] = MAxSym.SIPconf(nz,nr,nper,nt,dt,...
                           qrc,qzc,qrzc,qssc,...
                           q,s0,ibound,...
                           delta,mni,nparm,wseed,accl)
```

Input	**Description**
`nz`	number of layers n_z
`nr`	number of rings n_r
`nper`	number of stress periods
`nt`	vector with time step indices indicating the end of each stress period `length(nt) == nper+1` `nt(1)` = 1 indicates the initial drawdown condition at the start of the simulation the number of time steps for each stress period is `diff(nt)` if the simulation is steady state, `nt = [1 2]`
`dt`	vector with time steps Δt [T] `length(dt) == nt(end)-1` if the simulation is steady state, `dt` is ignored, but may not be empty or zero
`qrc`	`nz` x `nr` matrix with radial conductances Q^{rc} [L^2/T] `qrc(:,1) = 0` corresponds to the inner model boundary
`qzc`	`nz` x `nr` matrix with vertical conductances Q^{zc} [L^2/T] `qzc(1,:) = 0` corresponds to the upper model boundary if the model has one layer (`nz==1`), `qzc` is ignored
`qrzc`	`nz` x `nr` matrix with sum of conductances [L^2/T] i.e. sum of upper and lower vertical conductances and inner and outer conductances
`qssc`	`nz` x `nr` matrix with storage change terms Q^{ssc} [L^2] if the simulation is steady state, all `qssc` entries must be zero

q	cell where q{i} is nz x nr matrix with ring discharges Q [L³/T] for stress period i length(q) == nper a positive discharge value means that water is withdrawn q is empty if all discharges are zero throughout the simulation q{i} is empty if all discharges are zero for stress period i
s0	cell where s0{i} is nz x nr matrix with initial drawdowns s_0 [L] at the start of stress period i (starting heads in case of a steady state simulation) length(s0) == nper a positive initial drawdown value means that there is an instantaneous head rise s0 is empty if all initial drawdowns are zero throughout the simulation s0{i} is empty if all initial drawdowns are zero for stress period i
ibound	nz x nr matrix indicating the head property for each ring: ibound > 0 indicates an active, variable-head ring ibound = 0 indicates an inactive, impervious ring ibound < 0 indicates an active, constant-head ring
delta	criterion for convergence δ [L]
mni	maximum number of iterations for one time step iteration
nparm	number of SIP iteration parameters n_{parm} (double), which is a strictly positive value
wseed	iteration parameter seed σ for calculaton of SIP relaxation parameter ω (double)
accl	SIP accelerator α (double), which is a value between 0 and 1

Output	**Description**
s	nz x nr x nt(end) matrix with drawdowns s [L] a positive drawdown value means that head has risen since the start of the simulation recall that s(:,:,1) = s0{1} if the simulation is steady state, s is nz x nr matrix
niter	vector with number of iterations performed during each time step length(niter) == nt(end)-1 if the simulation is steady state, niter is scalar

Function mexADIunconf

Description

ANSI C ADI routine to solve finite-difference equations for 2D, axi-symmetric, unconfined model

Syntax

```
[s,niter] = MAxSym.mexADIunconf(nz,nr,nper,nt,dt,...
                                 qrc,qzc,qssc,qsyc,D1,hskz,...
                                 nq,idq,q,ns0,ids0,s0,ibound,...
                                 delta,mni)
```

Input	**Description**
nz	number of layers n_z (double)
nr	number of rings n_r (double)
nper	number of stress periods (double)
nt	int32 vector with zero based time step indices indicating the end of each stress period length(nt) == nper+1 nt(1) = 0 indicates the initial drawdown condition at the start of the simulation the number of time steps for each stress period is diff(nt) if the simulation is steady state, nt = [0 1]
dt	double vector with time steps Δt [T] length(dt) == nt(end)

	if the simulation is steady state, `dt` is ignored, but may not be empty or zero
`qrc`	`nz` x `nr` `double` matrix with radial conductances Q^{rc} [L²/T]
	`qrc(:,1) = 0` corresponds to the inner model boundary
`qzc`	`nz` x `nr` `double` matrix with vertical conductances Q^{zc} [L²/T]
	`qzc(1,:) = 0` corresponds to the upper model boundary
	if the model has one layer (`nz==1`), `qzc` is ignored, but may not be empty
`qssc`	`nz` x `nr` `double` matrix with storage change terms Q^{ssc} [L²]
	if the simulation is steady state, all `qssc` entries must be zero
`qsyc`	`1` x `nr` `double` vector with specific yield terms Q^{syc} [L²]
	if the simulation is steady state, all `qsyc` entries must be zero
`D1`	saturated thickness *D(i)* [L] of upper phreatic layer before simulation starts (`double`)
`hskz`	`1` x `nr` `double` vector with vertical flow correction terms for the top layers
	vertical flow is not corrected in ring `j` if `hskz(j)<=0`
	if the model has one layer only (`nz==1`), `hskz` is ignored, but may not be empty
`nq`	total number of non-zero discharges (`double`)
`idq`	`nq` x 2 `int32` matrix with zero based stress period indices and grid indices of discharged rings or `idq = [iper(:),igrid(:)]`
	where `iper` is stress period index
	and `igrid` is linear grid index equal to `nr*iring+ilay`
	with `iring` and `ilay` the ring and layer indices respectively
	if `nq == 0`, `idq` is ignored, but may not be empty
`q`	`double` vector of length `nq` with discharges *Q* [L³/T] corresponding to `idq`
	a positive discharge value means that water is withdrawn
	if `nq == 0`, `q` is ignored, but may not be empty
`ns0`	total number of non-zero initial drawdowns (`double`)
`ids0`	`ns0` x 2 `int32` matrix with zero based stress period indices and grid indices of rings with non-zero instantaneous head change or `ids0 = [iper(:),igrid(:)]`
	where `iper` is stress period index
	and `igrid` is linear grid index equal to `nr*iring+ilay`
	with `iring` and `ilay` the ring and layer indices respectively
	if `ns0 == 0`, `ids0` is ignored, but may not be empty
`s0`	`double` vector of length `ns0` with initial drawdowns s_0 [L] corresponding to `ids0`
	a positive initial drawdown value means that there is an instantaneous head rise
	if `ns0 == 0`, `s0` is ignored, but may not be empty
`ibound`	`nz` x `nr` `int8` matrix indicating the head property for each ring:
	`ibound` > 0 indicates an active, variable-head ring
	`ibound` = 0 indicates an inactive, impervious ring
	`ibound` < 0 indicates an active, constant-head ring
`delta`	criterion for convergence δ [L] (`double`)
`mni`	maximum number of iterations for one time step iteration (`double`)

Output	**Description**
`s`	`nz` x `nr` x (`nt(end)`+1) `double` matrix with drawdowns *s* [L]
	a positive drawdown value means that head has risen since the start of the simulation
	recall that `s(:,:,1) = s0{1}`
	if the simulation is steady state, `s` is `nz` x `nr` matrix
`niter`	`double` vector with number of iterations performed during each time step
	`length(niter) == nt(end)`
	if the simulation is steady state, `niter` is scalar

Function ADIunconf

Description

MATLAB ADI routine to solve finite-difference equations for 2D, axi-symmetric, unconfined model

Syntax

```
[s,niter] = MAxSym.ADIunconf(nz,nr,nper,nt,dt,...
                             qrc,qzc,qssc,qsyc,D1,hskz,...
                             q,s0,ibound,...
                             delta,mni)
```

Input	**Description**
`nz`	number of layers n_z
`nr`	number of rings n_r
`nper`	number of stress periods
`nt`	vector with time step indices indicating the end of each stress period `length(nt) == nper+1` `nt(1)` = 1 indicates the initial drawdown condition at the start of the simulation the number of time steps for each stress period is `diff(nt)` if the simulation is steady state, `nt = [1 2]`
`dt`	vector with time steps Δt [T] `length(dt) == nt(end)-1` if the simulation is steady state, `dt` is ignored, but may not be empty or zero
`qrc`	`nz` x `nr` matrix with radial conductances Q^{rc} [L^2/T] `qrc(:,1) = 0` corresponds to the inner model boundary
`qzc`	`nz` x `nr` matrix with vertical conductances Q^{zc} [L^2/T] `qzc(1,:) = 0` corresponds to the upper model boundary if the model has one layer (`nz==1`), `qzc` is ignored
`qssc`	`nz` x `nr` matrix with storage change terms Q^{ssc} [L^2] if the simulation is steady state, all `qssc` entries must be zero
`qsyc`	`1` x `nr` vector with specific yield terms Q^{syc} [L^2] if the simulation is steady state, all `qsyc` entries must be zero
`D1`	saturated thickness *D(i)* [L] of upper phreatic layer before simulation starts
`hskz`	`1` x `nr` vector with vertical flow correction terms for the top layers vertical flow is not corrected in ring `j` if `hskz(j)<=0` if the model has one layer only (`nz==1`), `hskz` is ignored, but may not be empty

`q`	`cell` where `q{i}` is `nz` x `nr` matrix with ring discharges *Q* [L^3/T] for stress period `i`
	`length(q) == nper`
	a positive discharge value means that water is withdrawn
	`q` is empty if all discharges are zero throughout the simulation
	`q{i}` is empty if all discharges are zero for stress period `i`

`s0`	`cell` where `s0{i}` is `nz` x `nr` matrix with initial drawdowns s_0 [L] at the start of stress period `i` (starting heads in case of a steady state simulation)
	`length(s0) == nper`
	a positive initial drawdown value means that there is an instantaneous head rise
	`s0` is empty if all initial drawdowns are zero throughout the simulation
	`s0{i}` is empty if all initial drawdowns are zero for stress period `i`

`ibound`	`nz` x `nr` matrix indicating the head property for each ring: `ibound` > 0 indicates an active, variable-head ring `ibound` = 0 indicates an inactive, impervious ring `ibound` < 0 indicates an active, constant-head ring
`delta`	criterion for convergence δ [L]

`mni`	maximum number of iterations for one time step iteration

Output	**Description**
`s`	`nz` x `nr` x `nt(end)` matrix with drawdowns *s* [L] a positive drawdown value means that head has risen since the start of the simulation recall that `s(:,:,1) = s0{1}` if the simulation is steady state, `s` is `nz` x `nr` matrix
`niter`	vector with number of iterations performed during each time step `length(niter) == nt(end)-1` if the simulation is steady state, `niter` is scalar

Function mexSIPunconf

Description
ANSI C SIP routine to solve finite-difference equations for 2D, axi-symmetric, unconfined model

Syntax
```
[s,niter] = MAxSym.mexSIPunconf(nz,nr,nper,nt,dt,...
                                qrc,qzc,qssc,qsyc,D1,hskz,...
                                nq,idq,q,ns0,ids0,s0,ibound,...
                                delta,mni,nparm,wseed,accl)
```

Input	**Description**
`nz`	number of layers n_z (`double`)
`nr`	number of rings n_r (`double`)
`nper`	number of stress periods (`double`)
`nt`	`int32` vector with zero based time step indices indicating the end of each stress period `length(nt) == nper+1` `nt(1)` = 0 indicates the initial drawdown condition at the start of the simulation the number of time steps for each stress period is `diff(nt)` if the simulation is steady state, `nt = [0 1]`
`dt`	`double` vector with time steps *Δt* [T] `length(dt) == nt(end)` if the simulation is steady state, `dt` is ignored, but may not be empty or zero
`qrc`	`nz` x `nr` `double` matrix with radial conductances Q^{rc} [L²/T] `qrc(:,1) = 0` corresponds to the inner model boundary
`qzc`	`nz` x `nr` `double` matrix with vertical conductances Q^{zc} [L²/T] `qzc(1,:) = 0` corresponds to the upper model boundary if the model has one layer (`nz==1`), `qzc` is ignored, but may not be empty
`qssc`	`nz` x `nr` `double` matrix with storage change terms Q^{ssc} [L²] if the simulation is steady state, all `qssc` entries must be zero
`qsyc`	`1` x `nr` `double` vector with specific yield terms Q^{syc} [L²] if the simulation is steady state, all `qsyc` entries must be zero
`D1`	saturated thickness *D(i)* [L] of upper phreatic layer before simulation starts (`double`)
`hskz`	`1` x `nr` `double` vector with vertical flow correction terms for the top layers vertical flow is not corrected in ring `j` if `hskz(j)<=0` if the model has one layer only (`nz==1`), `hskz` is ignored, but may not be empty
`nq`	total number of non-zero discharges (`double`)
`idq`	`nq` x 2 `int32` matrix with zero based stress period indices and grid indices of discharged rings or `idq = [iper(:),igrid(:)]` where `iper` is stress period index

	and `igrid` is linear grid index equal to `nr*iring+ilay` with `iring` and `ilay` the ring and layer indices respectively if `nq == 0`, `idq` is ignored, but may not be empty
`q`	`double` vector of length `nq` with discharges Q [L^3/T] corresponding to `idq` a positive discharge value means that water is withdrawn if `nq == 0`, `q` is ignored, but may not be empty
`ns0`	total number of non-zero initial drawdowns (`double`)
`ids0`	`ns0` x 2 `int32` matrix with zero based stress period indices and grid indices of rings with non-zero instantaneous head change or `ids0 = [iper(:),igrid(:)]` where `iper` is stress period index and `igrid` is linear grid index equal to `nr*iring+ilay` with `iring` and `ilay` the ring and layer indices respectively if `ns0 == 0`, `ids0` is ignored, but may not be empty
`s0`	`double` vector of length `ns0` with initial drawdowns s_0 [L] corresponding to `ids0` a positive initial drawdown value means that there is an instantaneous head rise if `ns0 == 0`, `s0` is ignored, but may not be empty
`ibound`	`nz` x `nr` `int8` matrix indicating the head property for each ring: `ibound` > 0 indicates an active, variable-head ring `ibound` = 0 indicates an inactive, impervious ring `ibound` < 0 indicates an active, constant-head ring
`delta`	criterion for convergence δ [L] (`double`)
`mni`	maximum number of iterations for one time step iteration (`double`)
`nparm`	number of SIP iteration parameters n_{parm} (`double`), which is a strictly positive value
`wseed`	iteration parameter seed σ for calculaton of SIP relaxation parameter ω (`double`)
`accl`	SIP accelerator α (`double`), which is a value between 0 and 1

Output	**Description**
`s`	`nz` x `nr` x (`nt(end)`+1) `double` matrix with drawdowns s [L] a positive drawdown value means that head has risen since the start of the simulation recall that `s(:,:,1) = s0{1}` if the simulation is steady state, `s` is `nz` x `nr` matrix
`niter`	`double` vector with number of iterations performed during each time step `length(niter) == nt(end)` if the simulation is steady state, `niter` is scalar

Function `SIPunconf`

Description
MATLAB SIP routine to solve finite-difference equations for 2D, axi-symmetric, unconfined model

Syntax
```
[s,niter] = MAxSym.SIPunconf(nz,nr,nper,nt,dt,...
                             qrc,qzc,qssc,qsyc,D1,hskz,...
                             q,s0,ibound,...
                             delta,mni,nparm,wseed,accl)
```

Input	**Description**
`nz`	number of layers n_z
`nr`	number of rings n_r
`nper`	number of stress periods
`nt`	vector with time step indices indicating the end of each stress period `length(nt) == nper+1`

	`nt(1)` = 1 indicates the initial drawdown condition at the start of the simulation the number of time steps for each stress period is `diff(nt)` if the simulation is steady state, `nt = [1 2]`
`dt`	vector with time steps *Δt* [T] `length(dt) == nt(end)-1` if the simulation is steady state, `dt` is ignored, but may not be empty or zero
`qrc`	`nz` x `nr` matrix with radial conductances Q^{rc} [L²/T] `qrc(:,1) = 0` corresponds to the inner model boundary
`qzc`	`nz` x `nr` matrix with vertical conductances Q^{zc} [L²/T] `qzc(1,:) = 0` corresponds to the upper model boundary if the model has one layer (`nz==1`), `qzc` is ignored
`qssc`	`nz` x `nr` `double` matrix with storage change terms Q^{ssc} [L²] if the simulation is steady state, all `qssc` entries must be zero
`qsyc`	`1` x `nr` vector with specific yield terms Q^{syc} [L²] if the simulation is steady state, all `qsyc` entries must be zero
`D1`	saturated thickness *D(i)* [L] of upper phreatic layer before simulation starts
`hskz`	`1` x `nr` vector with vertical flow correction terms for the top layers vertical flow is not corrected in ring `j` if `hskz(j)<=0` if the model has one layer only (`nz==1`), `hskz` is ignored, but may not be empty

`q`	`cell` where `q{i}` is `nz` x `nr` matrix with ring discharges *Q* [L³/T] for stress period `i`
	`length(q) == nper`
	a positive discharge value means that water is withdrawn
	`q` is empty if all discharges are zero throughout the simulation
	`q{i}` is empty if all discharges are zero for stress period `i`

`s0`	`cell` where `s0{i}` is `nz` x `nr` matrix with initial drawdowns s_0 [L] at the start of stress period `i` (starting heads in case of a steady state simulation)
	`length(s0) == nper`
	a positive initial drawdown value means that there is an instantaneous head rise
	`s0` is empty if all initial drawdowns are zero throughout the simulation
	`s0{i}` is empty if all initial drawdowns are zero for stress period `i`

`ibound`	`nz` x `nr` matrix indicating the head property for each ring: `ibound` > 0 indicates an active, variable-head ring `ibound` = 0 indicates an inactive, impervious ring `ibound` < 0 indicates an active, constant-head ring
`delta`	criterion for convergence *δ* [L]
`mni`	maximum number of iterations for one time step iteration
`nparm`	number of SIP iteration parameters n_{parm} (`double`), which is a strictly positive value
`wseed`	iteration parameter seed *σ* for calculaton of SIP relaxation parameter *ω* (`double`)
`accl`	SIP accelerator *α* (`double`), which is a value between 0 and 1

Output	**Description**
`s`	`nz` x `nr` x `nt(end)` matrix with drawdowns *s* [L] a positive drawdown value means that head has risen since the start of the simulation recall that `s(:,:,1) = s0{1}` if the simulation is steady state, `s` is `nz` x `nr` matrix
`niter`	vector with number of iterations performed during each time step `length(niter) == nt(end)-1` if the simulation is steady state, `niter` is scalar

Appendix: members of class Check

Property eps

Description
Maximum absolute difference used by method `equal` to test equality of floating point numbers

Attributes
`Constant, GetAccess = protected`

Value
`1e-15`

Method scalar

Description
Checks if MATLAB variable is scalar

Attributes
`Static, Access = protected`

Syntax
`ok = MAxSym.Check.scalar(x,allowempty)`

Input	**Description**
`x`	array to check
`allowempty`	logical indicating whether value may be empty (`true`) or not (`false`, default) optional argument

Output	**Description**
`ok`	`true` if `x` is a scalar, `false` otherwise

Method vector

Description
Checks if MATLAB variable is vector

Attributes
`Static, Access = protected`

Syntax
`ok = MAxSym.Check.vector(x,allowscalar,allowempty)`

Input	**Description**
`x`	array to check
`allowscalar`	logical indicating whether value may be scalar (`true`) or not (`false`, default) optional argument
`allowempty`	logical indicating whether value may be empty (`true`) or not (`false`, default) optional argument

Output	**Description**
`ok`	`true` if `x` is a vector, `false` otherwise

Method `length`

Description
Checks if MATLAB variable is vector of given length

Syntax
`ok = MAxSym.Check.length(v,len,allowscalar,allowempty)`

Input	**Description**
`v`	vector to check
`len`	required vector length
`allowscalar`	logical indicating whether vector may be scalar (`true`) or not (`false`, default) optional argument
`allowempty`	logical indicating whether vector may be empty (`true`) or not (`false`, default) optional argument

Output	**Description**
`ok`	`true` if `v` has required length `len`, `false` otherwise

Method `size`

Description
Checks if MATLAB variable is array of given size

Syntax
`ok = MAxSym.Check.size(a,siz,allowsingleton,allowempty)`

Input	**Description**
`a`	array to check
`siz`	required array size
`allowsingleton`	logical indicating whether singleton dimensions are allowed (`true`) or not (`false`, default) optional argument
`allowempty`	logical indicating whether array may be empty (`true`) or not (`false`, default) optional argument

Output	**Description**
`ok`	`true` if `a` has required size `siz`, `false` otherwise

Method `integer`

Description
Checks if MATLAB variable is numeric array containing integer elements

Syntax
`ok = MAxSym.Check.integer(x)`

Input	**Description**
`x`	array to check

Output	**Description**
`ok`	`true` if `x` has integer elements, `false` otherwise

Method `logical`

Description
Checks if MATLAB variable is logical array
Numeric arrays are also allowed, since numeric values can be interpreted as logicals

Syntax
```
ok = MAxSym.Check.logical(x)
```

Input	**Description**
`x`	array to check

Output	**Description**
`ok`	`true` if `x` is empty, logical, or numeric; `false` otherwise

Method `between`

Description
Checks if elements of numeric array are inside given interval

Precondition
Input array is numeric

Syntax
```
ok = MAxSym.Check.between(x,xlim,incl)
```

Input	**Description**
`x`	numeric array to check
`xlim`	vector `[xmin xmax]` indicates interval boundaries
`incl`	logical vector `[include_xmin include_xmax]` indicates whether interval boundaries must be included or not

Output	**Description**
`ok`	`true` if all elements of `x` are inside interval `[xmin xmax]`, `false` otherwise

Method `repmat`

Description
Replicates array to full size array
Returns empty array if input array is empty

Precondition
```
size(a,i) == 1 | size(a,i) == siz(i)
```

Syntax
```
f = MAxSym.Check.repmat(a,siz)
```

Input	**Description**
`a`	array to replicate
`siz`	required full size

Output	**Description**
`f`	full size array: `size(f,i) == siz(i)`

Method `repcellelements`

Description
Replicates cell elements to full size array
Empty cell elements remain empty

Precondition
`size(c{n},i) == 1 | size(c{n},i) == siz(i)`

Syntax
`f = MAxSym.Check.repcellelements(c,siz)`

Input	**Description**
`c`	cell with elements to replicate
`siz`	required full size

Output	**Description**
`f`	cell containing full size elements: `size(f{n},i) == siz(i)`

Method `equal`

Description
Replicates cell elements to full size array
Empty cell elements remain empty
Equivalent to operator `==`, but less strict, because `MAxSym.Check.eps` is used to check equality

Precondition
`x` and `y` are numeric arrays of same size
one of both arrays may be scalar

Syntax
`tf = MAxSym.Check.equal(x,y)`

Input	**Description**
`x`	numeric array
`y`	numeric array

Output	**Description**
`tf`	logical array of same size as `x` and `y` `true` if `abs(x-y) < MAxSym.Check.eps`, `false` otherwise

Method `equalsize`

Description
Checks if MATLAB variables have the same size

Syntax
`ok = MAxSym.Check.equalsize(x,y,allowsingleton,allowempty)`

Input	**Description**
`x`	array
`y`	array

`allowsingleton`	logical indicating whether `x` and `y` may have singleton dimensions (`true`) or not (`false`, default) singleton dimensions are not considered if `allowsingleton` is `true` optional argument
`allowempty`	logical indicating whether `x` and `y` may be empty (`true`) or not (`false`, default) `ok` is always `true` if `x` or `y` is empty and `allowempty` is `true` optional argument

Output	**Description**
`ok`	`true` if `x` and `y` have the same size, `false` otherwise

References

Bakker, M., 2010. TTim, a multi-aquifer transient analytic element model version 0.01. Water Resources Section, Civil Engineering and Geosciences, Delft University of Technology, 24pp. http://ttim.googlecode.com.

Bakker, M., Strack, O.D.L., 2003. Analytic elements for multiaquifer flow. Journal of Hydrology 271, 119-129.

Boulton, N.S., 1954. The drawdown of the watertable under non-steady conditions near a pumped well in an unconfined formation. Proc. Inst. Civil Engrs. 3, 564-579.

Boulton, N.S., 1963. Analysis of data from non-equilibrium pumping tests allowing for delayed yield from storage. Proc. Inst. Civil Engrs. 26, 469-482.

Butler, J.J., 1988. Pumping tests in nonuniform aquifers – the radially symmetric case. Journal of Hydrology 101, 15-30.

Cooley, R.L., 1972. Numerical simulation of flow in an aquifer overlain by a water table aquitard. Water Resources Research 8(4), 1046-1050.

Cooley, R.L., Case, C.M., 1973. Effect of a watertable aquitard on drawdown in an underlying pumping aquifer. Water Resources Research 9(2), 434-447.

Cooper, H.H., Jacob, C.E., 1946. A generalized graphical method for evaluating formation constants and summarizing well field history. Transactions American Geophysical Union 27, 526-534.

Cooper, H.H., Bredehoeft, J.D., Papadopulos, I.S., 1967. Response of a finite-diameter well to an instantaneous charge of water. Water Resources Research 3(1), 263-269.

Dantzig, G.B., 1947. Maximization of a linear function of variables subject to linear inequalities, In: Koopmans, T.C. (Ed.) Activity Analysis of Production and Allocation, Wiley & Chapman-Hall, New York-London, 1951, pp. 339-347.

De Glee, G.J., 1930. Over grondwaterstromingen bij wateronttrekking door middel van putten, Proefschrift Technische Hogeschool Delft, Waltman, Delft, 175pp.

Domenico, P.A., Schwartz, F.W., 1998. Physical and Chemical Hydrogeology, 2nd edn., John Wiley & Sons Inc, New York, 506pp.

Dupuit, J., 1863. Etudes Thèoriques et Pratiques sur le Mouvement des Eaux dans les Canaux Dècouverts et à travers les Terrains Permèables, 2nd edn., Dunot, Paris, 304pp.

Haitjema, H.M., 1995. Analytical Element Modeling of Groundwater Flow, Academic Press, San Diego, 394pp.

Hantush, M.S., 1964. Hydraulics of wells, In: Chow, V.T. (Ed.) Advances in Hydroscience, Vol I, Academic Press, New York and London, pp. 281-432.

Hantush, M.S., 1966. Analysis of data from pumping tests in anisotropic aquifers. Journal of Geophysical Research 71, 421-426.

Hantush, M.S., Jacob, C.E., 1955. Non-steady radial flow in an infinite leaky aquifer. Transactions American Geophysical Union 36, 95-100.

Harbaugh, A.W., 2005. MODFLOW-2005, the U.S. Geological Survey modular ground-water model - the Ground-Water Flow Process. U.S. Geological Survey Techniques and Methods 6-A16, 253pp.

Helm, D.C., 1976. One-dimensional simulation of aquifer system compaction near Pixley, California – 2. Stress-dependent parameters. Water Resources Research 12(3), 375-391.

Hemker, C.J., 1984. Steady groundwater flow in leaky multiple-aquifer systems. Journal of Hydrology 72, 355-374.

Hemker, C.J., 1985. Transient well flow in leaky multiple-aquifer systems. Journal of Hydrology 81, 111-126.

Hyder, Z, Butler, J.J., McElwee, C.D., Liu, W., 1994. Slug tests in partially penetrating wells. Water Resources Research 30(11), 2945-2957.

Kruseman, G.P., de Ridder, N.A., 1990. Analysis and Evaluation of Pumping Test Data, 2nd Edn., ILRI Publication 47, Wageningen, 377pp.

Lebbe, L., 1999. Hydraulic Parameter Identification. Generalized Interpretation Method for Single and Multiple Pumping Tests, Springer-Verlag, Berlin Heidelberg, 359pp.

Levenberg, K., 1944. A method for the solution of certain non-linear problems in least squares. The Quarterly of Applied Mathematics 2, 164-168.

Louwyck, A., Vandenbohede, A., Bakker, M., Lebbe, L., in press. Simulation of axi-symmetric flow towards wells: A finite-difference approach. Computers & Geosciences. doi:10.1016/j.cageo.2011.09.004.

Marquardt, D., 1963. An algorithm for least-squares estimation of nonlinear parameters. SIAM Journal on Applied Mathematics 11(2), 431-441.

McDonald, M.G., Harbaugh, A.W., 1988. A modular three-dimensional finite-difference ground-water flow model. USGS, Techniques of Water-Resources Investigations 06-A1, 588pp.

Papadopulos, I.S., Cooper, H.H., 1967. Drawdown in a well of large diameter. Water Resources Research 3(1), 241-244.

Neuman, S.P., 1972. Theory of flow in unconfined aquifers considering delayed response of the watertable. Water Resources Research 8(4), 1031-1045.

Reilly, T.E., Harbaugh, A.W., 1993. Simulation of cylindrical flow to a well using the US Geological Survey Modular Finite-Difference Ground-Water Flow Model. Ground Water 31(3), 489–494.

Shampine, L.F., 2008. Vectorized Adaptive Quadrature in MATLAB. Journal of Computational and Applied Mathematics 211, 131-140.

Stone, H.L., 1968. Iterative solution of implicit approximations of multidimensional partial differential equations. SIAM Journal of Numerical Analysis 5, 530-538.

Theis, C.V., 1935. The relation between the lowering of the piezometric surface and the rate and duration of discharge of a well using ground-water storage. American Geophysical Union Transactions 16, 519-524.

Thiem, G., 1906. Hydrologische Methoden. Gebhardt, Leipzig, 56pp.

Verruijt, A., 1970. Theory of Groundwater Flow. Macmillan Press, London, 144pp.

Wang, H.F., Anderson, M.P., 1982. Introduction to Groundwater Modeling: Finite Difference and Finite Element Methods, WH Freeman and Company, San Francisco, 237pp.

Weeks, E.P., 1969. Determining the ratio of horizontal to vertical permeability by aquifer-test analysis. Water Resources Research 5, 196-214.

Yeh, H.D., Yang, S.Y., 2006. A novel analytical solution for a slug test conducted in a well with a finite-thickness skin. Advances in Water Resources 29, 1479-1489.