The ModAEM Book
Version 1.9.0

Vic Kelson, Ph.D. PE
WHPA Inc.
320 West 8th St.
Bloomington, IN 47404

September 19, 2013

Contents

List of Figures

Chapter 1

Introduction

This manual describes the results of nearly 20 years of effort (much of it the "on again, off again" variety). ModAEM has been through four major revisions, several research branches, and has been used for countless wellhead protection models, even though most ModAEM users never knew they were using it! This chapter is to introduces to the history and philosophy of ModAEM.

Early ModAEM

The essential design of ModAEM was conceived during a series of long discussions between Mark Bakker, Steve Kraemer, and myself during a High Performance Computing Consortium workshop at EPA's Research Triangle Park facility in 1995 (I was first introduced to the "new" GMS preprocessor at the same meeting). I'm not sure that Steve and Mark remember the meeting or the design ideas, but they basically remain intact:

- Logical separation of the analytic element *functions* (dipoles/doublets, wells, etc.) from analytic element *applications* (wells, rivers, inhomogeneities)
- Implementation of efficient functions for a few very flexible analytic element functions (at that time, only wells, dipoles of complex strength, and ponds were anticipated)
- Make it all parallelizable with very little effort

- Abandon FORTRAN 77 in favor of a more modern programming language

I have been asked many times why ModAEM was written in Fortran-90 (and now in Fortran-95), rather than in a "better" language like C++. I struggled with the choice, but in the end I was advised by the folks at CICA (Center for Innovative Computer Applications) that Fortran 90 and High-Performance Fortran was a more flexible and workable alternative for parallel hardware. On many occasions, I've wished I had a fully object–oriented language like C++, but I've never wished I'd had all the migration issues as C++ has evolved for the past decade. Fortran 90/95 was a fine choice. Now, I can compile ModAEM with OpenMP for multi-core processors using a totally free toolchain!

WhAEM for Windows

Later in 1995, Henk Haitjema and I were hired by US EPA to develop a new version of the WhAEM for DOS code that had been released in 1994. The new WhAEM was to run under Windows 95 and its successors. We needed a free solver engine that had available source code (GFLOW was fully proprietary back then), so we decided that I should complete a simple ModAEM for use in WhAEM. This code eventually became ModAEM-1.0, and it was the computational heart of WhAEM until it was replaced with a stripped–down GFLOW in 2002.

ModAEM-1.0, or "EPA ModAEM" is a public–domain code that supports discharge–specified wells, discharge–specified and head–specified line sinks, no–flow boundaries, uniform flow, and recharge ponds. It also does streamline tracing and generates grids for contour lines. To the best of my knowledge, EPA still has the source code on the CoSMOS web site.

In addition to its use in WhAEM, I used ModAEM-1.0 as a research code for local 3-D models in my disseratation work. Very little of that code exists now.

The SFWMD years

In 1998, I finished my Ph.D. at Indiana and began working for the South Florida Water Management District in West Palm Beach, FL. I did some work with ModAEM, typically in the guise of WhAEM, and implemented a few additional features, including resistance line sinks. I also did experimental work on a regional transient flow model based on ModAEM. ModAEM was a useful research project, but there were few new applications.

In 2000, my wife, infant son, and I returned to Bloomington, where I joined WHPA Inc., a small consulting firm led by my best friend, Jack Wittman.

ModAEM reborn – the Idaho delineation project

In late 2000, WHPA teamed with Barr Engineering on a challenging project: the delineation of wellhead protection areas for over 450 wells in the Treasure Valley, near Boise. Two previous MODFLOW models, based on grid cells of a half–mile or more in size, were available. We needed a way to quickly delineate accurate capture zones for the wells based on the MODFLOW flow fields. I proposed that we rebuild ModAEM for the purpose, using the integrated–flux boundary conditions to chop sub–domains out of the MODFLOW models; we could then trace particles in the equivalent analytic element domains.

This effort required some substantial enhancements to ModAEM: inhomogeneities in transmissivity, rectangular area sinks, and bounded model domains. In addition, I restructured the internals of ModAEM to make it more object–based, including an "iterator" strategy that improved the independence of the code components.

The resulting code became ModAEM-1.2. I planned to make a full release of ModAEM-1.2 when the manual was ready, but time was limited, and it was not widely distributed. For some internal applications, I added head–specified wells, general polygonal area sinks and 3–D pathline tracing. I began looking for other developers who would be interested in developing ModAEM, and decided to release a version under the GNU General Public License, as soon as I could.

And by the way, we finished the job on time and within the budget!

ModAEM in GMS?

After the completion of the Idaho project, Jack and I presented the design and philosphy behind ModAEM in a seminar at EPA's Athens Lab. The positive feedback I received from the software engineers there was encouraging. Shortly thereafter, Norm Jones, the inventor of GMS, contacted me to inquire about the potential for using ModAEM as an analytic element solver for GMS. Norm and I had numerous discussions, and a plan was developed for the work. I set out to add some functions that would make the model sufficiently generic to be useful in GMS: base and thickness inhomogeneities, including common boundaries; improved streamline tracing; improved bounded domains with general–head boundaries; drains; and a number of GUI-related improvements were needed.

Plus, a manual. An early version of this manual was provided to Alan Lemon at BYU, and he built the GMS front–end for ModAEM. That work led to the release of ModAEM-1.4 in February, 2004.

What's Next?

After many years of work and several false starts, ModAEM is now a versatile modeling tool with a growing community of users. I am hopeful that a developer community will develop, and ModAEM will continue to grow in flexibility, performance, and modeling power. If you're interested in any aspect of ModAEM development, testing, validation, documentation, or if you want to fund the effort somehow, please visit the ModAEM Home Page at `http://www.wittmanhydro.com/modaem`.

Like me, ModAEM now has two children. Mark Bakker and I worked together on the design of the fully object–oriented Python AE code Tim (now TimSL). Nearly all of the solver logic and the internal organization is derived from ModAEM's solver; however since Tim is a research code, the parallelism–improving separation of functions from applications is omitted. TimSL now has a sibling, TimML, Mark Bakker's Bessel–function based model.

The Tim project provides a set of tools for analytic element education and research. ModAEM remains a stable, high–performance production code. Both projects continue to cross–pollinate each other. I expect that I will continue to develop and model with both codes for a long time to come.

1.1 The Philosophy of ModAEM

ModAEM has a set of governing principles. These have been constant throughout the development of the code and I don't anticipate changing them:

Keep it free This is embodied in the choice of the GNU General Public License for ModAEM-1.4 and beyond. Everyone is welcome to the source code, but if you enhance it and distribute it, you must distribute the source code for the enhancements.

Keep it simple That is, use as few mathematical functions as is necessary to achieve the desired objectives. To this day, ModAEM uses only relatively simple first–order and second–order elements. These are reliable, efficient, predictable, and sufficient for a wide variety of practical applications.

Keep it parallel Make sure that the original design goal of a parallel analytic element code remains. The current code should be easily compiled with the –parallel switch with compilers that support SMP hardware. I wish I had one.

Keep it portable ModAEM does not use extensions to the Fortran 95 language. It is clean and standard throughout. It has been successfully compiled on numerous hardware and software platforms.

Assume there's a preprocessor Nowadays, nobody wants to use a model that doesn't have a nice preprocessor. As a result, ModAEM simplifies the I/O model as much as possible. Many difficult tasks are expected to be performed by the preprocessor, for example, developing the topology of aquifer sub–domains. This makes the code more robust, but harder to use "by hand". I for one prefer using a good preprocessor.

Document the code ModAEM has always had detailed documentation built into the source code.

Have fun! For me, ModAEM has always been a pleasure and a great intellectual challenge. If it isn't fun, it isn't worth working on it!

1.2 How to read this manual

The following conventions are used for formatting in this manual:

`typewriter` text is used for sample input files

sans-serif text is used for file names and other related items

bold sans-serif text is used for ModAEM script file directives

bracketed expressions such as $[L]$ contain the dimensions for input data. Some examples are $[-]$ for dimensionless quantities, $[L]$ for length, $[L/T]$ for length per time (e.g. a hydraulic conductivity of 100 ft/d).

1.3 Conventions for numeric input

ModAEM makes use of "free–format" input for all numeric entries. Since the computational heart of ModAEM is based on complex numbers, most coordinate information is entered as complex quantities. In this manual, the type of the input value is provided in curly braces, e.g. $\{int\}$ for an integer value. When directed to provide a numeric value, use the following rules:

integer values $\{int\}$ Positive or negative integer values are allowed. ModAEM uses the Fortran 90 `SELECTED_INT_KIND` function to specify Chapter 1 About the Book the size of integer values (the default is 4 bytes per value). On 64-bit hardware (or on 32-bit hardware with compilers that support "quad–precision" values, you may overload the parameter `ModAEM_Integer` in u_constants.f95 and rebuild ModAEM. Note that this has not been tested; please report success or failure to Vic Kelson.
For integers, do not provide a decimal point. For negative numbers, the −sign *must* be immediately before the first digit of the value.

right: `3` `-13241`

wrong: `-1.234` `7E+07`

logical values $\{logical\}$ Values that may be either true or false. ModAEM uses the Fortran 90 `SELECTED_LOGICAL_KIND` function to specify the size of logical values (the default is 4 bytes per value). If desired, you may overload the parameter `ModAEM_Logical` in u_constants.f95 and rebuild ModAEM.
For logicals, the only legal values are `T` (true) or `F` (false). The value may be entered in either uppercase or lowercase.

right: `T` `F`

wrong: `yes` `0`

real values {*real*} Positive or negative floating–point values are allowed. ModAEM uses the Fortran 90 `SELECTED_REAL_KIND` technique to specify the size of floating–point values (the default is 8-bytes per value). On 64-bit hardware (or on 32-bit hardware with compilers that support "quad–precision" values, you may overload the parameter `ModAEM_Real` in u_constants.f95 and rebuild ModAEM[1]. Note that this has not been tested; please report success or failure to Vic Kelson.

Floating–point numbers may or may not contain a decimal point. If exponential notation is desired, use the characters $E \pm XX$ as a suffix, where `XX` is the exponent. No space can lie between the $-$ sign and the first digit of precision or between the mantissa and exponent.

right: `1.2    -3.1415926    6.02e+23`

wrong: `- 1.2    7.01 e+99`

complex values {*complex*} Pairs of floating–point values, surrounded by parentheses and separated by a comma are allowed, where the first value is the real part and the sceond value is the imaginary part. ModAEM uses the Fortran 90 `SELECTED_REAL_KIND` technique to specify the size of floating–point values, including complex values (the default is 8-bytes per value). On 64-bit hardware (or on 32-bit hardware with compilers that support "quad–precision" values, you may overload the parameter `ModAEM_Real` in u_constants.f95 and rebuild ModAEM. Note that this has not been tested; please report success or failure to Vic Kelson.

Complex numbers may or may not contain a decimal point. If exponential notation is desired, use the characters $E \pm XX$ as a suffix, where XX is the exponent. No space can lie between the $-$ sign and the first digit of precision or between the mantissa and exponent.

right: `(1.2,3.45)    (-3.1415926,0)`

wrong: `3+4i    (3.54)`

[1] Although in principle it would be possible to specify single precision, it is discouraged. On current 32-bit hardware, there is no speed benefit from the use of single precision floating–point arithmetic.

1.4 The right–hand rule

For some elements, e.g the boundary segments defined by the `bdy` directive in module AQU, the orientation of the points making up the element is significant. In all cases, ModAEM makes use of the “right–hand rule”. The element is oriented such that if the modeler were to stand at the first vertex facing the second vertex, the boundary condition is specified on the “left” side of the element (the index finger of the right hand, extended along the segment, points “in”). For example, for a flux–specified bdy element, the flux is numerically positive if water moves from the right to the left. Similarly, a head–specified boundary condition is to be met just to the left of the element.

Chapter 2

Groundwater modeling with ModAEM

How do I build an analytic element model with ModAEM? How does ModAEM work?

Using ModAEM is much like using any other analytic element code. Somehow, the modeler constructs a script file that controls the creation of the model, first defining the aquifer properties and their distribution, then adds elements to the aquifer that simulate various features in the flow system, such as rivers and wells. The script file also directs the solution of the model problem and uses various analytical tools to extract results as grids, trace streamlines, and write reports. Onc

2.1 About the AEM

This section introduces some important principles of analytic element models, with the purpose of introducing the new ModAEM user to analytic elements. Those who are interested in the "gory details" of analytic elements should see Strack, 1989. For a more complete discussion of modeling issues with analytic elements, please see Haitjema, 1995.

What is an analytic element?

The analytic element method is based on the superposition of analytic functions. An *analytic element* is a mathematical function that may be superimposed with other analytic elements to create a complete solution for a groundwater problem. In practice, two–dimensional, steady–state analytic elements come in two varieties,

Elements that satisfy Laplace's Equation

2.1.1 Discharge potentials

2.1.2 Example solutions

2.1.3 Analytic element functions and superposition

2.1.4 Boundary conditions for complex models

2.2 Using the AEM

2.2.1 Keep it simple

2.2.2 Stepwise modeling approach

2.2.3 Gotchas and troubleshooting

Chapter 3

ModAEM script files

ModAEM execution is controlled by the use of a ModAEM script file (with the extension .aem), and a ModAEM name file (called modaem.nam). These files are the only input files that are required by the model.

3.1 The ModAEM name file **modaem.nam**

The standard library for the Fortran-95 language does not provide a mechanism for gathering command-line arguments (e.g. the 'int main(int argc, char **argv)' in a C or C++ program or sys.argv[] in Python). Although nearly all current Fortran-95 compilers provide a library routine for this task, they are not syntactically consistent. One of the design objectives of the ModAEM project is that the code should be as portable as possible, so language extensions have been carefully avoided. Therefore, the official ModAEM release code uses a file called modaem.nam in the current working directory when the program begins execution to find the ModAEM script file. The name file provides the base file name for the ModAEM script file (and may provide other features in the future). Developers are encouraged to add platform-specific support for command-line arguments if they desire [1].

[1] It is expected that developers (particularly those who need a platform-specific version of ModAEM) will add the ability to fetch command-line arguments in the version of ModAEM that ships with their code. As of Fortran 95, the addition of this feature is compiler-dependent; however the Fortran 2003 library offers this capability, so future versions of ModAEM will likely take advantage of it.

3.1.1 Contents of the name file

The ModAEM name file modaem.nam can be created with any text editor. modaem.nam contains up to three lines of text, as follows:

Line 1 – Base file name for the model run The first line of the name file contains the "base" name of the files for the model run. For example, if the model input data are contained in a file called modaem.aem, then the contents of modaem.nam would be:

```
modaem
```

The extension .aem is appended to the file name by ModAEM. In addition, two output files will be created using the same base name, 'modaem':

modaem.err Will be written as the message file. This file echoes program input and messages issued during execution. It may be used as a run log file (and to see the results of some processing directives). Although some model results can be sent to the message file, it is not appropriate for extracting model results (e.g. heads) for use in GUI programs; the inquiry files written by module INQ (Section 5.9) are designed for this feature.

modaem.out Will be written as the output file. This file recieves an HTML document listing of the solution results, which may be useful in debugging. The output file is not appropriate for loading results to be displayed in GUI programs; the inquiry files written by module INQ (Section 5.9) are designed for this feature.

Line 2 – Name of the previous solution file The second file named in the name file is an optional file that contains the saved results from a previous solution that are to be reloaded, for example, to trace particles

It is understood that some vendors may wish to add a platform-specific GUI-style display to ModAEM (e.g. in a style similar to the MODFLOW/Win32 code that ships with the popular *Groundwater Vistas* MODFLOW GUI). It will be much appreciated if someone will make such a version available under GPL. Although it is much preferred that such an extension is released under GPL, the copyright holders understand the developer's concerns and will consider requests (we make no guarantees *a priori*) for special licensing arrangements for such extensions.However, we do not anticipate a willingness to engage in special licensing exceptions for computational features that may be added to the code.

from a prior solution without the need for re-solving. If it is provided, the results are read from the previous solution file after the problem is defined (see the **aem** directive and related information below), when the **sol** directive is encountered. To "solve" the model byCool pictures? Why would we do all this work if we weren't going to make cool pictures? – Mark Bakker

simply reloading the solution, issue the directive `sol 0` after the problem definition is complete. If the previous solution file is missing or if the file is empty, a warning will be issued and execution will continue. However, if the file is not empty, a fatal error results if the contents of the previous solution file are inconsistent with the problem definition.

The file name may have any extension that the modeler desires, although by convention, the extention .pre is recommended. If the name of previous solution file is omitted from the name file, no previous solution will be loaded.

Line 3 – Name of the file where solution results are to be written The third file named in the name file is an optional file where the results of the solution are to be written, for example, for use as the initial condition in a future ModAEM run. If it is provided, ModAEM will write the solution when the solution procedure is complete. The file may be reloaded by specifying it in the second line of a future name file (see above).

The file name may have any extension that the modeler desires, although by convention, the extention .pre is recommended. If the name of the save file is omitted from the name file, no solution will be saved.

3.2 The AEM script file (.aem file)

The AEM script file provides model elements and processing directives to ModAEM. The AEM script file can have any base file name (as specified in the modaem.nam file) and must have the extension .aem. The AEM script file is a flat text file that can be created with any text editor. Program directives are entered one per line.

The script file is divided into two sections, the problem definition section (or AEM section) and the processing section. Typically, ModAEM script files look like this:

```
aem
  aqu
  ...
  # aquifer description goes here
  end
  # other module sections go here...
  wl0 10
    ... discharge-specified well data goes in here ...
    # end of well data
  end
  # end of aem data
end
# processing directives go here ...
# End-of-data mark
eod
```

The attractive indentation style of the input file is optional, but highly recommended. By convention, each sub-section of the input file, e.g. wl0 or aqu, is indented two spaces for readability. ModAEM ignores the indentation when reading program input. Currently (at this writing, we are at version 1.8.7), ModAEM behaves unpredictibly when it encounters tab characters (ASCII 0x09) in input. The aem section of the input file (the portion contained between the aem and end directives in lines 1 and 14 of the above listing) defines the actual elements that make up the model. Within the AEM section, input for the various element definition modules (see Chapter ??) is provided.

The processing section follows the problem definition section of the script file. The various processing directives that are available are discussed in Chapter ??.

3.2.1 Directives which are common to all input modules

The following directives are available in all ModAEM input modules.

Comments

Comment lines in the AEM script file start with a hash mark (#) in the first column. Comment lines are ignored by ModaEM. For example:

```
# This is a comment line
```

Exiting a module (end directive)

The end directive causes ModAEM to leave the current module. For example, when in the wl0 module (which is started with the wl0 directive), the end directive returns processing to the AEM input module:

```
# the aem section is used to define the problem
aem
   # other module sections go here...
  wl0 10
    ... discharge-specified well data goes in here ...
     # end of well data
  end
  # end of aem data
end
# processing directives go here ...
# End of data mark
eod
```

3.2.2 Enabling debugging code (dbg directive)

The dbg directive is used to turn code marked as 'debug' code on or off during execution (useful for program debugging). Debug code is enabled or disabled at the level of a specific module. The ModAEM source code contains many assertions that can be used to test for internal errors in the code. The dbg command does not affect the detection of errors in program input, however. This command will typically be used only by developers.

3.2.3 Other directives for specific tasks

Begin defining a model problem domain (aem directive)

The aem directive begins the problem definition section of the ModAEM input file. See for a description of the various directives that may be used in the problem definition section.

Usage:

```
aem
```

```
  ... put model definition directives here ...
end
```

Processing directives

The various processing directives that are available once a problem has been defined in the AEM section are described in Chapter **??**.

Chapter 4

Aquifer specification

The aquifer module (AQU) is used to define the layout and properties of the ModAEM aquifer. In ModAEM, an aquifer is considered to be a single, horizontal, two-dimensional flow system. A ModAEM aquifer may be bounded or unbounded spatially, with a variety of boundary conditions specified at the perimeter of the bounded domain. All aquifers in ModAEM may have heterogeneous hydraulic conductivity, base elevation, thickness and porosity, specified in inhomogeneities. An inhomogeneity is a bounded sub-region of the aquifer where the hydraulic properties of the aquifer differ from the surrounding region. In ModAEM, inhomogeneities may be nested within other inhomogeneities, and may share common boundaries.

4.1 Aquifers in ModAEM

Like most analytic element models, ModAEM supports heterogeneous aquifers in which aquifer properties are constant within polygonal domains (aquifer properties are "piecewise–constant"). In ModAEM, each sub-domain may have its own aquifer base and top elevation, hydraulic conductivity, and porosity.

4.2 AQU Module Input

As with all other modules that are included in the problem definition section of a ModAEM script file, input for module AQU is contained between the `aqu` directive and the `end` directive. Module AQU differs from some of the other

ModAEM modules, in that it posesses optional submodules. The general layout for module AQU input is as follows:

```
\# Create an aquifer
aqu ndomains nstrings base thickness conductivity porosity
  ref <arguments> (optional) define a reference point and discharg
  bdy <arguments>
    (optional) define flow conditions at the perimeter
  end
  in0 <arguments>
    (optional) define inhomogeneities
  end
end
```

The remainder of this section describes the detailed usage of the AQU module directives.

4.2.1 Beginning the aquifer definition (directive aqu)

The aqu directive starts the process of defining the aquifer layout. In addition to the regional aquifer properties, the aqu directive allocates space for the definition of subdomains of differing aquifer properties, and boundary conditions for bounded models.

Usage:

```
  aqu ndom nstr bot thick hyd-cond poro avg-head
```

Parameters for the aqu directive:

ndom **(int)** The number of inhomogeneity domains in the aquifer, including the outside unbounded domain. For a single homogeneous aquifer, use 1. [−]

nstr **(int)** The number of strings that are used to bound the inhomogeneity domains. For a single inhomogeneous aquifer, use 0. [−]

bot **(real)** The base elevation of the aquifer [L]

thick **(real)** The thickness of the aquifer [L] `hyd-cond` **(real)** The hydraulic conductivity of the aquifer [L/T]

`poro` **(real)** Porosity of the aquifer as a fraction [−]

`avg-head` **(real)** This is an estimate of the average head in the aquifer. It is used to pre–condition the solution when inhomogeneities in base elevation or thickness are encountered and the flow condition is unconfined. If you expect to be using inhomogeneities in base elevation or aquifer thickness, a good estimate of the initial average head will greatly speed the solution process. If you are not using inhomogeneities in base elevation or aquifer thickness, it is recommended that the value of `bot` + `thick` be provided. [L]

Simple Aquifer Example To build a simple infinite aquifer with no inhomogeneities, with base elevation at 0 ft, thickness of 10 ft, hydraulic conductivity of 100 ft/d, porosity of 0.25 and an average head of 200 ft, only one domain and no strings are required. The `aqu` directives

```
aqu 1 0 0.0 10.0 100.0 0.25 200.0
end
```

will create this aquifer.

4.2.2 Defining a reference flow field (directive ref)

ModAEM differs from some analytic element codes in that the specification of a reference point is often unnecessary. Typically, the reference flow field is only needed in conceptual models or when a site–scale model based on a uniform flow field has been measured in the field.

What is the reference flow field? In an analytic element model, it is possible to define a problem in which all of the elements (wells, etc.) possess a specified discharge (e.g. the common conceptual model of one or more wells in a uniform flow field). In these cases, a unique solution can be generated only if the user specifies a point in the aquifer at which the head is known. Specification of the reference point enables the specification of a reference flow field, a uniform flow discharge that will be present everywhere in the aquifer, even if no elements are present. The reference flow field is an abstraction, and should be used with care, since it typically dominates the solution and increases the potential of calibration errors. In ModAEM, specification of the reference point is optional. If the reference point is omitted, ModAEM replaces it with a closure condition based upon continuity of flow. In ModAEM, there are three ways to provide the "far–field" regional flow conditions:

reference head and uniform flow In this case, the ref directive is provided. A reference point and a reference head, and also a far–field uniform flow discharge is provided. This strategy is typically used in conceptual problems and in simple site–scale problems where the hydraulic gradient has been measured from field observations. Do not place the reference point within the area of interest if other elements, e.g. line sinks, are to be used.

bounded aquifer domains ModAEM includes support for closed model domains via the bdy directive (see Section **??**, below). When a bounded aquifer is provided, the reference point and uniform flow rate must not be used.

unbounded aquifers with *modeled* far–field This is the most common strategy for large regional models to be built with analytic elements. Instead of a specific far–field flow condition, far–field elements are placed in the model domain. These elements generate the flow field at the perimeter of the study region, and "insulate" the study region from the infinite mathematical domain. In ModAEM, no reference point is required if at least one head–specified boundary condition is provided in the model. A continuity–of–flow condition is used instead [1]. If a reference point is specified far from the study region, the model will still function if the model is constructed properly (see Haitjema, 1995). Never place the reference point at a location within the study domain; unpredictable and incorrect model output may result.

The reference point and reference flow field are specified using the directive ref within the input for module AQU.

[1] In ModAEM-1.0 and ModAEM-1.2, the reference point was always required. In WhAEM, the GUI placed the reference point at a location far to the west of the extent or model elements, with a reference head set at the average of all head–specified conditions in the model. ModAEM-1.4 and later versions eliminate this requirement.

Usage:

```
  aqu ...
    ref (x0,y0) h0 (Qx0,Qy0)
  end
```

Parameters for the ref directive:

`(x0,y0)` **(complex)** The location of the reference point, $z_0 = x_0 + iy_0$. [L]

`head` **(real)** The head at the reference point. [L]

`(Qx0,Qy0)` **(complex)** The reference flow field, $Q_0 = Q_{x0} + iQ_{y0}$. The magnitude of the value Q_0 is computed as $Q_0 = k \times H \times dh/ds$ where k is the hydraulic conductivity, H is the saturated thickness and dh/ds is the hydraulic gradient in the direction of flow. The components are relative to the x-axis and y-axis; if θ is the angle relative to the x-axis in radians, the components Q_{x0} and Q_{y0} are computed as $Q_{x0} = Q_0 \cos\theta$ and $Q_{y0} = Q_0 \sin\theta$. [L^2/T]

Example of an aquifer with uniform flow and a reference point This reference aquifer definition creates an unbounded aquifer with a uniform flow field of 10.0 m/d in the positive x-direction, with a head of 100 *mathrm m* at the origin.

```
aqu 1 0 0.0 10.0 100.0 0.25
  ref (0,0) 100.0 (10.0,0.0)
end
```

4.3 Creating a bounded aquifer (bdy directive)

ModAEM supports the option of bounded aquifer domains, with a variety of boundary conditions specified at the perimeter of the flow domain (head specified, flux specified, and general–head). Bounded aquifers are defined in the ModAEM script file in two steps:

1. Specifying the perimeter of the aquifer and any islands (sub-regions within the aquifer that are to be ignored)

2. Specifying the boundary conditions for groundwater flow that are to be constructed at the perimeter.

It is important to understand that step (1) above actually has no affect on the flow problem; it simply tells ModAEM which regions of the potentially unbounded aquifer are to be ignored by the anaysis modules. This is convenient when computing grids for contouring. ModAEM returns an inactive region value whenever an analysis module requests the head (or discharge or other value) at a point in the inactive region. It is step (2) above that actually constructs the boundary conditions at the aquifer perimeter. Specification of the boundary conditions is sufficient to govern the flow problem[2].

4.3.1 Specifying conditions on the aquifer perimeter (bdy directive)

The bdy directive begins the definition of flow conditions at the perimeter of a bounded aquifer. For each segment of the boundary, a pair of vertices is provided, along with the boundary condition to be met along the segment. In the current release of ModAEM, this is the only instance where line segments are fully defined in this manner. When this was originally implemented in ModAEM-1.2, it was intended for the construction of ModAEM sub–domain models from large regional MODFLOW models (ref: Vic's talk at GSA – Denver). This inputstyle is more flexible for preprocessing in "telescoping" domain models. The boundary segments must uniquely bound a closed region; ModAEM will automatically build the bounding polygon for the bdy elements. A fatal error will be issued if the boundary elements do not bound a unique, conticuous polygonal region. If the model requires that inactive regions be included within a model (in either a bounded or unbounded aquifer), see the isl command within module HB0.

Usage:

```
aqu ...
  ref ...
  bdy nbdy
    (x1,y1) (x2,y2) head flux ghb-distance bdy-flag
    ...
```

[2] For example, the GMS user interface handles the perimeter computations internally.

```
end
```

Parameters for the **bdy** directive

`nbdy` **(int)** The maximum number of boundary elements in the problem. {*int*}

Specifying the boundary elements For each segment along the model's perimeter boundary, a line segment is provided using the parameters below. No more than `nbdy` boundary segments are allowed. The orientation of the boundary segments is important. ModAEM makes use of the "right–hand" rule convention (section **??**) is used. Fluxes are numerically positive if they cross the element from right–to–left. Note that this means that segments that define a polygonal outer boundary is *positively oriented* (counterclockwise). For segments that bound islands, the active region is on the left of a polygon if the vertices are arranged counter–clockwise.

`(x1,y1)` **(complex)** The first point on the line segment, $z_1 = x_1 + iy_1$. [L]

`(x2,y2)` **(complex)** The second point on the line segment, $z_2 = x_2 + iy_2$. [L]

`flux` **(real)** The value for the specified total volumetric flow rate across the segment, e.g. a cell-by-cell MODFLOW value. The value is positive if water moves from right to left when viewed along the path z_1z_2. [L^3/T]

`head` **(real)** The specified head at the center of the segment z_1z_2. [L]

`ghb-distance` **(real)** The perpendicular distance from the element to a presumed specified head located somewhere to the right of the element. The general–head boundary creates head–dependent flux (Neumann) condition across the element. This is analogous to a MODFLOW GHB cell located at the edge of the model. ModAEM computes a resistance for the GHB based on the provided distance and the transmissivity of the sub–domain on the left side of the element. [L]

`bdy-flag` **(int)** A flag that controls the boundary–condition type. Codes are 0 (head), 1 (flux), and 2 (general–head). [–]

When a head-specified or general-head condition is required, the value of the flux parameter is ignored. Similarly, when a flux-specified condition is required, the value of the head parameter is ignored. This is intended for use as check information when ModAEM is used to model a detailed region with perimeter boundaries from another model. ModAEM reports the specified and modeled heads and fluxes for all bdy elements in the output HTML file.

Note The bdy directives are part of module AQU. No end directive is to be used to terminate the input of boundary segments (the end directive terminates input to module AQU). This is distinguished from the inhomogeneity submodule IN0, discussed in section **??**.

4.4 The inhomogeneity submodule

The IN0 submodule provides the AQU aquifer module with support for subdomains of differing properties. IN0 does not provide "spatially–varying" properties, as MODFLOW does on a cell-by-cell basis — each sub-domain possesses constant properties within the sub-domain. Sub-domains are polygonal, and may have common boundaries. In ModAEM, inhomogeneity domains may be constructed in two ways:

polygonal boundaries (Figure ??) This is similar to the commercial code GFLOW (Haitjema Consulting) and the 2003 version of EPA WhAEM for Windows. Each inhomogeneity must be bounded by a single closed contour. Inhomogeneities may be nested, but their boundaries may not overlap or intersect.

common boundary inhomogeneities (Figure ??) This is similar to the commercial codes SLAEM and MLAEM (Strack Consulting). In addition to simple closed polygons, inhomogeneity domains may share edges. This greatly complicates preprocessing, but is very useful, especially for problems with a varying aquifer base elevation.

The following properties may be changed in ModAEM inhomogeneities:

- Hydraulic conductivity, K
- Aquifer base elevation, b

Figure 4.1: Potentiometric countours for two polygonal hydraulic conductivity inhomogeneities in a field of uniform flow. Flow is from left to right (red contours represent higher potentiometric heads). The domains are specified as follows: domain 0 (the outside) has $K = 10\,m/d$; domain 1 has $K = 100\,m/d$ and is shaded in blue; domain #2 has $K = 2\,m/d$ and is shaded in beige. No string specifications (`str` directive) were required.

Figure 4.2: Potentiometric countours for two polygonal hydraulic conductivity inhomogeneities with a common boundary in a field of uniform flow. Flow from left to right (red contours represent higher heads). The domains are specified as follows: domain 0 (the outside) has $K = 10\,m/d$; domain 1 has $K = 100\,m/d$ and is shaded in blue; domain #2 has $K = 2\,m/d$ and is shaded in beige. Three strings are required; all share the same starting and ending points. The three strings are specified as follows: string 1 (red) has $left = 0$, $right = 1$; string 2 (yellow) has $left = 1$; $right = 2$; string 3 (purple) has $left = 2$, $right = 0$.

- Aquifer thickness, H
- Aquifer effective porosity, n

In the case where base elevation and aquifer thickness are constant throughout the model domain, or when the flow is confined everywhere in the model, the equations solved by ModAEM for inhomogeneities are linear; only a few iterations are needed to achieve an accurate solution. In the case where the base elevation or aquifer thickness varies *and* there are regions within the model where the flow is unconfined, the solution matrix contains coefficients that are dependent on the saturated thickness; additional iterations are required to achieve an accurate solution. For this reason, each sub-domain in the model has an additional parameter, the "initial average head". The initial average head is the modeler's best estimate for the average head in the sub-domain. It is used to compute the saturated thickness during the first model iteration. Obviously, a good estimate for the initial average head helps the model stability.

4.4.1 Specifying inhomogeneities

As mentioned above, ModAEM allows for both closed polygonal inhomogeneities that do not share edges and for complex polygonal inhomogeneities that may have shared edges. ModAEM requires that the specification of inhomogeneities be done in two steps:

1. Defining the polygonal regions using the dom directive
2. Where necessary, defining the strings of elements that implement the boundaries using the str directive.

Task (2) can be very complicated. Fortunately there are preprocessing tools such as GMS (EMS-I) and ArcInfo that provide an "arc–node" data representation to make it simpler. For many problems, inhomogeneities with common boundaries are unnecessary. Fortunately, ModAEM makes it easier to specify these (see below).

Usage:

```
aqu ndomains nstrings base ...
  in0
    dom nvertices bot thick hyd-cond poro avg-head id
      (x1,y1)
      (x2,y2)
      ...
    dom ...
      ...
    str nvertices left right id
      (x1,y1)
      (x2,y2)
      ...
    str ...
      ...
  end
end
```

As noted above, the maximum number of domains and strings to be used are specified in the aqu directive that begins the aquifer specification. The specification of inhomogeneities begins with the in0 directive, which enters the inhomogeneity specification section of the ModAEM script file. Inside the IN0 section of the script, domains are specified first, then strings are defined that actually implement the boundaries[3].

Parameters for domain specification (directive dom)

`nvertices` **(int)** The maximum number of vertices which may be used to delineate the boundary of the domain. [−]

`base` **(real)** The base elevation of the aquifer [L]

`thickness` **(real)** The thickness of the aquifer [L]

`conductivity` **(real)** The hydraulic conductivity (real) of the aquifer [L/T]

`porosity` **(real)** Porosity of the aquifer as a fraction [−]

`initial-avg-head` **(real)** This is an estimate of the average head in the aquifer. It is used to pre–condition the solution when inhomogeneities in base elevation or thickness are encountered and the flow condition is unconfined. If you expect to be using inhomogeneities in base elevation or aquifer thickness, a good estimate of the initial average head will greatly speed the solution process. If you are not using inhomogeneities in base elevation or aquifer thickness, it is recommended that the value of `base` + `thickness` be provided. [L] {*real*} As usual, flow may be confined and/or unconfined within the domain. If the flow is unconfined, additional iterations may be required for an accurate solution.

`id` **(int)** A unique identification number for the domain.

Specifying the vertices along the domain perimeter Following the `dom` directive, up to `nvertices` vertices may be provided, one vertex per line in the input file. Each vertex is a single complex coordinate. Each vertex is entered as a single complex coordinate, $(\mathtt{x_p}, \mathtt{y_p})$, where p is the index of the vertex along the perimeter, $z_p = x_p + y_p$ [L]

[3] For problems without common boundaries, no string definitions are required; ModAEM automatically builds strings for closed domains.

Parameters for string specification (directive `str`) As mentioned above, ModAEM makes things easier for modelers who build input files by hand or from shapefiles that do not support an "arc/node" spatial data structure. If no common boundaries are to be specified, e.g. as in Figure **??**, the `str` directive should be omitted. ModAEM will automatically build closed strings of elements. The following describes the input format for inhomogeneity strings (please refer to Figure **??**).

`nvertices` **(int)** The maximum number of vertices which may be used to delineate the string. [−]

`left` **(int)** The ID number of the polygon that is to the left of the string (see section **??**). [−]

`right` **(int)** The ID number of the polygon that is to the left of the string (see section **??**). [−]

`closed` **(logical)** Flag; if true, the string closes on itself, making a polygon. If false, the string is an open polyline. [−]

`id` **(int)** A unique identification number for the string. [−]

Once the aquifer is defined using module AQU (see Chapter **??**), additional boundary conditions are added to the model by superposition. This chapter describes the elements that are available in ModAEM. Currently, the following element modules are provided:

Discharge–specified wells (WL0) These are wells for which the pumping rate is known, for example, water supply wells or irrigation wells. This is the most commonly–used type of well; both pumping wells and injection wells are supported. These are analougous to wells created with the MODFLOW WEL package.

Head–specified wells (WL1) These are wells for which the pumping rate is not *a priori* known to the modeler. The best example is a dewatering well that turns on and off according to a water level measurement. For WL1 elements, the modeler provides the location and radius of the well, plus a location in the aquifer and a head value; ModAEM computes the pumping rate of the well[4]. These are somewhat analogous to MODFLOW constant–head cells.

Discharge–specified line sinks (LS0) These are line segments that add or remove a specific amount of water along their length, where the modeler provides the amount of water to be added. Some examples of discharge–specified line sinks are infiltration galleries or rivers in arid climates that are typically dry, but infiltrate water at certain times of the year. These are analogous to a group of wells created with the MODFLOW WEL package.

Head–specified line sinks (LS1) These are line segments that add or remove water along their lengths, but for which the pumping rate is not *a priori* known to the modeler, and where there is no “entry resistance”, e.g a silty stream bed, between the line sink and the aa group ofquifer. LS1 line sinks are often used to represent rivers in the far field when the modeler wishes to use an unbounded aquifer with a modeled far field (see Section **??**). These are somewhat analogous to a group of MODFLOW constant–head cells; they differ because in MODFLOW, constant head cells are specified as part of the BAS package, not as a separate component.

[4] As will be discussed in Section **??**, WL1 wells should not be confused with calibration targets or inverse models.

Resistance line sinks (LS2) These are line segments that add or remove water along their lengths, but for which the pumping rate is not *a priori* known to the modeler, and where an "entry resistance", e.g a silty stream bed, between the line sink and the aquifer is present. LS2 line sinks are often used to represent surface waters or drains in the near field. LS2 line sinks may be created as rivers (analogous to the MODFLOW RIV package), drains (analogous to the MODFLOW DRN package), or general–head boundaries (analogous to the MODFLOW GHB package). Streamflow routing may be performed for LS2 elements (in a manner similar to the MODFLOW STR package), using the analysis module RT0 (see Section **??**).

Horizontal no–flow boundaries (HB0) These are elements that create a linear no–flow condition within the active area of a ModAEM model. These may be used to model sheet pilings, slurry walls, faults, and other linear no–flow conditions that require an active aquifer domain on both sides of the line. This should not be used for bounded models or for "islands" in an aquifer domain; use the BDY elements included in module AQU (Section **??**) for details. These are analougous to the MODFLOW HFB package.

Circular area–sinks (PD0) These are elements that provide an areal infiltration or exfiltration rate over a circular sub–domain, using a circular "pond" function described by Strack (1989). The "sink density", or rate of infitration per unit of surface area, is specified by the modeler. These are typically used in example problems and for conceptual models, although they are convenient for representing circular irrigators. For most practical modeling applications, module PD0 is superseded by the polygonal area–sink module AS0 (Section **??**).

Polygonal area–sinks (AS0) These are elements that provide an areal infiltration or exfiltration rate over a polygonal sub–domain. The "sink density", or rate of infitration per unit of surface area, is specified by the modeler. These are typically used as sources of areal recharge, e.g. from rainfall, or for infiltration galleries.

4.5 Discharge-specified wells (module WL0)

Module WL0 creates discharge-specified wells. ModAEM processing enters module WL0 in response to the wl0 directive.

Usage:

```
wl0 nwells
  (xw,yw) pumping-rate radius id
    # Optional features follow if desired
     ppw ...
     eff ...
     dhd ...
     ddn ...
  (xw, yw) ...
  ...
end
```

Parameters for the wl0 directive

`nwells` **(int)** The maximum number of wells in the problem [−]

Specifying well elements The wl0 directive is followed by one record for every well in the model. If more than nwells well elements is provided, an ModAEM will terminate and report the error. Each well record has the following form:

```
(xw,yw) pumping-rate radius id
```

Parameters

`(xw,yw)` **(complex)** The coordinates of the center of the well. [L]

`pumping-rate` **(real)** The pumping rate of the well. The value is positive if the element removes water from the aquifer, negative if it adds water to the aquifer [5]. [L^3/T]

[5] Users who make use of ModAEM using the GMS preprocessor will note that GMS makes use of the MODFLOW convention that abstraction of water from the aquifer is negative, while injection is positive. Conversion to ModAEM's convention is handled transparently by GMS.

`radius` **(real)** The radius of the well. [L]

`id` **(int)** A unique identification number for the well. [−]

Optional wl0 features ModAEM offers several optional features for modeling discharge-specified wells. Two

4.6 Head-specified wells (module WL1)

Module WL1 creates head-specified wells. Head-specified wells should be used in cases where a well is to be used to maintain a particular water level at some location in the aquifer. The model computes the pumping rate for each head–specified well element that allows the model to match the specified head; these elements *must not* be used for the purpose of calibrating the model to known heads. ModAEM processing enters module WL1 in response to the wl1 directive.

Usage:

```
wl1 nwells
  (xw,yw) head (xc,yc) radius id
  ...
end
```

Parameters `nwells` (int) The maximum number of wells in the problem [−]

4.6.1 Specifying head–specified well elements

The wl1 directive is followed by one record for every well in the model. If more than nwells well elements is provided, an ModAEM will terminate and report the error. Each well record has the following form:

```
(xw,yw) head (xc,yc) radius id
```

Parameters

`(xw,yw)` **(complex)** The coordinates of the center of the well. [L]

`head` **(real)** The specified head at the control point. [L]

`(xc,yc)` **(complex)** The coordinates of the point where the head condition is to be met. [L]

`radius` **(real)** The radius of the well. [L]

`id` **(integer)** A unique (integer) identification number for the well. [−]

4.7 Discharge-specified line-sinks (module LS0)

Module LS0 creates discharge-specified line-sinks. These are line segments that add or remove a specific amount of water along their length, where the modeler provides the amount of water to be added. Some examples of discharge–specified line sinks are infiltration galleries or rivers in arid climates that are typically dry, but infiltrate water at certain times of the year. These are analogous to a group of wells created with the MODFLOW WEL package.

Usage:

```
ls0 nstrings
  str npts id
    (x,y) strength
    ...
  str ...
    ...
end
```

Parameters for the ls0 directive

`nstrings` **(integer)** The maximum number of discharge-specified line-sink strings in the problem. [−]

Following the ls0 directive, the user provides strings of line sinks using the str directive. A string of line sinks is composed of a list of vertices. One line sink element is created for each pair of consecutive vertices.

Creating a string of elements

```
str npts id
  (x,y) strength
  ...
```

Parameters

`npts` **(integer)** The maximum number of vertices in the line-sink string. [−]

`id` **(integer)** A unique identification number for the string. [−]

Specifying vertices for line–sink strings Following the `str` directive, two or more data records define the vertices of the line–sink string. The parameters provided for each vertex are as follows.

```
(x,y) strength
```

`(x,y)` **(complex)** Coordinates of the vertex. [L]

`strength` **(real)** The sink density σ of the line–sink string at the vertex. The total volumetric infiltration or abstraction along the line will be the value $(\sigma_1+\sigma_2)\frac{L}{2}$ $[L^3/T]$ where σ_1 and σ_2 are the strengths at consecutive vertices and L is the distance between the vertices. The sink density is defined as the total extraction rate of the line-sink per unit length. The value is positive if the element removes water from the aquifer, negative if it adds water to the aquifer [6]. [L^2/T]

[6] Users who make use of ModAEM using the GMS preprocessor will note that GMS makes use of the MODFLOW convention that abstraction of water from the aquifer is negative, while injection is positive. Conversion to ModAEM's convention is handled transparently by GMS.

4.8 Head-specified line-sinks (module **LS1**)

Module LS1 creates head–specified line–sink elements. These are line segments that add or remove water along their lengths, but for which the pumping rate is not *a priori* known to the modeler, and where there is no "entry resistance", e.g a silty stream bed, between the line sink and the aquifer. LS1 line sinks are often used to represent rivers in the far field when the modeler wishes to use an unbounded aquifer with a modeled far field (see Section **??**). These elements are somewhat analogous to a group of MODFLOW constant–head cells; they differ because in MODFLOW, constant head cells are specified as part of the aquifer, not as a separate element.

Usage:

```
ls1 nstrings
  str npts id
    (x,y) head
      ...
  str ...
      ...
end
```

Parameters for the **ls1** directive

`nstrings` (integer) The maximum number of discharge-specified line–sink strings in the problem. [−]

Following the ls1 directive, the user provides strings of line sinks using the str directive. A string of line sinks is composed of a list of vertices. One line sink element is created for each pair of consecutive vertices.

Creating a string of elements

```
str npts id
  (x,y) head
  ...
```

Parameters for the `str` directive

`npts` **(integer)** The maximum number of vertices in the line-sink string. [−]

`id` **(integer)** A unique identification number for the string. [−]

Specifying vertices for line–sink strings Following the `str` directive, two or more data records define the vertices of the line–sink string. The parameters provided for each vertex are as follows.

`(x,y)` **(complex)** Coordinates of the vertex. [L]

`head` **(real)** The specified head at the center of the line-sink string. [L]

4.9 Line-sinks with entry resistance (module LS2)

Module LS2 creates head-specified line-sinks. These are line segments that add or remove water along their lengths, but for which the pumping rate is not *a priori* known to the modeler, and where an "entry resistance", e.g a silty stream bed, between the line sink and the aquifer is present. LS2 line sinks are often used to represent surface waters or drains in the near field. LS2 line sinks may be created as rivers (analogous to the MODFLOW RIV package), drains (analogous to the MODFLOW DRN package), or general–head boundaries (analogous to the MODFLOW GHB package). Streamflow routing may be performed for LS2 elements (in a manner similar to the MODFLOW STR package), using the analysis module RT0 (see Section **??**).

Usage:

```
ls2 nstrings
  str npts mode conductance id
    (x,y) head bottom
    ...
  str ...
    ...
end
```

Parameters for the ls2 directive

`nstrings` **(integer)** The maximum number of discharge-specified line–sink strings in the problem. [−]

Following the ls2 directive, the user provides strings of line sinks using the str directive. A string of line sinks is composed of a list of vertices. One line sink element is created for each pair of consecutive vertices.

Creating a string of elements

```
str npts mode conductance id
  (x,y) head bottom
  ...
```

Parameters for the str directive

`npts` **(integer)** The maximum number of vertices in the line-sink string. [−]

`id` **(integer)** A unique identification number for the string. [−]

`mode` **(integer)** Defines the behavior of the line sink. The value provided is an integer from the list below:

> `general-head boundary (0)` The boundary is always active, in a manner analogous with the MODFLOW GHB package.
>
> `river (1)` The line sink becomes a discharge–specified feature if the head in the aquifer falls below the bottom of the resistance layer (a "percolating" condition). In this case, the infiltration density is computed as $\sigma = c \times (h_r - h_b)$ where c is the conductance of the line sink (see below), h_r is the specified stage in the river, and $h_b$is the elevation of the bottom of the resistance layer for this line sink. This is analogous to the MODFLOW RIV package.
>
> `drain (2)` The line sink will be removed from the solution with a sink density of zero when the head in the aquifer falls below the bottom of the drain. This is analogous to the MODFLOW DRN package.

`conductance` **(real)** The conductance for the line sink. The conductance is defined in a manner consistent with MODFLOW; for a river, the conductance is defined to be $c = w \times \frac{k_c}{t_c}$ $[L/T]$, where c is the conductance, w is the width of the stream, k_c is the vertical hydraulic conductivity of the resistance layer, and t_c is the thickness of the resistance layer. [L/T]

`id` **(integer)** A unique identification number for this string. [−]

Specifying vertices for line–sink strings Following the `str` directive, two or more data records define the vertices of the line–sink string. The parameters provided for each vertex are as follows.

`(x,y)` **(complex)** Coordinates of the vertex. [L]

`head` **(real)** The specified head at this vertex. The model interpolates the head along the line segment. Since the head is specified at the center of the line segment, the average of the bottom elevation between adjacent vertices is used. [L] `bottom` **()** real The elevation of the bottom of the resistance layer at this vertex. The model interpolates the bottom elevation along the line segment. Since the head is specified at the center of the line segment, the average of the bottom elevation between adjacent vertices is used. [L]

4.10 No–flow boundary walls (module HB0)

Module HB0 creates no–flow boundary walls (e.g. sheet pilings or slurry walls). These are elements that create a linear no–flow condition within the active area of a ModAEM model. These may be used to model sheet pilings, slurry walls, faults, and other linear no–flow conditions that require an active aquifer domain on both sides of the line. This should not be used for bounded models or for "islands" in an aquifer domain; use the BDY elements included in module AQU (Section **??**) for details. These are analogous to the MODFLOW HFB package.

Usage:

```
hb0 nstrings
  str npts id
    (x,y)
    ...
  str ...
    ...
end
```

Parameters for the hb0 directive

`nstrings` **(integer)** The maximum number of no–flow strings in the problem. [−]

Following the `hb0` directive, the user provides strings of line sinks using the `str` directive. A no–flow string is composed of a list of vertices. One line doublet element is created for each pair of consecutive vertices.

Creating a string of elements

```
str npts id
  (x,y) strength
  ...
```

Parameters for the str directive

`npts` **(integer)** The maximum number of vertices in the line-sink string. [−]

`id` **(integer)** A unique identification number for the string. [−]

Specifying vertices for line–sink strings Following the `str` directive, two or more data records define the vertices of the no–flow string. The parameters provided for each vertex are as follows.

`(x,y)` **(complex)** Coordinates of the vertex. [L]

4.11 Discharge-specified circular area-sinks (module PD0)

Module PD0 creates discharge-specified circular area-sinks (colloquially known as ponds).

Usage:

```
pd0 nponds
  (xc,yc) strength radius id
  ...
end
```

Parameters for the pd0 directive

`nwells` The maximum number of wells in the problem

Specifying pond elements The `wl0` directive is followed by one record for every pond in the model. If more than `nponds` elements are provided, ModAEM will terminate and report the error. Each pond record has the following parameters:

```
(xc,yc) sink-density radius id
```

`(xc,yc)` **(complex)** The coordinates of the center of the pond. [L]

`sink-density` **(real)** The sink density of the pond. This is the value $\gamma = Q_p/A_p$, where $\gamma\,[L/T]$ is the sink density, $Q_p\,[L^3/T]$ is the total amount of water infiltrated or abstracted by the pond, and $A_p\,[L^2]$ is the area of the pond. The value is positive if the element removes water from the aquifer, negative if it adds water to the aquifer [7] [L/T]

`radius` **(real)** The radius of the pond. [L]

`id` **(real)** A unique identification number for the pond. [−]

[7] Users who make use of ModAEM using the GMS preprocessor will note that GMS makes use of the MODFLOW convention that abstraction of water from the aquifer is negative, while injection is positive. Conversion to ModAEM's convention is handled transparently by GMS.

4.12 Polygonal area-sinks (module AS0)

Module AS0 creates discharge-specified polygonal area–sink elements. These are elements that provide an areal infiltration or exfiltration rate over a polygonal sub–domain. The "sink density", or rate of infiltration per unit of surface area, is specified by the modeler. These are typically used as sources of areal recharge, e.g. from rainfall, or for infiltration galleries.

Usage:

```
as0 top/bottom nareas
  str npts strength id
    (x,y)
    ...
  str ...
    ...
end
```

Please note: In versions of ModAEM prior to version 1.8, the "bottom" and "nareas" parameters were reversed in the AS0 input.

Parameters for the as0 directive

`top/bottom` **(integer)** An integer that specifies whether the area sinks are to be placed at the aquifer top (`0`) or bottom (`1`). [−]

`nareas` **(integer)** The maximum number of discharge–specified area–sink polygons in the problem. [−]

Following the `as0` directive, the user provides polygons using the `str` directive. A polygon is composed of a list of vertices. It is not necessary to duplicate the first vertex to close the polygon; ModAEM automatically closes the polygon.

Creating an element

```
str npts sink-density id
  (x,y)
  ...
```

Parameters for the str directive

`npts` **(integer)** The maximum number of vertices in the line-sink string. $[-]$

`sink-density` The sink density of the area sink. This is the value $\gamma = Q_p/A_p$, where $\gamma\,[L/T]$ is the sink density, $Q_p\,[L^3/T]$ is the total amount of water infiltrated or abstracted by the element, and $A_p\,[L^2]$ is the area of the polygon. The value is positive if the element removes water from the aquifer, negative if it adds water to the aquifer. [8] [L/T]

`id` **(integer)** A unique identification number for the string. $[-]$

Specifying vertices for polygons Following the `str` directive, three or more data records define the vertices of the area–sink perimeter. The parameters provided for each vertex are as follows.

`(x,y)` **(complex)** Coordinates of the vertex. [L]

[8] Users who make use of ModAEM using the GMS preprocessor will note that GMS makes use of the MODFLOW convention that abstraction of water from the aquifer is negative, while injection is positive. Conversion to ModAEM's convention is handled transparently by GMS.

Chapter 5

Processing Directives

This chapter describes processing directives that control the solution process and general imspection of values for ModAEM. Many of the directives described here are most commonly used for program debugging, experimentation and development of example problems. When computing results that are to be used for analysis purposes (e.g. within GUI–based modeling environments), the modules GRI (grid/contour generation), EXT (data extraction), INQ (element results inquiry), and OBS (observation data)[1]. See Chapter **??** for details about the analysis modules.

Each of these directives is given outside the "AEM section" of the ModAEM script. The general layout of the script is as follows, with the processing directives marked in bold face.

```
aem
  aqu...
    ref...
    bdy...
    in0...
      ...
    end
  end
  # Other elements go here
end
\textbf{\#processingdirectivesgohere}
\textbf{sol...}
```

[1] Available in ModAEM-1.4.1 and later versions.

```
    \textbf{hea...}
    eod
```

In nearly every case, the first processing directive issued should be `sol x`, where `x` is the number of iterations (if a solution is to be loaded, use `sol 0` to simply load the saved results file). After the solution is complete, the other directives will be useful.

Processing directives for solving and reporting solution results

`SOL` Solves the model, based on the available input and results from a previous solution loaded from disk[2].

`RPT` Generates a report of all solution information in HTML format.

Processing directives that retrieve data from the model at specific points

`HEA` Reports the head at some location in the model to the error log file.

`POT` Reports the complex potential at some location in the model to the error log file.

`DIS` Reports the discharge at some location in the model to the error log file.

`VEL` Reports the velocity at some location in the model to the error log file.

`RCH` Reports the net exfiltration rate at some location in the model to the error log file.

`THK` Reports the saturated thickness at some location in the model to the error log file.

Processing directives that compute numerical approximations for model testing

`GRA` Reports the numerical gradient at some location in the model to the error log file.

`LAP` Reports the numerical laplacian at some location in the model to the error log file.

[2] Available in ModAEM-1.4.1 and later versions.

Processing directives that compute values for a line segment

`FLO` Reports the integrated flow across a line segment in the model to the error log file.

The remainder of this chapter describes the general–purpose processing directives in detail.

5.1 Directive SOL – solve the model

After a model has been defined (using the AEM module input section), it must be solved prior to performing any analyses. ModAEM uses an iterative solution scheme – at each iteration, the solution is improved based on the previous iteration, including the incorporation of non–linear elements such as resistance line sinks or inhomogeneity boundaries in regions where the flow is unconfined.

Usage:

```
sol niter relaxation
```

Parameters for the sol directive:

niter The number of iterations to be performed. {*integer*}
The number of iterations to be used depends strongly on the problem to be solved. The following list describes the issues that determine the number of iterations needed for model convergence. Note that this list is a rule–of–thumb; the modeler should look closely at the solution to ensure that it has fully converged.

1–2iterations Simple models that are linear at all points in the domain. Typically, this means that the flow is confined everywhere and that no resistance line sinks that can be removed from solution (that is, "river" and "drain" line sinks) are present. These problems should give accurate solutions with only 1–2 iterations.

3–8iterations More complex models that make use of "river" and "drain" line sinks, or have large areas in the model domain in which the flow is unconfined, but where the aquifer base elevation and thickness are constant everywhere will typically converge in 8 iterations or less.

moreiterations Very complex models in which the aquifer base elevation and/or thickness varies and the flow is unconfined, or problems where baseflow routing is in use. These problems may require 10 or more i terations to achieve convergence.

relaxation The relaxation factor to be used. {*integer*}

This parameter instructs ModAEM to relax the solution during iterations. If *relaxation* $= 1$, then ModAEM applies all of the computed adjustments in the strength coefficients on each iteration. For $0 < relaxation < 1$, the value of *relaxation* is multiplied by the strenght adjustments. In some very complex models, this reduces the "stiffness" of the solution algorithm and may reduce the possibility of oscillations during iterations. In nearly all cases, this parameter should be set to 1.0.

5.1.1 Loading a previous solution

In ModAEM-1.8 and later versions, the modeler has the option of loading a previous solution. This is an advantage for large complex models; the previous solution may be loaded prior to post-processing and analysis, e.g. particle tracking or computing grids of heads. The problem definition, i.e. the aquifer definition and creation of all elements must be complete prior to loading the results.

Because the previous solution is loaded in response to the `sol` directive a special version of the directive, `sol 0`, is provided. By issuing the `sol` directive with no iterations, the previous solution will be loaded, and all of ModAEM's internal data structures will be restored, but no addition solution step will be performed. Whenever a previous solution is to be used, the `sol 0` directive should be the first directive in the ModAEM script after the AEM section is complete.

5.1.2 Saving a solution for future re–use

If the modeler provides a name for a "solution save file" on line 3 of the ModAEM name file, ModAEM stores the solution there as the final step in the "post–solve" procedure. No directives are required in the ModAEM script file.

5.2 Directive RPT – report the solution in HTML format

After the solution is complete (see directive `SOL` above),

5.3 Directives that compute analytic values at a single point

The following procesing directives compute a value and report it in a human–readable format to the run–time message file.

5.3.1 HEA - report the modeled head

Reports the potentiometric head at a specified point to the message file and to the console. A solution must be present (see the `sol` directive) prior to issuing this directive.

Usage:

```
hea(x,y)
```

Parameters for the hea directive

`(x,y)` The desired location in the complex plane $x + iy$. $\{complex\}$

Example: The head at the location $(100, 100)$ is reported in response to the following directive:

```
HEA(100.0,100.0)
```

5.3.2 POT - report the modeled complex potential

Reports the complex potential at a specified point to the message file and to the console. The complex potential is defined as $\Omega = \Phi + i\Psi$, where Φ is the discharge potential and Ψ is the streamfunction[3]. A solution must be present (see the `sol` directive) prior to issuing this directive.

Usage:

```
pot(x,y)
```

[3] Note that the streamfunction does not exist for problems that include recharge; the reported streamfunction value is not useful in regions where a recharge element (see module AS0 or PD0) is present.

Parameters for the pot directive

`(x,y)` The desired location in the complex plane $x + iy$. $\{complex\}$

Example The complex potential at the location $(100, 100)$ is reported in response to the following directive:

```
pot(100.0,100.0)
```

5.3.3 DIS - report the total aquifer discharge

Reports the total aquifer discharge $Q_x + iQ_y$ at a specified point to the message file and to the console. The total discharge is a two-dimensional analogue for the specific discharge:

$$Q_i = Q_x + iQ_y = \int_{z_{bot}}^{z_{top}} (q_x + iq_y) dz$$

where $q_x(z)$ and $q_y(z)$ are the horizontal components of the specific discharge $q_i = -k\partial_i H$ at the elevation z, and z_{top} and z_{bot} are the elevations of the aquifer top and bottom, respectively[4]. A solution must be present (see the `sol` directive) prior to issing this directive.

Usage:

```
dis(x,y)
```

Parameters for the dis directive

`(x,y)` The desired location in the complex plane $x + iy$. $\{complex\}$

Example The complex discharge at the location $(100, 100)$ is reported in response to the following directive:

```
dis(100.0,100.0)
```

[4] Since ModAEM is a 2–D code, it does not explicitly compute the integral. The value is computed analytically by differentiating the discharge potential Φ at the coordinate $x + iy$. See e.g. Strack (1989) or Haitjema (1995) for details. A detailed description of the formulation of ModAEM will be included in a future version of this book.

5.3.4 VEL - report the horizontal groundwater velocity

Reports the horizontal aquifer velocity $v_x + iv_y$ at a specified point to the message file and to the console. The velocity should be interpreted as the *vertically–averaged* velocity at the point in question:

$$\bar{v}_i = \bar{v}_x + i\bar{v}_y = \frac{Q_i}{hn_e}$$

where $\bar{v}_x(z)$ and $\bar{v}_y$ are the reported horizontal components of the average velocity, h is the saturated thickness, and n_e is the effective porosity. A solution must be present (see the `sol` directive) prior to issing this directive.

Usage:

```
vel(x,y)
```

Parameters for the vel directive

`(x,y)` The desired location in the complex plane $x + iy$. $\{complex\}$

Example The average groundwater velocity at the location $(100, 100)$ is reported in response to the following directive:

```
vel(100.0,100.0)
```

5.3.5 RCH - report the net recharge rate

Reports the net rate of areal recharge at a specified point to the message file and to the console. The reported recharge rate has a sign consistent with the ModAEM conventions as described in e.g., Sections**??** and **??**. The reported rate is negative if the net recharge rate injects water into the aquifer, positive if water is removed. A solution must be present (see the `sol` directive) prior to issuing this directive.

Usage:

```
rch(x,y)
```

Parameters for the rch directive

(x,y) The desired location in the complex plane $x + iy$. {*complex*}

Example The net recharge rate (negative for recharge) at the location $(100, 100)$ is reported in response to the following directive:

```
rch(100.0,100.0)
```

5.4 Directives that compute numerical approximations for testing

Two directives are provided that approximately compute the gradient in the potential and laplacian of the potential, for the purpose of testing the analytic functions that underlie all ModAEM computations.

5.4.1 GRA - report the modeled numerical gradient in potential

Reports the *numerical* gradient in the discharge potential) at a specified point to the message file and console. This directive is commonly used in program debugging; the numerical gradient should have approximately the same value as the total discharge (see the dis directive in Section **??**). A solution must be present (see the sol directive) prior to issing this directive.

The aproximate gradient $(\hat{q}_x, \hat{q}_y)$ is computed at (x, y), a point in the 2D plane, using a spacing δ provided by the user. The calculations are performed according to the following expression.

$$\begin{aligned}\hat{q}_x &= \frac{1}{\delta}\left[\Phi(x+\delta, y) - \Phi(x-\delta, y)\right] \\ \hat{q}_y &= \frac{1}{\delta}\left[\Phi(x, y+\delta) - \Phi(x, y-\delta)\right]\end{aligned}$$

in general, it is expected that a more accurate estimate of the gradient is obtained by using a small value of δ.

Usage:

```
gra{[}z{]}delta
```

Reports the numerical gradient in the potential at the complex coordinate, $z = (x, y)$, using the spacing δ. Note that in Fortran free-format reads, the two parts of the complex coordinate are provided as (x, y) pairs.

Example: The numerical gradient at the coordinate $(100, 100)$, using a spacing of 1.0 is reported in response to the following directive:

```
gra(100.0,100.0)1.0
```

5.5 Directives that compute a net analytic value for a line segment

ModAEM provides a directive that is used to report model results along a line segment. Future versions of ModAEM may include additional directives for line segments.

5.5.1 FLO - report the total flow across a path

Directs ModAEM to report the total integrated groundwater flow across a linear path connecting two specified points to the message file (.err file) and console. The value is computed as

$$Q(z_1, z_2) = \int_{z_1}^{z_2} Q_n ds$$

where

$z_1 \mathbf{and} z_2$ are the points at the end-points of the line segment, $z_j = x_j + iy_j$

Q_n is the discharge normal to the line segment $z_1 z_2$

s is the direction along the line segment $z_1 z_2$

A solution must be present (see the `sol` directive) prior to issing this directive.

Usage:

```
flo(x1,y1)(x2,y2)
```

Reports the integrated groundwater flux across the line-segment connecting (x_1, y_1) and $(x_{2,}y_2)$ in units of L^3/T. The sign of the result is determined by the right-hand rule (Section **??**).

Example: The total integrated flux across the line segment containing $(50, 50)$ and $(100, 100)$ is reported in response to the directive

```
flo(50.0,50.0)(100.0,100.0)
```

5.6 Extracting model results in machine-readable format (module INQ)

Module INQ provides the ability to extract results from the model in a machine-readable format that is convenient for post-processing tools and graphical user interface (GUI) programs.

This section needs to be completed

Chapter 6

Analysis modules

6.1 Generating grids of model results (module GRI)??

Module GRI provides a facility for the construction of grids (e.g. for contour plotting) in a format compatible with SURFER$^{\text{TM}}$, Matlab$^{\text{TM}}$, or other software packages. Within the grid module, the window to be gridded must be specified, along with the number of points on the long axis of the grid.

5.7.1 Selecting the output file type (opt directive) The opt directive instructs the grid module which type of output file to create. Usage:

opt grid-type The opt directive expects one parameter, as follows. grid-type Choose surfer for an ASCII SURFER$^{\text{TM}}$-compatible grid (with the extension .grd) and matlab for

an ASCII MATLABTM-compatible grid (with the extension .m). If the OPT directive is omitted, the grid-type will default to surfer.

Example To select a MatlabTM-compatible output file, issue the directive

opt matlab

5.7.2 Defining the grid window (win directive) Defines the window to be gridded. Usage :

win (x1,y1) (x2,y2) The win directive expects the following parameters. (x1,y1) The lower-left corner of the (rectangular) region to be gridded. (x2,y2) The upper-right corner of the (rectangular) region to be gridded.

Example

win (-100.0,-100.0) (100.0,100.0)

sets the lower-left and upper-right corners of the window for the GRI module at the coordinates \Gamma \Delta \Gamma 100\Delta \Delta \Gamma 100\Theta and\Gamma

100\Delta 100\Theta , respectively.

5.7.3 Choosing the grid resolution (dim directive) Sets the number of grid points along the long axis of the window. Usage:

dim npts The dim directive expects one parameter as follows. npts The number of evenly-spaced points to compute along the long axis of the rectangular grid region. Module GRI

will compute an appropriate number of points along the short axis to ensure that the grid has regular spacing in both directions.

Example To make grid(s) with a resolution of 50 points along the long axis, issue the directive

gri 50

5.7.4 Computing a grid and writing it to a file (directives hea, pot, psi, q_x, and q_y) Once the grid type, grid region, and grid resolution are specified (see the directives opt, dim, and win above), grid files may be computed for a variety of model output values. Currently the following directives are available:

HEA - Create a grid of heads Usage:

hea base-filename generates a grid of the potentiometric head f on the file base-filename_head.grd (or base-filename_head.m)

POT - Create a grid of potentials Usage:

pot base-filename generates a grid of the discharge potential \Phi on the file base-filename_potential.grd (or base-filename_potential.m)

PSI - Create a grid of stream functions Usage:

psi base-filename generates a grid of the streamfunction \Psi on the file base-filename_psi.grd (or base-filename_psi.m)

Q_X - Create a grid of discharges in the x-direction Usage:

q_x base-filename generates a grid of the total aquifer discharge in the x-direction potentiometric heads on the file base-filename_qx.grd (or base-filename_qx.m)

Q_Y - Create a grid of discharges in the y-direction Usage:

q_y base-filename generates a grid of the total aquifer discharge in the y-direction on the file base-filename_qy.grd (or base-filename_qy.m)

6.2 Streamline tracing (module TR0)

5.8 Module TR0 - Trace The TR0 directive instructs ModAEM to enter the trace module, which is used to trace 2-D streamlines. The TR0 directive must have a matching END directive. Within the TR0 module, the following directives are valid:

WIN - Set the tracing window. Default tuning parameters are derived from the window size.

TUN - Set tuning parameters Sets tuning parameters for the tracing algorithm. Usage:

TUN step prox frac small step The base step size prox The proximity (in terms of the current step size) to boundary conditions for reducing the step size frac The factor for step size reductions small Smallest allowable step size

TIM - Specify maximum time allowed for particle tracing POI - Release a single particle at the specified location LIN - Release particles along a line N particles along a line

GRI - Release a grid of particles in the sub-window WL0 - Release N particles in reverse from the well bore of a WL0 (discharge-specified well) element. WL1 - Release N particles in reverse from the well bore of a WL1 (head-specified well) element.

Chapter 7

Validation

This chapter will describe the validation problems that ship with the official ModAEM release.

Chapter 8

ModAEM Tools

This chapter describes various preprocessing and postprocessing tools that are either included in the ModAEM distribution or are available from third-party vendors.

Appendix A

Mathematical formulation of ModAEM elements

Ugh...