# The ModAEM-1.8 User's Guide

**Vic Kelson, WHPA Inc.**

**March, 2007**

# 1 Introduction

This manual describes the results of nearly ten years of effort (much of it the "on again, off again" variety). ModAEM has been through three 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.

## 1.1 History

This is an approximate chronology of the major events in the development of ModAEM:

### 1.1.1 Beginnings

Like every graduate student, I began this project thinking that I could quickly write a code that did what my advisor's code did. I learned about analytic elements from Dr. Henk Haitjema at Indiana University. Henk was a student of Otto Strack, and wrote the first AE code, SYLENS. Otto later developed the more comprehensive codes SLAEM, SLAEMS, MLAEM, MVAEM and more codes of which I am unaware. SLAEM and its children have always been proprietary codes. Henk needed a research and teaching code, and he developed the code GFLOW in the late 1980's and early 1990's. GFLOW is a popular commercial code, and recently, Henk began to make source code available for it.

I first started hacking around with analytic element codes in 1992–1993, when I began working on my Ph.D. My first effort was an "integrated flux" boundary

condition that persists into ModAEM (and GFLOW) today. It was implemented in SLWL, which was (and for many is still) the lingua franca for beginning analytic element programmers.

I spent more of my time working on preprocessing tools for analytic elements and the use of analytic element models as screening tools for numerical models, and also on FRACFLOW, a model of flow in discrete fracture networks.

In 1994, I began work on what would become ModAEM. As part of a research project funded by U.S. EPA, I was offered the opportunity to develop and test an analytic element code that would run on massively–parallel computers. I began with a cobbled–up version of SLWL, but the old–style FORTRAN 77 nature of the code made parallelization on message–passing hardware difficult in those days (as it most likely would be now). After doing benchmarks with my mangled SLWL, it was clear that an improved design was needed

### 1.1.2 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 upon advice from 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 better choice. ModAEM is still parallelizable, although no one I know has tried it on SMP hardware yet.

The earliest versions of ModAEM were experimental codes for testing the parallel algorithms. It wasn't a real modeling tool until

### 1.1.3 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.

### 1.1.4 The SFWMD years

In 1998, I finished my Ph.D. and began working for the South Florida Water Management District in West Palm Beach, FL. I did some development work on

ModAEM, typically in the form 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 my my grad–school colleague, Jack Wittman.

### 1.1.5 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!

### 1.1.6 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.

### 1.1.7 ModAEM-1.6 and ModAEM-1.8

Since 2004, development of ModAEM has progressed, with the addition of baseflow routing along surface water reaches, improved I/O (including HTML reporting), and enhanced tools for profile modeling with analytic elements. In addition, a variety of scripts in the Python language now provide advanced model development support for GIS users, including inverse modeling support.

### 1.1.8 What's Next?

After many years of work and a few false starts, ModAEM has become 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. For information about those codes, visit http://www.bakkerhydro.com.

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.

## 1.2 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 later. Everyone is welcome to the source code, but if you enhance it and distribute it, you must distribute the source code for the enhancements. I have always hoped that ModAEM could become a powerful starting point for other developers, and I hope to be taught by them!

**Keep it generic**

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.

**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!

## 1.3 How to read this manual

The following conventions are used for formatting in this manual:

`typewriter text` is used for sample input files, file names and other related items;

*italicized 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).

## 1.4 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, or “pairs” of real numbers. In this manual, the type and units for each input value is provided at the right margin in underlined bold text, e.g. **real [L]** for a real value that has units of length. When directed to provide a numeric value, use the following rules:

### 1.4.1 Integer values **int**

Positive or negative integer values are allowed. ModAEM uses the Fortran 90 `SELECTED_INT_KIND` function to specify 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. 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

### 1.4.2 Logical values **logical**

Logical values 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.

### 1.4.3 Real values **real**

Positive or negative floating–point values are allowed. ModAEM uses the Fortran 90 `SELECTED_REAL_KIND` function 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. Note that this has not been tested; please report success or failure. 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. 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

### 1.4.4 Complex values (or pairs of real 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. Complex numbers may or may not contain a decimal point. If exponential notation is desired, use the characters $E \pm XX$ as a suffix. 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)

## 1.5 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.

# 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.

## 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

1.1.1 Discharge potentials

1.1.2 Example solutions

1.1.3 Analytic element functions and superposition

1.1.4 Boundary conditions for complex models

1.2 Using the AEM

1.2.1 Keep it simple

1.2.2 Stepwise modeling approach

1.2.3 Gotchas and troubleshooting

# 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.

### 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 found in a file called `modaem.aem`, then the contents of `modaem.nam` would be:

```
modaem
```

The `.aem` extension 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) are designed for this feature.

**modaem.out.html**

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.

## 3.2 The AEM script 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 *AEM* section of the input file is the portion contained between the aem and end directives in lines 1 and 14 of the above listing. Within the *AEM* section, input for the various element definition modules (see chapters 4 and 5) are provided.

The processing section follows the problem definition section of the script file. The various processing directives that are available are discussed in Chapter 6.

### 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. ModAEM ignores comment lines. Example:

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

**Continuation marker**

If an input line is too long to fit within the 256-character limitation, place the continuation marker \ at the end of the line to continue a directive. Example:

```
# The following line requires two lines of input
dom 10 10.0 0.0 100.0 0.2 \
    2000.0 100001 This is a label
```

**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
```

**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. 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 6.

# 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 porosity1, specified in "inhomogeneities". An inhomogeneity is a bounded subregion 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.

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

## 4.1 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 possesses optional submodules. The general layout for module *AQU* input is as follows:

```
# Create an aquifer
aqu ndomains nstrings base height hyd-cond porosity avg-head
  ref <arguments>
  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 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 ndomains nstrings base thickness hyd-cond porosity avg-head
```

Parameters for the *aqu* directive:

**ndomains** **integer [–]**

The number of inhomogeneity domains in the aquifer, including the "outside" unbounded domain (for a single homogeneous aquifer, use 1).

**nstrings** **integer [–]**

The number of strings that are used to bound the inhomogeneity domains (for a single inhomogeneous aquifer, use 0).

**base** **real [L]**

The base elevation of the aquifer.

**height** **real [L]**

The thickness of the aquifer. ModAEM automatically determines whether flow is confined or unconfined at any point, according to the value of the *discharge potential* (see e.g. Strack, 1989). For a model that is unconfined everywhere, set the thickness to a large value.

**hyd-cond** **real [L]**

The hydraulic conductivity of the aquifer. For a model that is based on *transmissivity*, set the base elevation to a very low value, e.g. -10000, the thickess to 1.0 and provide the transmissivity wherever the conductivity is requested.

**porosity** **real [–]**

Porosity of the aquifer as a fraction.

**avg-head** **real [L]**

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.

### 4.2.1 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. To create this aquifer, use the *aqu* module input below:

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

## 4.3 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 fact, it's difficult to justify calibration based on aquifer parameters when uniform flow is in use). In ModAEM, specification of the reference point is optional. If the reference point it 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, a reference head, and a far–field uniform flow discharge is required. 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 provides support for closed model domains via the *bdy* directive (see Section 4.4, 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 a 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. 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). If you are using a reference point, never place it 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*.

**Usage:**

```
ref (x0,y0) h0 (Qx0,Qy0)
```

Parameters for the *ref* directive:

**( x0, y0 )** **real [L, L]**

The location of the reference point, *(x, y)*

**head** **real [L, L]**

The head at the reference point.

**( Qx0, Qy0 )** **real [L, L]**

The reference flow field, as a vector $Q_0 = Q_{x0} + iQ_{y0}$. The magnitude of the value $Q_0$ is computed as $Q_0 = k \times H \times \frac{dh}{ds}$ where $k$ is the hydraulic conductivity, $H$ is the saturated thickness and $\frac{dh}{ds}$ is the hydraulic gradient in the direction of flow. The components are relative to the *x*-axis and *y*-axis; if $\theta$ is the angle of orientation of the discharge vector relative to the *x*-axis, the components are computed as $Q_{x0} = Q_0 \cos\theta$ and $Q_{y0} = Q_0 \sin\theta$.

Example of an aquifer with uniform flow and a reference point

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

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 m* at the origin.

## 4.4 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 by providing line segments along the model perimeter, each of which has a boundary condition associated with it.

For convenience when computing grids for contouring or for tracing pathlines, ModAEM determines the polygon that bounds the active area of the model and returns an "inactive region" value whenever an analysis module requests the head (or discharge or other value) at a point in the inactive region.

**Specifying conditions on the aquifer perimeter**

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.

Usage:

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

Parameters for the *bdy* directive

**nbdy** **integer [–]**

The maximum number of boundary elements in the problem.

### 4.4.1 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 1.5) is used. Fluxes are numerically positive if they cross the element from right–to–left. Note that this means that the polygonal outer boundary is "positively oriented" (points go counterclockwise).

Each segment is entered as follows:

```
(x1,y1) (x2,y2) flux head ghb-distance bdy-flag id label
```

**(x1, y1)** **complex [L, L]**

The first point on the line segment as an *(x, y)* pair.

**(x2, y2)** **complex [L, L]**

The second point on the line segment as an *(x, y)* pair.

**flux** **real [$L^3/T$]**

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 according to the right–hand rule (Section 1.5).

**head** **real [L]**

The specified head at the center of the segment

**ghb-distance** **real [L]**

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 aquifer subdomain on the left side of the element.

**bdy-flag** **integer [–]**

A flag that controls the type of boundary–condition along the segmen. Codes are **0** (head), **1** (flux), and **2** (general–head).

**id** **integer [–]**

This is a unique ID number for the segment. It allows the user to look up solution results for the segment in the output HTML file.

**label** **text**

An optional text label for the segment.

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. The ignored entries are available for use as check information, e.g. 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* directive is part of module *AQU*. No *end* directive is used to terminate these inputs (the *end* directive terminates input to module *AQU*). This is distinguished from the inhomogeneity sub–module *IN0*, discussed in Section 4.5.

## 4.5 The inhomogeneity sub–module (*IN0*)

The *IN0* sub–module provides the *AQU* aquifer module with support for sub–domains 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 subdomain. Sub–domains are polygonal, and may have common boundaries. In ModAEM, inhomogeneity domains may be constructed in two ways:

**Polygonal inhomogeneities**

This is similar to the pre–2005 versions of 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 (see Figure 1).

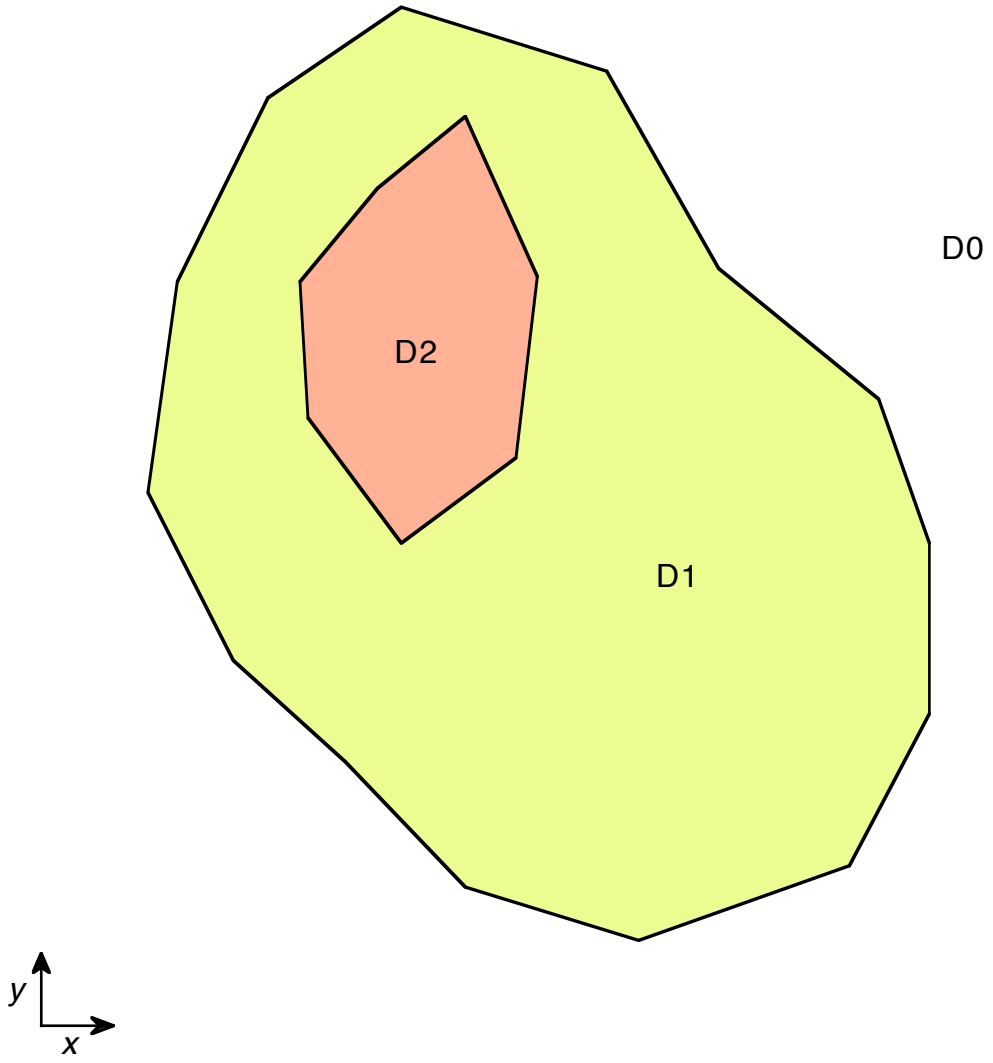


**Figure 1. Nested polygonal inhomogeneities. For each of the three domains, the hydraulic conductivity, aquifer base and thickness, and porosity may differ.**

**Common boundary inhomogeneities**

This is similar to the commercial codes SLAEM and MLAEM (Strack Consulting). In addition to simple closed polygons, inhomogeneity domains may share common edges. This greatly complicates preprocessing but is very useful, especially for problems with a varying aquifer base elevation (Figure 2).

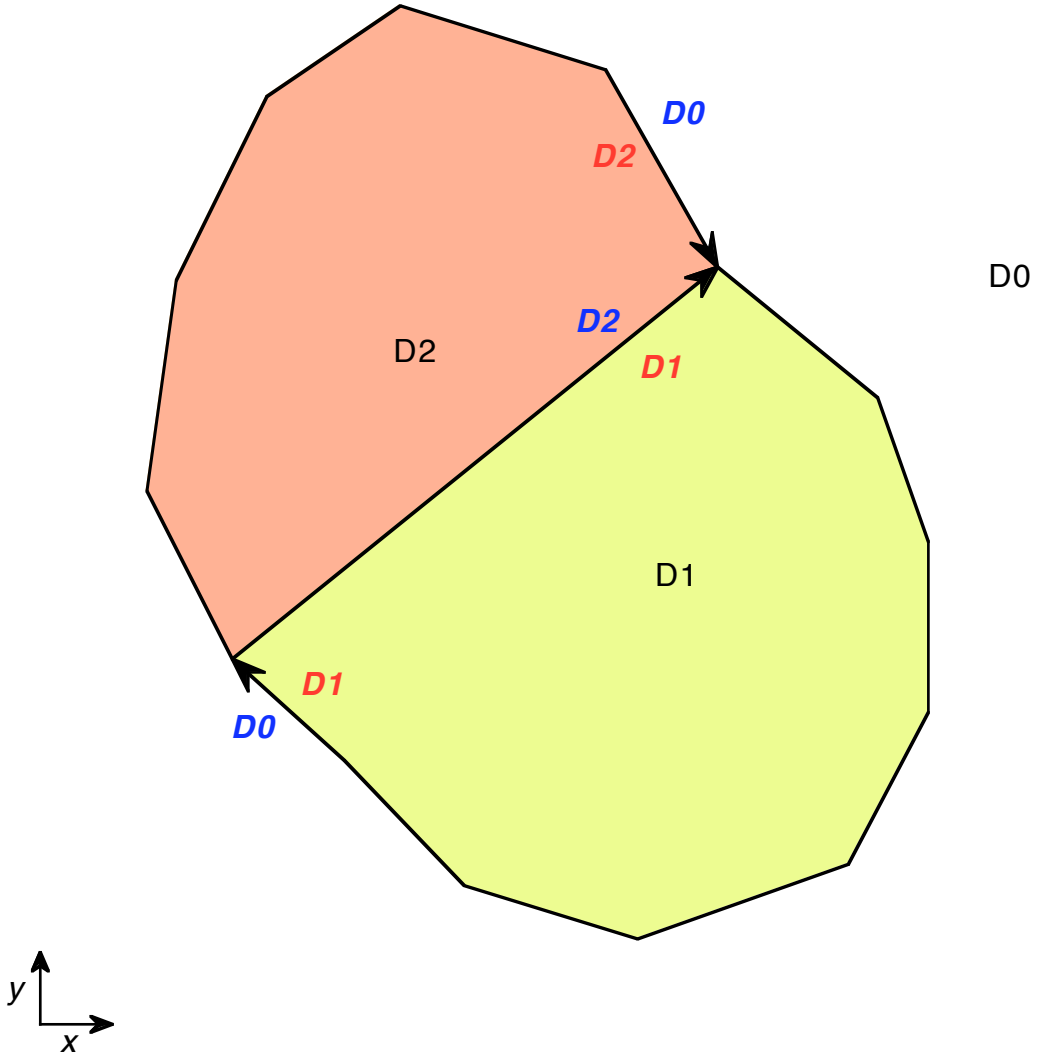


**Figure 2. Polygonal inhomogeneities with a common boundary. For each of the three domains, the hydraulic conductivity, aquifer base and thickness, and porosity may differ. For the perimeter arcs (black lines), the arrows denote the orientation of the arc (the arrowhead is placed at the last point on the arc). For each arc, the "righthand" domain is labeled in red text and the "left–hand" domain is shown in blue text.**

The following properties may be changed in ModAEM inhomogeneities:

- Hydraulic conductivity, *K*
- Aquifer base elevation, *b*
- Aquifer thickness, *H*

- Aquifer effective porosity, $n_e$

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 ModAEM solution for inhomogeneities is linear, and 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 subdomain 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 subdomain. 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.

### 4.5.1 Specifying inhomogeneities

As mentioned above, ModAEM allows for both closed polygonal inhomogeneities that do not share edges and for conjoined polygonal inhomogeneities that may have shared edges. The *in0* directive begins the process of specifying inhomogeneities. ModAEM requires that this be done in two steps:

1. Define the polygonal regions using the *dom* directive
2. If common boundaries are present, define the strings of elements that implement the boundaries using the *str* directive.

The second task, when required, can be very challenging. 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, ir is not necessary to use common boundaries. Fortunately, ModAEM makes it easier to specify these.

**Usage**:

```
aqu ndomains nstrings base ...
  in0
    dom npts base height hyd_cond porosity avg-head id ponded lbl
      (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 are provided in the *aqu* directive that begins the aquifer specification. The specification of inhomogeneities begins with the *in0* directive. Inside the *IN0* section of the script, domains are specified first, then if necessary, the strings that actually implement the boundaries are provided.

**Defining an inhomogeneity domain (*dom* directive)**

Domains are defined as follows:

```
dom npts base thickness hyd-cond porosity avg-head id ponded lbl
   (x1, y1)
   (x2, y2)
   ...
```

**nvertices** **integer**

The maximum number of vertices which may be used to delineate the boundary of the domain.

**base** **real [L]**

The base elevation of the aquifer.

**thickness** **real [L]**

The thickness of the aquifer.

**hyd-cond** **real [L/T]**

The hydraulic conductivity of the aquifer domain.

**porosity** **real [–]**

Porosity of the aquifer as a fraction.

**avg-head** **real [L]**

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 may be 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. 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. [L] {real}

**id** **integer**

A unique identification number for the domain.

**ponded** **logical**

This is a flag that states whether the subdomain is to be considered to be "ponded water", e.g. a pond, lake, or gravel pit. If the domain is considered ponded water, pathline traces will be terminated at the point the particle enters the domain.

**lbl** **text**

An optional textual label for the domain.

Following each *dom* directive, up to *npts* vertices may be provided, one vertex per line in the input file. Each vertex is a single coordinate provided as an *(x, y)* pair. Data entry for coordinates ends when a *dom*, *str*, or *end* directive is encountered.

**Defining a bounding string of elements (*str* directive)**

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 1, it is not necessary to define the perimeter strings and the *str* directive may be omitted. ModAEM will automatically build closed strings of elements.

If common boundaries are to be used, all perimeter arcs must be provided using the str directive, as follows (see Figure 2):

```
str npts left right closed id lbl
      (x1,y1)
      (x2,y2)
      ...
```

**npts** **integer [–]**

The maximum number of points which may be used to delineate the arc.

**left** **integer [–]**

The ID number of the polygon on the left of the arc (see Section111.5)

**right** **integer [–]**

The ID number of the polygon on the right of the arc (see Section 1.5)

**closed** **boolean**

Flag: if true, the string closes on itself, making a polygon. If false, the string is an open polyline.

**id** **integer [–]**

A unique identification number for the arc.

**lbl** **text**

An optional textual label for the arc.

# 5 Element modules

Once the aquifer is defined using module *AQU* (see Chapter 4), 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 analogous 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. 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 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. 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 aquifer, not as a separate element.

**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). Stream flow routing may be performed for *LS2* .

**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; use the *BDY* elements included in module *AQU*. These are analogous 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 the "pond" function described by Strack (1989). The modeler provides the rate of infiltration per unit of surface area (or "sink density) for the element. 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 deprecated by the polygonal area–sink module *AS0*.

**Polygonal area–sinks (*AS0*)**

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

## 5.1 Discharge-specified wells (module *WL0*)

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

Usage of the *wl0* directive:

```
wl0 nwells
  (xw,yw) pumping-rate radius id
  ...
end
```

**nwells** **integer**

The maximum number of wells in the problem

The *wl0* directive is followed by one record for every well in the model. If more than *nwells* well elements is provided, ModAEM will terminate and report the error.

Each well record has this form:

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

**(xw, yw)** **complex [L, L]**

The coordinates of the center of the well.

**pumping-rate** **real [$L^3/T$]**

The well's pumping rate. The value is positive if the element removes water from the aquifer, negative if it adds water to the aquifer. Note that this is the opposite sign convention as USGS MODFLOW.

**radius** **real [L]**

The radius of the well

**id** **integer [–]**

A unique identification number for the well.

**lbl** **text**

A text label for the well.

## 5.2 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 of the *wl1* directive:

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

**nwells** **integer**

The maximum number of wells in the problem

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

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

**(xw, yw)** **complex [L, L]**

The coordinates of the center of the well.

**head** **real [L]**

The specified head at the control point.

**(xc, yc)** **complex [L, L]**

The coordinates of the point where the head condition is to be met

**radius** **real [L]**

The radius of the well.

**id** **integer**

A unique identification number for the well.

**lbl** **text**

A text label for the well.

## 5.3 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 of the *ls0* directive:

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

**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.

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

**npts** **integer [–]**

The maximum number of vertices in the line-sink string.

**id** **integer [–]**

A unique identification number for the string.

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 [L, L]**

Coordinates of the vertex.

**strength** **real [L²/T]**

The sink density $\sigma$ of the line–sink string at the vertex. The sink density is defined as the volumetric abstraction 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. The total volumetric infiltration or abstraction along the line is $Q = L\left(\frac{\sigma_1 + \sigma_2}{2}\right)$ [L³/T] where $\sigma_1$ and $\sigma_2$ are the strengths at consecutive vertices and $L$ is the distance between the vertices.

## 5.4 Head-specified line-sinks (module *LS1*)

Module *LS1* 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 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. 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 aquifer, not as a separate element.

Usage of the *ls1* directive:

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

**nstrings** **integer**

The maximum number of head-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.

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

**npts** **integer [–]**

The maximum number of vertices in the line-sink string.

**id** **integer [–]**

A unique identification number for the string.

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 [L, L]**

Coordinates of the vertex.

**head** **real [L]**

The specified head at the center of the line-sink string.

## 5.5 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.

Usage:

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

**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.

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

**npts** **integer [–]**

The maximum number of vertices in the line-sink 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 $\sigma = c(h_b - h_r)$ 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 [L/T]**

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 $c = w\frac{k_c}{t_c}$ 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.

**id** **integer [–]**

A unique identification number for the string.

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 [L, L]**

Coordinates of the vertex.

**head** **real [L]**

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.

**bottom** **real [L]**

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.

## 5.6 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. 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.

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 [L, L]**

Coordinates of the vertex.

## 5.7 Discharge-specified circular area-sinks (module *PD0*)

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

Usage:

```
pd0 nponds
  (xw,yw) strength radius id
  ...
end
```

**nponds** **integer [–]**

The maximum number of ponds in the problem

The *pd0* directive is followed by one record for every pond in the model. If more than *nponds* well elements is provided, an ModAEM will terminate and report the error. Each pond record has the following parameters:

**(xw, yw)** **complex [L, L]**

The coordinates of the center of the pond.

**sink-density** **real [L/T]**

The sink density of the pond. This is the value $\gamma = \frac{Q_p}{A_p}$ where $\gamma$ 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.

**radius** **real [L]**

The radius of the pond.

**id** **integer [–]**

A unique identification number for the pond.

## 5.8 Polygonal area-sinks (module *AS0*)

Module *AS0* creates discharge-specified polygonal area-sinks. 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 nareas
  str npts strength id
    (x,y)
    ...
  str ...
    ...
end
```

**nareas** **integer [–}**

The maximum number of discharge-specified line-sink strings 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.

Parameters for the *str* directive

**npts** **integer [–]**

The maximum number of vertices in the line-sink string.

**sink-density** **real [L/T]**

The sink density of the pond. This is the value $\gamma = \frac{Q_p}{A_p}$ where $\gamma$ 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. top/bottom An integer that specifies whether the area sinks are to be placed at the aquifer top (0) or bottom (1).

**id** **integer [–]**

A unique identification number for the string.

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

**(x, y)** **complex [L, L]**

Coordinates of the vertex.

# 6 Processing Directives

This chapter describes processing directives that control the solution process and general inspection 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).

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
# processing directives go here
sol ...
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.

### 6.1.1 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.

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

### 6.1.2 Processing directives that retrieve data from the model at a specific point

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.

### 6.1.3 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.

### 6.1.4 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.

## 6.2 Processing directives for solving the model and reporting the solution results

The following commands provide support for solving the model, saving and reloading previous solutions, or sending a complete model solution summary to an HTML report file.

### 6.2.1 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
```

**niter** **integer [–]**

The number of iterations to be performed. 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; your mileage may vary

**1–2 iterations**

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.

**3–8 iterations**

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.

**more iterations**

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. May require 10 or more iterations to achieve convergence.

**relaxation** **real [–]**

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 strength 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.

### 6.2.2 Directive RPT – report the solution in HTML format

Add words

## 6.3 Directives for extracting particular solution results

These directives provide a fast way to extract particular values from the modeled solution for display on the console or in the run log file.

### 6.3.1 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)
```

**(x,y)** **complex [L, L]**

The desired location as an *(x, y)* pair.

**Example:**

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

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

### 6.3.2 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 $\Phi$ is the discharge potential and $\Psi$ is the stream function. A solution must be present (see the *sol* directive) prior to issuing this directive.

Usage:

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

**(x,y)** **complex [L, L]**

The desired location as an *(x, y)* pair.

**Example:**

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

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

### 6.3.3 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: $Q_i = \int_{z_{bottom}}^{z_{top}} \left( q_x(z) + iq_y(z) \right) dz$ where $q_x(z)$ and $q_y(z)$ are the horizontal components of the specific discharge $q_i = -k\, \partial_i h$ at the position elevation z, and $z_{top}$ and $z_{bottom}$ are the elevations of the aquifer top and bottom, respectively. A solution must be present (see the *sol* directive) prior to issuing this directive.

Usage:

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

**(x, y)** **complex [L, L]**

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)
```

## 6.4 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 $\overline{v_i} = \frac{\overline{q_i}}{n_e} = \frac{\overline{Q_i}}{Hn_e}$ where $v_i$ 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 issuing this directive.

Usage:

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

**(x, y)** **complex [L, L]**

The desired location as an *(x, y)* pair.

**Example:**

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

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

### 6.4.1 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 5.7 and 5.8. 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)
```

**(x, y)** **complex [L, L]**

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)
```

### 6.4.2 FLO - report the total flow across a path

Directs ModAEM to report the total integrated groundwater flow (in units of $L^3/T$) across a linear path connecting two specified points to the message file and console. A solution must be present (see the *sol* directive) prior to issuing this directive. The sign on the reported value follows the right–hand rule (Section 1.5)

Usage:

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

**(x1, y1)** **complex [L, L]**

The first point on the path segment.

**(x2, y2)** **complex [L, L]**

The second point on the path segment.

**Example:**

To report the total integrated flux across a line segment from (100,100) to (200,200), use the directive

```
flo (100.0,100.0) (200.0,200.0)
```

## 6.5 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.

### 6.5.1 GRA - report the modeled numerical gradient in potential

Reports the numerical gradient in the discharge potential $\partial_i \Phi$ 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 6.3.3). A solution must be present (see the *sol* directive) prior to issuing this directive.

Usage:

```
gra (x,y) delta
```

**(x, y)** **complex [L, L]**

The *(x, y)* location where the numerical gradient is to be computed.

**delta**

The spacing for the points about *(x, y)* where the numerical derivatives are to be computed.

**Example:**

The numerical gradient in the complex potential at (100, 100) using a spacing of 0.1 is computed by the directive:

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

### 6.5.2 LAP - report the modeled numerical Laplacian of the potential

Reports the numerical Laplacian of the discharge potential $\nabla^2\Phi$ at a specified point to the "message" file and console. This directive is commonly used in program debugging; the numerical Laplacian should have approximately the same value as the total recharge rate (see the *rch* directive in Section 6.3.3). A solution must be present (see the *sol* directive) prior to issuing this directive.

Usage:

```
lap (x,y) delta
```

**(x, y)** **complex [L, L]**

The *(x, y)* location where the numerical Laplacian is to be computed.

**delta**

The spacing for the points about *(x, y)* where the numerical derivatives are to be computed.

**Example:**

The numerical Laplacian in the complex potential at (100, 100) using a spacing of 0.1 is computed by the directive:

```
lap (100.0,100.0) 1.0
```

# 7 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.

Module *INQ* provides three categories of processing directives:

- File management directives;
- Values extraction in a manner similar to the HEA and other directives that write to the run-time message log; and
- Element results extraction that writes the solution constants (strength coefficients and boundary-condition check values).

The remainder of this section describes the *INQ* directives in detail.

## 7.1 Module *INQ* file management directives

The commands in this section manage the writing of files from module *INQ*. At present, the options are limited. Future plans include a binary file option.

### 7.1.1 *FIL* directive – Initialize a new output file

The *FIL* directive opens a new output file. The file will be used for all output from the other *INQ* directives until the *END* directive or another *FIL* directive is encountered.

Usage:

```
fil filename.ext
```

**filename.ext** **text**

The name of the file to be written, including its extension. Module *INQ* does not append an extension to its output files.

## 7.2 Directives for model inspection

This section describes the INQ commands that extract particular values from the model. Output from these directives is written to the output file that was opened by the FIL directive (Section 7.1.1).

For each of these directives, each record in the output file is prefixed with the three-letter command for easy identification by postprocessor codes.

### 7.2.1 HEA - report the modeled head

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

Usage:

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

**(x,y)** **complex [L, L]**

The desired location as an *(x, y)* pair.

**id** **integer [–]**

An identifying number that may be used to find the value by a postprocessor.

**Output:**

Output is written as comma–separated data in the format:

```
HEA, ID, X, Y, HEAD
```

### 7.2.2 POT - report the modeled complex potential

Reports the complex potential at a specified point to the output file. The complex potential is defined as $\Omega = \Phi + i\Psi$ , where $\Phi$ is the discharge potential and $\Psi$ is the stream function. A solution must be present (see the *sol* directive) prior to issuing this directive.

Usage:

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

**(x,y)** **complex [L, L]**

The desired location as an *(x, y)* pair.

**id** **integer [–]**

An identifying number that may be used to find the value by a postprocessor.

**Output:**

Output is written as comma–separated data in the format:

```
POT, ID, X, Y, PHI, PSI
```

### 7.2.3 DIS - report the total aquifer discharge

Reports the total aquifer discharge $Q_x + iQ_y$ at a specified point to the output file. The total discharge is a two-dimensional analogue for the specific discharge: $Q_i = \int_{z_{bottom}}^{z_{top}} \left( q_x(z) + iq_y(z) \right) dz$ where $q_x(z)$ and $q_y(z)$ are the horizontal components of the specific discharge $q_i = -k\ \partial_i h$ at the position elevation z, and $z_{top}$ and $z_{bottom}$ are the elevations of the aquifer top and bottom, respectively. A solution must be present (see the *sol* directive) prior to issuing this directive.

Usage:

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

**(x, y)** **complex [L, L]**

The desired location in the complex plane x+iy. {complex}

**id** **integer [–]**

An identifying number that may be used to find the value by a postprocessor.

**Output:**

Output is written as comma–separated data in the format:

```
DIS, ID, X, Y, Q_X, Q_Y
```

## 7.3 VEL - report the horizontal groundwater velocity

Reports the horizontal aquifer velocity $v_x + iv_y$ at a specified point to the output file. The velocity should be interpreted as the vertically–averaged velocity at the point in question $\overline{v_i} = \frac{\overline{q_i}}{n_e} = \frac{\overline{Q_i}}{Hn_e}$ where $v_i$ 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 issuing this directive.

Usage:

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

**(x, y)** **complex [L, L]**

The desired location as an *(x, y)* pair.

**id** **integer [–]**

An identifying number that may be used to find the value by a postprocessor.

**Output:**

Output is written as comma–separated data in the format:

```
VEL, ID, X, Y, V_X, V_Y
```

### 7.3.1 RCH - report the net recharge rate

Reports the net rate of areal recharge at a specified point to the output file. The reported recharge rate has a sign consistent with the ModAEM conventions as described in e.g., Sections 5.7 and 5.8. 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) id
```

**(x, y)** **complex [L, L]**

The desired location in the complex plane x+iy. {complex}

**id** **integer [–]**

An identifying number that may be used to find the value by a postprocessor.

**Output:**

Output is written as comma–separated data in the format:

```
RCH, ID, X, Y, GAMMA
```

### 7.3.2 SAT - report the saturated thickness

Reports the saturated thickness at a specified point to the output. The reported thickness is the specified aquifer thickness, or the difference between the modeled head and the base elevation, whichever is smaller. A solution must be present (see the *sol* directive) prior to issuing this directive.

Usage:

```
sat (x,y) id
```

**(x, y)** **complex [L, L]**

The desired location in the complex plane x+iy. {complex}

**id** **integer [–]**

An identifying number that may be used to find the value by a postprocessor.

**Output:**

Output is written as comma–separated data in the format:

```
SAT, ID, X, Y, H
```

### 7.3.3 FLO - report the total flow across a path

Directs ModAEM to report the total integrated groundwater flow (in units of $L^3/T$) across a linear path connecting two specified points to the message file and console. A solution must be present (see the *sol* directive) prior to issuing this directive. The sign on the reported value follows the right–hand rule (Section 1.5)

Usage:

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

**(x1, y1)** **complex [L, L]**

The first point on the path segment.

**(x2, y2)** **complex [L, L]**

The second point on the path segment.

**id** **integer [–]**

An identifying number that may be used to find the value by a postprocessor.

**Output:**

Output is written as comma–separated data in the format:

```
FLO, ID, X1, Y1, X2, Y2, Q
```

## 7.4 Directives that report values associated with element results

This section discusses the INQ directives that report the results of the solution for the analytic elements in the model. The results include strength coefficients, measures of the accuracy of the match at the boundary conditions, and other values such as the modeled heads at wells.

### 7.4.1 Directive *WL0* – report the results for discharge–specified wells

Directive *WL0* writes a single record for each discharge–specified well in the model. Each well may be identified by the ID number associated with the well in the model run script (see Section 5.1). Usage:

```
WL0
```

For each well, an output record of the form below is written.

```
WL0, ID, XC, YC, Q, R, HEAD, DRY
```

**ID**

The ID number of the well

**(XC, YC)**

The location of the center of the well

**Q**

The pumping rate of the well

**R**

The radius of the well

**HEAD**

The modeled Dupuit–Forchheimer head at the well

**DRY**

Logical flag. *T* if the modeled head at the well is at or below the aquifer base, *F* if not.

### 7.4.2 Directive *WL1* – report the results for discharge–specified wells

Directive *WL1* writes a single record for each head–specified well in the model. Each well may be identified by the ID number associated with the well in the model run script (see Section 5.1). Usage:

```
WL1
```

For each well, an output record of the form below is written.

```
WL1, ID, XC, YC, R, XCP, YCP, CPHEAD, Q, HEAD, ERROR
```

**ID**

The ID number of the well

**(XC, YC)**

The location of the center of the well

**R**

The radius of the well

**(XCP, YCP)**

The location of the point where the specified head is provided

**CPHEAD**

The specified head at *(XCP, YCP)*

**Q**

The pumping rate of the well

**HEAD**

The modeled Dupuit–Forchheimer head at *(XCP, YCP)*

**ERROR**

The error in head at *(XCP, YCP)*

### 7.4.3 Directive *LS0* – report the results for discharge–specified wells

Directive *LS0* writes a single record for each discharge–specified line sink in the model. Each well may be identified by the ID number associated with the line sink in the model run script (see Section 5.1). Usage:

```
LS0
```

For each line–sink segment, an output record of the form below is written.

```
LS0, ID, SEG, X, Y, L, STRENGTH, HEAD
```

**ID**

The ID number of the line–sink string

**SEG**

The segment index along the string

**(X, Y)**

The location of the first vertex of the segment

**L**

The length of the segment

**STRENGTH**

The sink density of the segment

**HEAD**

The modeled Dupuit–Forchheimer head at the center of the segment

**Note**

For the last vertex of the string, the only relevant information in the output record is the *(x, y)* location.

### 7.4.4 Directive *LS1* – report the results for head–specified wells

Directive *LS1* writes a single record for each head–specified line sink in the model. Each well may be identified by the ID number associated with the line sink in the model run script (see Section 5.1). Usage:

```
LS1
```

For each line–sink segment, an output record of the form below is written.

```
LS1, ID, SEG, X, Y, L, CPHEAD, STRENGTH, HEAD, ERROR
```

**ID**

The ID number of the line–sink string

**SEG**

The segment index along the string

**(X, Y)**

The location of the first vertex of the segment

**L**

The length of the segment

**CPHEAD**

The specified head at the center of the segment

**STRENGTH**

The sink density of the segment

**HEAD**

The modeled head at the center of the segment

**ERROR**

The difference CPHEAD-HEAD

**Note**

For the last vertex of the string, the only relevant information in the output record is the *(x, y)* location.

### 7.4.5 Directive *LS2* – report the results for head–specified wells

Directive *LS2* writes a single record for each resistance–specified line sink in the model. Each well may be identified by the ID number associated with the line sink in the model run script (see Section 5.1). Usage:

```
LS2
```

For each line–sink segment, an output record of the form below is written (on a single line).

```
LS2, ID, SEG, MODE, ENABLED?, X, Y, L, CPHEAD, DEPTH,
STRENGTH, HEAD, Q_ERROR, ROUTE?, ROUTE_MODE, DOWN_ID, QB, QOV, QS
```

**ID**

The ID number of the line–sink string

**SEG**

The segment index along the string

**MODE**

The boundary condition mode for the segment (see Section 7.4.5)

**ENABLED?**

*T* if the segment is enabled as a boundary condition at the end of the solution. *F* if the segment has been disabled (e.g. if it has run dry during the routing code or is percolating).

**(X, Y)**

The location of the first vertex of the segment

**L**

The length of the segment

**CPHEAD**

The specified head at the center of the segment

**DEPTH**

The value of the "depth" parameter

**STRENGTH**

The sink density of the segment

**HEAD**

The modeled head at the center of the segment

**Q_ERROR**

The difference between the modeled strength and the infiltration rate computed from the conductance and the head difference across the resistance layer

**ROUTE?**

*T* if flow routing along the string is enabled, *F* if not.

**DOWN_ID**

The ID number of the downstream reach (if routing is enabled)

**QB, QOV, QS**

The modeled baseflow, overland flow, and stream flow, respectively, at the end of the segment

**Note**

For the last vertex of the string, the only relevant information in the output record is the *(x, y)* location.

# 8 Analysis modules

ModAEM provides for the implementation of *analysis* modules, which are specialized modules that perform specific computations or analyses on model results. In the future, analysis modules will also be able to intervene in the solution process for specialized applications.

Presently, two analysis modules are available for generating grids of model results for contouring and other analyses, and particle tracking.

## 8.1 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, Matlab, 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.

An example of input for generating a grid of heads is shown below. Note that the *end* directive is needed to exit the module.

```
gri
  dim 50
  win (-100,-100) (100,100)
  hea mygrid
end
```

See the following pages for detailed instructions about module *GRI*.

### 8.1.1 Selecting the *GRI* output file options (*opt* directive)

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

```
opt option-value
```

**option-value** **text**

Selects an option, as listed below.

**SURFER**

Write the output file in SURFER ASCII format. The output extension (see Section 8.1.4) is `.grd`.

**MATLAB**

Write the file in Matlab format. The output extension (see Section 8.1.4) is `.m`.

**ARC**

Write the file as an ESRI ASCII ARC grid. The output extension (see Section 8.1.4) is `.grd`.

**XYZ**

Write the file as *(x, y, z)* triplets, one to a line. The output extension (see Section 8.1.4) is `.xyz`.

**PROFILE**

Enables "profile" mode. This mode allows for high–resolution grids in the vertical direction and sets the "missing value" flag for points in the section where the pressure head is negative (potentiometric head is less than the elevation). This option is "added" to the file-format options above.

### 8.1.2 Defining the *GRI* output grid window (*win* directive)

Defines the window to be gridded.

Usage:

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

**(x1, y1)** **complex [L, L]**

The lower-left corner of the (rectangular) region to be gridded.

**(x2, y2)** **complex [L, L]**

The upper-right corner of the (rectangular) region to be gridded.

**Example:**

To set the gridding window to the region from (-100,-100) to (100,100) use the directive

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

**Note:**

When the grid is computed, the spacing of the points in the output grid is determined according to the grid resolution (see the *DIM* command in Section 8.1.3) and the setting of the *PROFILE* option (Section 8.1.1).

### 8.1.3 Choosing the *GRI* module grid resolution (*dim* directive)

Sets the number of grid points along the long axis of the window.

Usage:

dim npts

**npts** **integer [–]**

The number of evenly-spaced points to compute along the long axis of the rectangular grid region. Depending upon whether the `option profile` flag has been set (see Section 8.1.1), the grid spacing will be computed as follows:

**PROFILE mode off**

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

**PROFILE mode on**

ModAEM will compute the same number of grid points in both dimensions (for higher vertical resolution in profile models).

**Example:**

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

```
gri 50
```

### 8.1.4 Computing a grid and writing it to a file (directives *hea*, *pot*, *psi*, *q_x*, *q_y*, and *flo*)

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**

`hea filename`

generates a grid of the potentiometric head $h$ on the file `filename_head.ext` , where the extension `ext` is determined by the file type (see Section 8.1.1).

**POT – Create a grid of potentials**

`pot filename`

generates a grid of the discharge potential $\Phi$ on the file `filename_pot.ext` , where the extension `ext` is determined by the file type (see Section 8.1.1).

**PSI – Create a grid of stream functions**

`psi filename`

generates a grid of the stream function $\Psi$ on the file `filename_psi.ext` , where the extension `ext` is determined by the file type (see Section 8.1.1).

**Q_X – Create a grid of discharges in the *x*–direction**

`q_x filename`

generates a grid of the discharge $q_x$ on the file `filename_qx.ext` , where the extension `ext` is determined by the file type (see Section 8.1.1).

**Q_Y – Create a grid of discharges in the y-direction**

`q_y filename`

generates a grid of the discharge $q_y$ on the file `filename_qy.ext` , where the extension `ext` is determined by the file type (see Section 8.1.1).

**FLO – Create a grid of integrated flow values**

```
flo filename (x, y)
```

generates a grid of the integrated flow between the point *(x, y)* and the points in the grid on the file `filename_qy.ext`, where the extension *`ext`* is determined by the file type (see Section 8.1.1). This command is useful for flow–nets in profile models, because it removes the *branch cuts* adjacent to wells and line–sink elements.

## 8.2 Tracing particle path lines (module *TR0*)

Module *TR0* provides facilities for determining particle paths and groundwater travel times by particle tracking.

An example of input for generating a set of pathlines is shown below. Note that the *end* directive is needed to exit the module.

```
tr0
  fil mypaths
  win (-100,-100) (100,100)
  poi (10,10) 10 0 0 1
  ...
end
```

See the following pages for detailed instructions about module *TR0*.

### 8.2.1 Choosing the output path line file (*FIL* directive)

The *fil* directive creates a new output file. Usage:

```
fil filename
```

**filename** **text**

The name of the file to be written. ModAEM appends the extension "`.tr0`" for the output file name.

### 8.2.2 Defining the *TR0* spatial window (*WIN* directive)

Defines the window for tracing. When path lines leave the window, they will be terminated.

Usage:

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

**(x1, y1)** **complex [L, L]**

The lower-left corner of the (rectangular) region for tracing.

**(x2, y2)** **complex [L, L]**

The upper-right corner of the (rectangular) region for tracing.

**Example:**

To set the tracing window to the region from (-100,-100) to (100,100) use the directive

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

### 8.2.3 Setting the tracing parameters (*TUN* directive)

The *tun* directive adjusts the parameters that control the accuracy and performance of the predictor–corrector tracing algorithm. For most problems, the default settings are sufficient, but for certain applications, it may be beneficial to adjust the parameters.

Usage:

```
tun step prox factor min-step
```

**step** **real [L]**

The step size for the predictor–corrector tracing algorithm. Like most analytic element codes, ModAEM computes steps based on the length of the steps (as opposed to stepping in time). After a step is taken, the time is updated according to the average velocity along the step.

**prox** **real [–]**

The proximity to elements for reducing the step size, expressed as a number of current step values. The current step size will be reduced at a distance of *prox*current_step* by the factor *factor*.

**factor** **real [–]**

The factor for reducing the current step size when within a distance of *prox*current_step* of an element or boundary.

**min-step** **real [–]**

The minimum step size to be used. If *min-step* is less than *factor*step*, the step size will be reduced recursively, but it will never be made less than *min-step*.

### 8.2.4 Setting the maximum tracing time (*TIM* directive)

When particle paths are computed, they will be terminated if the travel time exceeds the maximum travel time specified in the *tim* directive.

Usage:

```
tim max-time
```

**max-time** **real [T]**

The maximum time of travel for pathlines.

### 8.2.5 Setting the transport parameters (*TRA* directive)

This directive sets the transport properties for concentration calculations along pathlines. Currently, ModAEM supports retardation and decay.

Usage:

```
tra retardation decay
```

**retard** **real [–]**

The retardation factor.

**deacy** **real [$T^{-1}$]**

The first-order decay coefficient.

### 8.2.6 Trace a single particle (*POI* directive)

This directive releases a single particle for tracing.

Usage:

```
poi (x0, y0) z0 t0 c0 dir
```

**(x0, y0)** **complex [L, L]**

The *(x, y)* position of the starting point.

**z0** **real [L]**

The elevation of the starting point. If it is above the top of the saturated thickness, it will be placed at the top of the saturated thickness; if below the aquifer bottom, it will be placed at the aquifer bottom.

**t0** **real [T]**

The starting time for the particle. This facilitates continuing a pathline if desired.

**c0** **real [M/L$^3$]**

The starting concentration for the particle.

**dir** **integer [–]**

The direction of tracing, forward (1) or backward (-1) in time.

### 8.2.7 Trace a particles from along a line segment (*LIN* directive)

This directive releases equally–spaced particles along a line segment for tracing.

Usage:

```
poi (x1, y1) (x2, y2) n z0 t0 c0 dir
```

**(x1, y1) and (x2, y2)** **complex [L, L]**

The *(x, y)* positions of the ends of the line segment.

**n** **integer [–]**

The number of particles to be traced.

**z0** **real [L]**

The elevation of the starting point. If it is above the top of the saturated thickness, it will be placed at the top of the saturated thickness; if below the aquifer bottom, it will be placed at the aquifer bottom.

**t0** **real [T]**

The starting time for the particle. This facilitates continuing a pathline if desired.

**c0** **real [M/L$^3$]**

The starting concentration for the particle.

**dir** **integer [–]**

The direction of tracing, forward (1) or backward (-1) in time.

### 8.2.8 Trace a particles from a grid of starting points (*GRI* directive)

This directive releases equally–spaced particles in a rectangular grid for tracing.

Usage:

```
poi (x1, y1) (x2, y2) n z0 t0 c0 dir
```

**(x1, y1) and (x2, y2)** **complex [L, L]**

The *(x, y)* positions of the upper–left and lower–right corners of the grid.

**n** **integer [–]**

The number of particles to be traced, along the long axis of the rectangle bounded by *(x1, y1)* and *(x2, y2)*.

**z0** **real [L]**

The elevation of the starting point. If it is above the top of the saturated thickness, it will be placed at the top of the saturated thickness; if below the aquifer bottom, it will be placed at the aquifer bottom.

**t0** **real [T]**

The starting time for the particle. This facilitates continuing a pathline if desired.

**c0** **real [M/L$^3$]**

The starting concentration for the particle.

**dir** **integer [–]**

The direction of tracing, forward (1) or backward (-1) in time.

### 8.2.9 Trace a particles from a *WL0* well element (*WL0* directive)

This directive releases equally–spaced particles around a discharge–specified well. The tracing direction will be selected according to the pumping rate of the well, forward from injection wells or backward for pumping wells.

Usage:

```
wl0 id n z0 t0 c0 dir
```

**id** **integer [–]**

The ID number of the well.

**n** **integer [–]**

The number of particles to be traced about the well.

**z0** **real [L]**

The elevation of the starting point. If it is above the top of the saturated thickness, it will be placed at the top of the saturated thickness; if below the aquifer bottom, it will be placed at the aquifer bottom.

**t0** **real [T]**

The starting time for the particle. This facilitates continuing a pathline if desired.

**c0** **real [M/L$^3$]**

The starting concentration for the particle.

**dir** **integer [–]**

The direction of tracing, forward (1) or backward (-1) in time.

### 8.2.10 Trace a particles from a *WL1* well element (*WL1* directive)

This directive releases equally–spaced particles around a head–specified well. The tracing direction will be selected according to the pumping rate of the well, forward from injection wells or backward for pumping wells.

Usage:

```
wl1 id n z0 t0 c0 dir
```

**id** **integer [–]**

The ID number of the well.

**n** **integer [–]**

The number of particles to be traced about the well.

**z0** **real [L]**

The elevation of the starting point. If it is above the top of the saturated thickness, it will be placed at the top of the saturated thickness; if below the aquifer bottom, it will be placed at the aquifer bottom.

**t0** **real [T]**

The starting time for the particle. This facilitates continuing a pathline if desired.

**c0** **real [M/L$^3$]**

The starting concentration for the particle.

**dir** **integer [–]**

The direction of tracing, forward (1) or backward (-1) in time.