USGS
science for a changing world

Groundwater Resources Program

GWM–VI—Groundwater Management with Parallel Processing for Multiple MODFLOW Versions

Chapter 48 of
Section A, Groundwater
Book 6, Modeling Techniques

Techniques and Methods 6–A48

U.S. Department of the Interior
U.S. Geological Survey

GWM–VI—Groundwater Management with Parallel Processing for Multiple MODFLOW Versions

By Edward R. Banta and David P. Ahlfeld

Chapter 48 of
Section A, Groundwater
Book 6, Modeling Techniques

Groundwater Resources Program

Techniques and Methods 6–A48

U.S. Department of the Interior
U.S. Geological Survey

U.S. Department of the Interior
SALLY JEWELL, Secretary

U.S. Geological Survey
Suzette M. Kimball, Acting Director

U.S. Geological Survey, Reston, Virginia: 2013

For more information on the USGS—the Federal source for science about the Earth, its natural and living resources, natural hazards, and the environment, visit http://www.usgs.gov or call 1–888–ASK–USGS.

For an overview of USGS information products, including maps, imagery, and publications, visit http://www.usgs.gov/pubprod

To order this and other USGS information products, visit http://store.usgs.gov

Suggested citation:
Banta, E.R., and Ahlfeld, D.P., 2013, GWM–VI—Groundwater management with parallel processing for multiple MODFLOW versions: U.S. Geological Survey Techniques and Methods, book 6, chap. A48, 33 p., http://pubs.usgs.gov/tm/6a48/.

Preface

This report documents GWM–VI, a computer program that can be used to analyze groundwater-management problems using alternate versions of the U.S. Geological Survey modular, three-dimensional groundwater-flow model, MODFLOW, with support for parallel processing. The performance of the program has been tested in a variety of applications. Future applications, however, might reveal errors that were not detected in the test simulations. Users are requested to notify the U.S. Geological Survey, Colorado Water Science Center, of any errors found in this report or the computer program. Updates might occasionally be made to both the report and to the computer program. Users can check for updates on the Internet at UR*L: http://water.usgs.gov/software/lists/groundwater/.*

Contents

Figures

Tables

GWM–VI—Groundwater Management with Parallel Processing for Multiple MODFLOW Versions

By Edward R. Banta[1] and David P. Ahlfeld[2]

Abstract

Groundwater Management–Version Independent (GWM–VI) is a new version of the Groundwater Management Process of MODFLOW. The Groundwater Management Process couples groundwater-flow simulation with a capability to optimize stresses on the simulated aquifer based on an objective function and constraints imposed on stresses and aquifer state. GWM–VI extends prior versions of Groundwater Management in two significant ways—(1) it can be used with any version of MODFLOW that meets certain requirements on input and output, and (2) it is structured to allow parallel processing of the repeated runs of the MODFLOW model that are required to solve the optimization problem. GWM–VI uses the same input structure for files that describe the management problem as that used by prior versions of Groundwater Management. GWM–VI requires only minor changes to the input files used by the MODFLOW model. GWM–VI uses the Joint Universal Parameter IdenTification and Evaluation of Reliability Application Programming Interface (JUPITER-API) to implement both version independence and parallel processing. GWM–VI communicates with the MODFLOW model by manipulating certain input files and interpreting results from the MODFLOW listing file and binary output files. Nearly all capabilities of prior versions of Groundwater Management are available in GWM–VI. GWM–VI has been tested with MODFLOW-2005, MODFLOW-NWT (a Newton formulation for MODFLOW-2005), MF2005-FMP2 (the Farm Process for MODFLOW-2005), SEAWAT, and CFP (Conduit Flow Process for MODFLOW-2005). This report provides sample problems that demonstrate a range of applications of GWM–VI and the directory structure and input information required to use the parallel-processing capability.

[1]U.S. Geological Survey, Denver, Colo.

[2]University of Massachusetts, Amherst

Introduction

GWM–VI is a new version of a computer program designed to solve groundwater-management problems. The Groundwater Management (GWM) Process (Ahlfeld and others, 2005, 2009, 2011; Ahlfeld and Barlow, 2013) enables users of the U.S. Geological Survey (USGS) modular, three-dimensional groundwater model MODFLOW to analyze and solve various types of groundwater-management problems. The first version of GWM (Ahlfeld and others, 2005) used MODFLOW-2000 (Harbaugh and others, 2000). It was originally named MF2000-GWM; the name was later changed to GWM-2000. GWM was adapted to MODFLOW-2005 (Harbaugh, 2005) with the July 2007 release of MF2005-GWM. Support for Local Grid Refinement (LGR; Mehl and Hill, 2005 and 2007) was added, and the program name was changed with the release of GWM-2005 (Ahlfeld and others, 2009). The state-variable capability (Ahlfeld and others, 2011) and use of multi-node wells as decision variables (Ahlfeld and Barlow, 2013) were added in later versions of GWM-2005. The theory underlying GWM, methods for formulating management problems, detailed descriptions of capabilities, discussion of sample problems, and instructions for GWM input files are described in the reports referenced above. The authors of this report presume that the reader is sufficiently familiar with GWM that this information need not be repeated here.

To obtain a solution to a groundwater-management problem, GWM needs to make numerous simulations of the groundwater-flow problem. The GWM Process implemented in GWM-2000 and GWM-2005 is compiled with their respective versions of MODFLOW. This limits applicability of the GWM Process to cases in which the user has implemented a model application using either MODFLOW-2000 or MODFLOW-2005. Over the years, alternate versions of MODFLOW have been developed. Examples of these versions are MODFLOW-NWT (Niswonger and others, 2011), which uses an alternate-solution technique for the groundwater-flow process, and MF2005-FMP2 (Schmid and Hanson, 2009), which uses MODFLOW-2005 as the underlying flow-process simulator and adds additional process-modeling capabilities. GWM–VI is designed to work with these alternate versions. GWM–VI also has been tested with SEAWAT (Langevin

and others, 2008) and the Conduit Flow Process (CFP) for MODFLOW-2005 (Shoemaker and others, 2007). Other MODFLOW-based simulation programs also may be successfully used with GWM–VI, provided that they fulfill the requirements described in the section titled "MODFLOW Input and Output Requirements."

For simplicity in the remainder of this report, the term MODFLOW is used as a generic term to represent any MODFLOW-based groundwater-flow simulation program that meets the requirements described in the "MODFLOW Input and Output Requirements" section.

This report documents the GWM–VI (Groundwater Management – Version Independent) suite of programs, which includes GWM–VI, MMProc, and JRunnerM. In this report, "GWM–VI" is used to refer to the main program. The group of three programs is referred to as the "GWM–VI suite of programs" or simply the "GWM–VI suite." The GWM–VI program obtains information about the flow process by running a MODFLOW model prepared and identified by the user. The GWM–VI program manipulates some of the input required by MODFLOW and uses output generated by MODFLOW. MMProc preprocesses some of the MODFLOW input files to incorporate model-input values managed by GWM–VI, and it extracts information needed by GWM–VI from MODFLOW output files. JRunnerM is used in parallel processing to implement a link between GWM–VI and MMProc. Each of these programs is described in its own section.

For problems that necessitate lengthy run times to solve the groundwater-flow problem, GWM–VI can be used in a parallel-processing mode, using a suitable set of single-processor, multi-processor, and(or) multi-core computers to provide a substantial reduction in execution time, in comparison with GWM-2000 or GWM-2005. GWM–VI uses the technology of the Joint Universal Parameter IdenTification and Evaluation of Reliability Application Programming Interface (JUPITER API) (Banta and others, 2006) to implement parallel processing and version independence.

GWM–VI supports nearly all the features available in GWM-2005; features not available are described in the section titled "GWM Features Not Available in GWM–VI." The parallel-processing capabilities and requirements for running GWM–VI in parallel-processing mode are described in various sections of this report. Finally, a section is devoted to using GWM–VI, including details on setting up directories and the output generated by the programs.

The GWM–VI suite of programs, including source code, executable files for the Windows operating system, and test-case input and output files, may be obtained by downloading the distribution file from the URL listed in the Preface of this report. The distribution file is a self-extracting archive of directories and files, with the directories structured as illustrated in figure 1. The "Bin" directory contains executable files for the three programs in the GWM–VI suite of programs and a double-precision executable file of MODFLOW-2005 for running example analyses. The "Doc" directory contains documentation files. The "Src" directory contains all source-code files needed to build the GWM–VI suite of programs. The "Test" directory contains a number of subdirectories that contain files for testing GWM–VI; they also serve as examples of how input for GWM–VI is prepared. The "Test"\test-run directory contains batch files that can be used to run each example.

```
GWM-VI.1_0_0
  Bin
  Doc
  Src
    GWM-VI
    JRunnerM
    JUPITER_lib
    MMProc
    PrecUtls_lib
  Test
    Data
    Dewater
    Dewatermb
    Drain
    Maximin
    MNW-supply
    Runner1
    Runner2
    Seawater
    Storage
    Streamflow
    Supply2
    Test-out
    Test-run
```

Figure 1. Structure of distribution directory for the GWM–VI suite of programs

Overview of the GWM–VI Suite of Programs

The GWM–VI suite of programs was developed to address a need to use GWM with alternate versions of MODFLOW and to be able to apply multi-processor computers or multiple computers in a network to the task of efficiently solving groundwater-management problems. The GWM–VI suite comprises three computer programs:

GWM–VI is the main program for the GWM–VI suite. It reads GWM input, manipulates MODFLOW input, and uses MODFLOW output to formulate and solve linear, nonlinear, and mixed-binary linear groundwater-management problems. GWM–VI may be used in serial-processing mode or in parallel-processing mode. In serial-processing mode, GWM–VI invokes MMProc to preprocess MODFLOW input and to postprocess MODFLOW output. In parallel-processing mode, GWM–VI, in JUPITER API terminology (Banta and others, 2006), acts as a "dispatcher" program. In its dispatcher

capacity, GWM–VI assigns groundwater-flow simulations to be run by instances of a "runner" program on the same computer as the one running GWM–VI or on other computers in a network. In parallel-processing mode, the runner program (JRunnerM) effects communication of data between GWM–VI and MMProc.

MMProc uses data prepared by GWM–VI to prepare MODFLOW input, to execute MODFLOW, and to extract required values from MODFLOW output. The user is responsible for providing a MODFLOW executable file and input files needed to simulate the groundwater system of interest.

JRunnerM is a runner program based on the JUPITER API. It is similar to the jrunner program documented by Banta and others (2006), with enhancements required by GWM–VI.

Each of these programs is documented in the following sections of this report. In normal usage, the user executes GWM–VI. MMProc, JRunnerM, and MODFLOW are invoked or used automatically, as described in the following sections and demonstrated in appendix 1.

Implementation of Version Independence and Parallel Processing

GWM–VI requires that the user provide a MODFLOW executable file and appropriate input files to carry out the flow-process simulation. GWM–VI reads both the input for the GWM Process and for MODFLOW provided by the user. GWM–VI follows the conventions for input and output structure used by MODFLOW in its different versions. It expects to find a MODFLOW Name file, Discretization file, Basic Package input file, and other MODFLOW input files; if relevant to the management problem, it also expects to find Well (WEL) and MNW2 (Konikow and others, 2009) Package input files. During execution of GWM–VI, MODFLOW is repeatedly run. Before each run, GWM–VI or one of the other programs in the suite communicates the values of the flow-rate decision variables to MODFLOW by rewriting the WEL and/or MNW2 input files. MODFLOW is then executed. The MODFLOW executable file is expected to create an output LIST file similar to that produced by MODFLOW-2005. It is also expected to produce binary files containing heads and cell-by-cell flows in a form similar to that produced by MODFLOW-2005. MMProc reads the LIST file and the various binary files to interpret the results of the MODFLOW run.

GWM–VI has been successfully tested with the following versions of MODFLOW: MODFLOW-2005, Version 1.9 (Harbaugh, 2005); SEAWAT, Version 4 (Langevin and others, 2008); MODFLOW-NWT, Version 1.0.4 (Niswonger and others, 2011); CFP, Version 1.8 (Shoemaker and others, 2007); and Farm Process (FMP2), Version 1.0 (Schmid and others, 2006; Schmid and Hanson, 2009). Other versions of MODFLOW may function with GWM–VI. Any version of MODFLOW to be used with GWM–VI must meet the requirements described in the "MODFLOW Input and Output Requirements" section.

Previous versions of GWM (for example, GWM-2005) incorporate both optimization and the simulation of groundwater flow in a single program but support only serial processing. An important aspect of the development of parallel processing for GWM as documented in this report was the separation of the optimization functionality from the simulation of groundwater flow. This approach has two benefits: (1) it facilitates the implementation of support for parallel processing, and (2) it provides the option to use alternate versions of MODFLOW. In this approach, optimization is handled by GWM–VI and groundwater flow is simulated by the user's choice of a version of MODFLOW that meets the requirements listed in the "MODFLOW Input and Output Requirements" section. The GWM–VI suite of programs makes extensive use of modules of the JUPITER API (Banta and others, 2006) to effect communication between GWM–VI and MODFLOW. The Dependents Module of the JUPITER API is used to define model-dependent variables, which are used for implementing head-based, streamflow-based, storage-based, and drain-based constraints. The Model Input-Output (Model_IO) Module is used to prepare model-input files that define groundwater-system stresses controlled by flow-rate decision variables. The Model_IO Module also is used to extract model-calculated values needed to construct the response matrix for solution of the groundwater-management problem and to define variables subject to constraint. Several other modules of the JUPITER API are used as needed for support functionality. When a model run is needed, GWM–VI uses capabilities of the JUPITER API to prepare model-input files, execute the model, and acquire model-calculated results of interest.

When GWM–VI is run in serial-processing mode, GWM–VI invokes MMProc directly, and MMProc runs MODFLOW by invoking an operating-system call using a command provided by the user. The command may be an operating-system command that directly invokes MODFLOW, or it may be a batch file or other script file that includes a command to invoke MODFLOW.

When GWM–VI is run in parallel-processing mode, base groundwater-flow simulations are carried out by invoking MMProc directly, as described for serial-processing mode. For populating the response matrix, model runs are made in parallel. For these model runs, the user sets up a series of "runner" directories, such that model runs may be executed simultaneously without interfering with each other. The runner directories may be located on a single computer or on multiple computers on a local area network. Details of parallel-processing are presented in the section titled "Sequence of Program Execution—Parallel Processing."

GWM-2005 was designed to recognize and deal appropriately with certain common problematic situations that arise during the simulation and optimization steps necessary to solve a groundwater-management problem. GWM–VI is designed to handle the same problematic situations. However, because the optimization and the groundwater-flow simulations are handled by separate

programs, the implementation of the response to these situations is necessarily somewhat different. Three types of problems related to simulation of the groundwater-flow system by MODFLOW are addressed: (1) nonconvergence of a MODFLOW simulation, (2) conversion of managed-flow cells to "dry," and (3) conversion of cells involved in head-based constraints to "dry." In addition, as with GWM-2005, GWM–VI is designed to evaluate the precision of the response coefficients generated by perturbation of flow-rate decision variables and to modify perturbation increments if the coefficients are not sufficiently precise.

In addition to the values extracted from MODFLOW output, MMProc produces a short file (named "modflow.status") containing a flag that indicates convergence status and a flag that indicates if any managed-flow cell was converted to "dry." If flags in modflow.status indicate either that MODFLOW did not converge or that one or more managed-flow cells were converted to "dry," JRunnerM returns meaningless placeholder values in place of model-calculated values to GWM–VI, along with the values read from modflow.status.

GWM–VI uses a modified version of the Parallel-Processing Module of the JUPITER API. The modifications enable GWM–VI to evaluate the values from modflow.status, which indicate the convergence status and active or "dry" status of managed-flow cells. If the MODFLOW simulation did not converge or if one or more managed-flow cells went "dry," the modified parallel-processing module would revise the perturbation increment for the appropriate decision variable and make another run of the MODFLOW simulation in one of the runner directories. HDRY is a numeric value written by MODFLOW to elements in an array of model-calculated head values corresponding to cells that have been converted to inactive status (identified as "dry" in MODFLOW output) during solution of the groundwater-flow equation. "Drying" of head-constraint cells is recognized by the modified parallel-processing module by comparing the model-calculated head values reported by JRunnerM with HDRY as read from one of MODFLOW's flow packages (LPF, BCF6, Hydrologic Unit Flow [HUF, Anderman and Hill, 2000], or Upstream Weighting [UPW, Niswonger and others, 2011]). The modified parallel-processing module also evaluates the precision of the response coefficients, as is done by GWM-2005. If any head-constraint cells go "dry" or if response coefficients are not sufficiently precise, MODFLOW simulations are rerun in an attempt to correct the problem, as is done by GWM-2005.

GWM Features Not Available in GWM–VI

GWM–VI includes most of the capabilities of GWM-2005. However, some capabilities are not supported because of the difference in the way the two codes are constructed. GWM-2005 is compiled into a single executable file that includes MODFLOW-2005. As a result, GWM-2005 can access all arrays within MODFLOW-2005 and extract whatever information is needed.

In contrast, GWM–VI depends upon the ability of MODFLOW to output the LIST file and binary files of relevant information. These files are the only means by which GWM–VI can extract information from MODFLOW. Any GWM-2005 capability that relies on information that cannot be obtained from the MODFLOW output files is either unavailable or available in a modified form in GWM–VI. These limitations are discussed in this section.

CRITMFC – Alternate Flow Process Termination Tests

CRITMFC is a parameter in the GWM "Solution and Output-Control Parameter" (SOLN) file that controls the method used to determine the acceptability of a MODFLOW run. Setting CRITMFC to a positive value in GWM-2005 causes the test for acceptability to be based on the percentage discrepancy in the volumetric budget rather than the convergence criteria HCLOSE and(or) RCLOSE. Checking for the percentage discrepancy with adequate precision is not possible without access to MODFLOW arrays at each time step. As a result, the parameter CRITMFC is not supported in GWM–VI, and the test for acceptability is based solely on the convergence criteria HCLOSE and(or) RCLOSE.

Head in MNW Wells

GWM-2005 includes capabilities that impose constraints and define state variables using the head in wells represented by the MNW2 Package (Konikow and others, 2009). GWM-2005 accomplishes this by directly accessing MNW arrays. The MNW2 Package (version 7.1, dated 8/26/2011) does not provide a means to output this information to binary files. As a result, head in wells cannot be directly determined by GWM–VI. If a constraint on head in a well is desired, then the GWM–VI user should impose constraints on the head in those cells in which the MNW well is located. Similarly, the user can define state variables as the head in those cells in which the MNW well is located. This approach is demonstrated in the MNW-Supply problem distributed with GWM–VI.

In GWM-2005, the head at each managed MNW well is tested at the conclusion of each stress period to see if it has become dry (that is, its value is equal to HDRY). If the head is dry, then the simulation is considered to have failed. Because GWM–VI does not have access to the head in MNW wells, GWM–VI instead automatically checks the head at all cells in which the MNW well is located. If the head in any of these cells is dry, then the simulation is considered to have failed. The checking of heads described here can be deactivated using parameter NPGNMX in the GWM SOLN file (Ahlfeld and others, 2005).

Output of Managed Flows in Volumetric Budget

In GWM-2005, managed WEL-type flows are listed on a separate line of the volumetric budget report that appears in

the LIST file produced by the flow-process simulation. This line is labeled "MANAGED WELLS." This line does not appear in the LIST file output from MODFLOW when used with GWM–VI. This is a result of the process in which flows are provided to MODFLOW—namely, both managed and unmanaged flows are provided in single WEL or MNW2 files. Therefore, MODFLOW cannot distinguish between managed and unmanaged flows in its calculation of the WELLS term in the volumetric budget printed to the LIST file.

Multi-Grid Flow Processes

GWM-2005 has the capability to solve problems in which the flow process is simulated using Local Grid Refinement (LGR, Mehl and Hill, 2007). This feature is not supported in this release of GWM–VI.

Stream Package

GWM-2005 has the capability to solve problems in which streamflow is represented using either the Stream Package (STR) or the Streamflow-routing Package (SFR). GWM–VI only supports the SFR Package (Niswonger and Prudic, 2006).

GWM–VI

The GWM–VI program implements the capabilities of GWM-2005 that are related to formulation of groundwater-management problems; construction of the response matrix; and solution of the linear, nonlinear, and mixed-binary linear management problems. Description and input instructions for GWM–VI are presented in following sections.

Description of GWM–VI

Nearly all aspects of the management formulation (decision variables, objective function, and constraints) that are supported by GWM-2005 are supported by GWM–VI. Decision variables include flow-rate decision variables, external decision variables, and binary variables. As in GWM-2005, a single objective function is either minimized or maximized with a choice on the weighting applied to the three types of decision variables. Four types of constraints can be specified: (1) upper and lower bounds on the flow-rate and external decision variables, (2) linear summations of the decision variables, (3) hydraulic-head-based constraints, and (4) streamflow-based constraints. In addition, state variables can be defined for head, streamflow (as simulated using the SFR Package [Niswonger and Prudic, 2006]), change in aquifer storage, and drain flow. These state variables can be placed in the objective function or included in linear summation constraints.

The GWM–VI program uses the Response Matrix Solution Package documented by Ahlfeld and others (2005). The response-coefficient matrix is populated by perturbing individual flow-rate decision variables and calculating response coefficients from results of the solution of the groundwater-flow equation. Whenever groundwater-flow simulation results are needed, either for a base run or a run in which flow-rate decision variables are perturbed, the suite of GWM–VI programs prepares model-input values based on flow-rate decision variables and interprets output files produced by MODFLOW.

GWM–VI is programmed using design principles consistent with the JUPITER API (Banta and others, 2008). A programmer's guide for GWM–VI is provided in appendix 2. Details of the JUPITER API parallel-processing methodology are described in chapter 12 of Banta and others, 2006.

GWM–VI File Input Instructions

GWM–VI reads an input file that references a number of other input files containing information that defines the groundwater-management problem. This file is similar to the GWM file required by GWM-2005 and described in detail in Ahlfeld and others (2011). The GWM–VI input file consists of a series of lines, each with a keyword followed by a file name. The GWM–VI input file has the following items:

0. #Text [optional]
1. **OUT** Fname [optional]
2. **DECVAR** Fname
3. **STAVAR** Fname [optional]
4. **OBJFNC** Fname
5. **VARCON** Fname
6. **SUMCON** Fname [optional]
   HEDCON Fname [optional]
   STRMCON Fname [optional]
7. **SOLN** Fname
8. **CONTROL** Fname
9. **NAM** Fname

Keywords are in **bold** font. Fname is a file name or path of an output or input file of the type indicated by the corresponding keyword. Explanation of each item follows:

0. Item 0 is optional and consists of one or more comment lines designated by a "#" character in column 1. Text is printed to the output file.
1. Item 1 is optional. A filename for output from GWM–VI may be specified as Fname following the keyword **OUT** in this item. If item 1 is omitted, a default name of GWM. OUT, written in the directory in which GWM–VI is invoked, is used.

2. Each management problem must include a file that provides information about the decision variables; the name of this file is specified by Fname following the keyword **DECVAR** in this item.

3. Item 3 is optional. Each management problem may include a file that provides information about the state variables; the name of this file is specified by Fname following the keyword **STAVAR** in this item.

4. Each management problem must include a file that provides information about the objective function; the name of this file is specified by Fname following the keyword **OBJFNC** in this item.

5. Each management problem must include a file that provides information on the lower and upper bounds specified for the flow-rate and external decision variables; the name of this file is specified by Fname following the keyword **VARCON** in this item.

6. Item 6 is optional. Each management problem may include up to three files that provide information about the types of constraints of the management problem that are allowed in GWM–VI. These files are specified by the records of item 6, which can be listed in any order. The keyword **SUMCON** identifies a file used to specify linear summation constraints; **HEDCON** identifies a file used to specify head constraints; and **STRMCON** identifies a file used to specify streamflow constraints.

7. Each management problem must include a file that provides information about the solution and output-control parameters necessary for a GWM simulation; the name of this file is specified by Fname following the keyword **SOLN** in this item.

8. Each management problem must include a file of information to define additional output options, to define the way in which GWM–VI will invoke MODFLOW and MMProc, and, optionally, to enable parallel processing; the name of this file is specified by Fname following the keyword **CONTROL**. Instructions for preparing the **CONTROL** input are described in the next section.

9. GWM–VI also needs to read the MODFLOW Name file. The Name file is expected to follow the form described in Harbaugh (2005). The name of the Name file is specified by Fname following the keyword **NAM**.

Items 0 to 7 of the GWM–VI input file are common to both GWM-2005 and GWM–VI. Input instructions for these items (files of type **DECVAR**, **STAVAR, OBJFNC**, **VARCON**, **SUMCON**, **HEDCON**, **STRMCON**, and **SOLN)** are described in Ahlfeld and others (2005, 2011) and Ahlfeld and Barlow (2013); input instructions for these files are not provided in this report.

The GWM Process information provided in items 2 through 7 of the GWM–VI input file is closely linked to the output that must be produced by MODFLOW. For example, if streamflow constraints are specified at particular stress periods in the file identified by the STRMCON keyword, then the output from the flow process must include a binary file that contains streamflows at those specified stress periods. A detailed description of the requirements of binary-file output is provided in the "Output Requirements for MODFLOW" section.

A GWM–VI input file can be used by GWM-2005; comparison of results between GWM-2005 and GWM–VI is thereby facilitated for management problems that are based on a MODFLOW-2005 executable. Items 8 and 9 of the GWM–VI input file are required for GWM–VI but not for GWM-2005. When a GWM–VI input file is run with GWM-2005, these two lines will be ignored.

When using GWM–VI, names of flow-rate decision variables, head-based constraints, streamflow-based constraints, and drain-based constraints need to conform to the JUPITER naming convention (Banta and others, 2006) because JUPITER modules are used to implement communication between GWM–VI and JRunnerM. The following two rules comprise the JUPITER naming convention:

Rule 1. The first character needs to be a letter of the English alphabet.

Rule 2. All characters after the first letter need to be a letter, digit, or member of the set: "_"; "."; ":"; "&"; "#"; and "@" (underscore, dot, colon, ampersand, number sign, and at symbol).

Lengths of names specified in GWM input are not restricted by the JUPITER naming convention because the maximum length permitted for decision-variable and constraint names in the DECVAR, HEDCON, and STRMCON files is less that the length limit imposed by the JUPITER API.

CONTROL File Input Instructions

A file identified by the keyword CONTROL in the GWM–VI input file provides input to define output options, to define the way in which GWM–VI invokes MODFLOW and MMProc, and, optionally, to enable parallel processing. JUPITER API-style input blocks provide information in the CONTROL file. For detailed information concerning the structure and use of input blocks, the reader is referred to Banta and others (2006). The instructions provided in this section, however, are sufficient for constructing the relatively simple input blocks required by GWM–VI. An example CONTROL file, which is used for the Dewater problem described in appendix 1, is provided in figure 2 for reference.

Input blocks have the basic structure:

BEGIN Blocklabel [Blockformat]
Blockbody
END Blocklabel

Where:

BEGIN is a keyword that defines the first line of an input block.

Blocklabel is one of several keywords recognized by GWM–VI. GWM–VI recognizes the following blocklabels: **Options, Simulation**, **Model_Command_Lines**, **Parallel_Control**, and **Parallel_Runners**.

Blockformat is an optional keyword that defines the format in which the input block is structured. Valid blockformats include **Keywords** and **Table**. If Blockformat is omitted or misspelled, the blockformat defaults to **Keywords**. Any input block can use either of the two blockformats, but for simplicity in describing the CONTROL file, the **Keywords** blockformat will be used in constructing the Options, Simulation, Model_Command_Lines, and Parallel_Control input blocks, and the **Table** blockformat will be used in constructing the Parallel_Runners input block.

Blockbody consists of one or more lines containing data to be used as input. Content and format of the blockbody depend on the specified blocklabel and blockformat.

END is a keyword that indicates the end of the input block. The blocklabel following the **END** keyword needs to match the blocklabel specified in the corresponding **BEGIN** line.

The format of the blockbody is determined by the choice of blockformat on the BEGIN line. When blockformat is **Keywords**, data are provided in the form of "keyword=value" entries, where "keyword" is one of the keywords recognized in an input block identified by a particular blocklabel, and "value" is an integer, floating point number, text string, or Boolean (true or false) value, as appropriate for the keyword preceding the "=" sign. In the keyword=value entry, spaces may be used on either side of the "=" sign as desired, and single or double quotation marks may be used in pairs. If value is a text string containing embedded blanks, quotation marks must enclose either the value or the entire keyword=value entry. For Boolean values, "True," "T," "Yes," and "Y" are synonymous; similarly, "False," "F," "No," and "N" are synonymous. Keywords and Boolean values are case-insensitive. Case is retained in text-string values, but comparison of strings is case-insensitive.

When blockformat is Table, the data in the blockbody are organized in rows and columns. A line of the form:

NROW=nr **NCOL**=nc **COLUMNLABELS**

```
BEGIN Options
  Verbose = 3
  MessageFile = dewater_messages.txt
END Options

BEGIN Simulation
  SimCommand = "..\..\bin\mf2005dbl ..\data\dewater.nam"
END Simulation

BEGIN Model_Command_Lines
  command = ..\data\dewater_cmr.bat
  commandid = Dewater_cmr
END Model_Command_Lines

BEGIN Parallel_Control
  parallel=true  wait=0.01
  VerboseRunner=4
  AutoStopRunners = true
  TimeOutFactor = 4.0
END Parallel_Control

BEGIN Parallel_Runners table
  nrow=2 ncol=3 columnlabels
  RunnerName  RunnerDir    RunTime
  Runner1     ..\runner1\   2.0
  Runner2     ..\runner2\   2.0
END Parallel_Runners
```

Figure 2. Example GWM–VI CONTROL file used for the Dewater sample problem in which parallel processing is invoked.

must follow the BEGIN line, where
nr is the number of rows of data and
nc is the number of columns.

One line of column labels must appear next, where the column labels include the keywords recognized within the input block. The recognized column labels are the same as the keywords that are recognized when the blockformat is specified as **Keywords**. A total of nc column labels are read. Following the line of column labels, GWM–VI reads nr lines of data, where nc values are read from each line. Text strings containing embedded blanks must be enclosed in single or double quotes. Column labels and Boolean values are case-insensitive; case is retained in text strings, and columns may appear in any order. Blockbody may contain extra columns headed by column labels that are not required or recognized in a particular input block; extra columns are ignored.

Note that keywords and column labels are "recognized" in JUPITER input blocks and that input blocks, by design, are allowed to contain keywords or columns that are not recognized and, therefore, unused. This logic precludes identification of spelling errors by GWM–VI. Many input values have appropriate defaults, as noted in the instructions for each input-block type in following sections. If a keyword or column label is misspelled or not present, no error message is issued, and the default value (if applicable) is used. If GWM–VI generates unexpected results, check the spelling of keywords.

GWM–VI reads up to five input blocks from the CONTROL file; three of these are optional. The blocklabels and order of the input blocks conform to the conventions of the JUPITER API. The input blocks are documented in the following sections.

Options Input Block

The Options input block is optional; blockformat **Keywords** is convenient. GWM–VI recognizes two keywords in this input block: **MessageFile** and **Verbose**.

MessageFile: Value is a text string to be used as a filename for an output file. If the MessageFile option is used, output generated by GWM–VI that is not directly related to the groundwater-management problem is directed to the file identified in value rather than the main GWM–VI output file. The output that is affected is generated by parts of the code that are used to implement communication between GWM–VI and either MMProc or JRunnerM. Output generated during reading of the Discretization file also is affected.

Verbose: Value is an integer that controls verbosity of output not directly related to the groundwater-management problem; the default value is 3. When MessageFile is specified, Verbose affects only the output directed to the MessageFile file. Valid values and meanings are:

0. No extraneous output;
1. Write warnings only;
2. Write warnings and notes;
3. Write warnings and notes, and echo selected input (default);
4. Write warnings and notes, and echo all input; and
5. Write warnings, notes, echoed input, and miscellaneous information.

Simulation Input Block

The Simulation input block is required because the command that MMProc will use to invoke MODFLOW is read from the Simulation input block; blockformat **Keywords** is convenient. GWM–VI recognizes one keyword in this input block: **SimCommand**.

SimCommand: Value is a text string to be used as an operating-system command that MMProc will use to start a MODFLOW run. The command needs to be valid if issued in the directory where GWM–VI is invoked or, when parallel processing is enabled, in any of the runner directories. This requirement imposes a degree of uniformity in the directory structure used for runner directories, as well as the directory where GWM–VI is invoked.

Model_Command_Lines Input Block

The Model_Command_Lines input block is required; blockformat **Keywords** is convenient. Two keywords are recognized: **Command** and **CommandID**; only **Command** is required. If a **CommandID** entry is used, it must follow the **Command** entry.

Command: Value is a text string to be used as an operating-system command to invoke MMProc. Value may be either an operating-system command that invokes MMProc directly, or it may be the name of a script file, such as an MS-DOS batch file, that invokes MMProc. Command is invoked by GWM–VI when run in serial-processing mode or when making base groundwater-flow simulations in parallel-processing mode. Command is invoked by JRunnerM to make groundwater-flow simulations required to populate the response matrix when GWM–VI is run in parallel-processing mode.

CommandID: Value is a text string used to identify the command in output generated by GWM–VI. CommandID plays no role in the functionality of GWM–VI. If omitted, the command identifier defaults to a blank string.

Parallel_Control Input Block

The Parallel_Control input block is required in order to invoke parallel processing but otherwise is optional; blockformat **Keywords** is convenient. Six keywords

are recognized: **Parallel**, **Wait**, **VerboseRunner**, **AutoStopRunners**, **OperatingSystem**, and **TimeoutFactor**.

Parallel: Value is a Boolean value, which, when True, invokes parallel processing. If omitted, **Parallel** defaults to False. When **Parallel** is specified as True, the Parallel_Runners input block is required.

Wait: Value is a floating-point number. **Wait** is a time delay, in seconds, used as needed in checking for presence of, and reading from, the JUPITER signal files (Banta and others, 2006) used to communicate between GWM–VI and JRunnerM. If omitted, **Wait** defaults to 0.001 seconds (sec). In most cases the default value is appropriate; however, in some cases **Wait** may need to be set to a larger value. The need to increase **Wait** can be identified by the appearance of the following message on the screen running either GWM–VI or JRunnerM: "Warning: WAIT time may be too small." Little is to be gained by setting **Wait** smaller than the default value.

VerboseRunner: Value is an integer that controls verbosity of output generated by JRunnerM. The valid values and meanings are the same as those defined for **Verbose**. If omitted, **VerboseRunner** defaults to 3.

AutoStopRunners: Value is a Boolean value. If **AutoStopRunners** is set to True, at the conclusion of a GWM–VI run, all runners (instances of JRunnerM) will terminate execution. If a sequence of runs of GWM–VI is anticipated and runner directories are located on remote computers in a network, users may find it preferable to allow runners to reset and continue running when GWM–VI execution finishes. To have runners reset rather than terminate, specify **AutoStopRunners** as False. If omitted, **AutoStopRunners** defaults to True.

OperatingSystem: Value is a text string indicating the operating system under which GWM–VI and JRunnerM are run. Valid values are "WINDOWS," "DOS," "UNIX," and "LINUX." In the context of GWM–VI, WINDOWS and DOS are synonymous, and UNIX and LINUX are synonymous. If omitted or misspelled, **OperatingSystem** defaults to "WINDOWS."

TimeoutFactor: Value is a floating-point number. The product of **TimeoutFactor** and **RunTime** of the Parallel_Runners input block is used to determine if a model run is overdue. If omitted, **TimeoutFactor** defaults to 3.0.

Parallel_Runners Input Block

The Parallel_Runners input block is required when **Parallel** is set to True in the Parallel_Control input block; blockformat Table is convenient. Three keywords (column labels) are recognized: **RunnerName**, **RunnerDir**, and **RunTime**; **RunnerName** and **RunnerDir** are required. For each processor or processor core involved in a parallel-processing setup, a row in the Parallel_Runners input block defines a name, a runner directory, and, generally, an expected run time for one model simulation.

RunnerName: Value is a text string that is a name used to identify the runner.

RunnerDir: Value is a text string identifying the runner directory where an instance of JRunnerM will run. The string needs to end in the directory-separator character used by the operating system, either "\" or "/".

RunTime: Value is a floating-point number defining the expected model run time (seconds). If omitted, **RunTime** defaults to 10 sec. The expected run time for each runner is adjusted as the runner completes model runs. When the parallel-processing capability is activated, GWM–VI keeps track of model run times and, if a model run is overdue, it starts the model run on a different runner. The product of the expected run time and **TimeoutFactor** of the Parallel_Control input block is used to determine if a model run is overdue. The expected run time does not need to be particularly accurate. However, to avoid unnecessary restarts due to overdue model runs, set **RunTime** to a value at least as large as the time expected for a model run to complete on each runner.

MMProc

The MMProc program is designed to be invoked either by GWM–VI or by JRunnerM; it is likely that users would never need to invoke it directly. Input for MMProc is prepared automatically by either GWM–VI or JRunnerM, and output generated by MMProc is read by either GWM–VI or JRunnerM. For users interested in a more complete understanding of the operation of the GWM–VI suite of programs, the "Pre- and Postprocessing by MMProc" section gives an overview of the functionality served by MMProc. The "Error Handling and Importance of Maintaining Duplicate Well and MNW2 Input Files" section contains information useful for all users.

Pre- and Postprocessing by MMProc

In brief, the job of MMProc is to prepare input for MODFLOW (including input controlled by the optimization algorithm of GWM–VI), execute MODFLOW, extract simulated head and cell-by-cell flow values of interest from MODFLOW's binary output files, interpret model-run status based on the MODFLOW LIST file and head values simulated at managed-stress cells, and report the simulated values and MODFLOW status to two text output files. MMProc is able to read binary cell-by-cell flow data generated by any of the following flow packages: BCF6, LPF, HUF (Anderman and Hill, 2000, 2003; Anderman and others, 2002), and UPW (Niswonger and others, 2011).

When MMProc starts, it opens a file named mmproc.in (which is prepared automatically by GWM–VI or JRunnerM), from which input will be read. If this file is not found, MMProc terminates with an error message. From the mmproc.in file, MMProc reads:

- The command (SimCommand of the Simulation input block of the CONTROL file) that will be used to invoke MODFLOW;
- The path of the MODFLOW Name file;
- The MODFLOW spatial and temporal discretization dimensions;
- Various variables defined in the MODFLOW Basic, flow, and solver package input files;
- The path of the binary head file written by MODFLOW;
- The path of the binary cell-by-cell flow file used by the MODFLOW flow package, if needed;
- The paths of binary cell-by-cell flow files used by the SFR2 Package (Niswonger and Prudic, 2006), or Drain Package, or both, if needed;
- Data to define managed stresses simulated with the Well Package;
- Data to define cells and desired flows included in managed MNW wells;
- Data to enable MMProc to extract specified model-simulated head and cell-by-cell flow values from MODFLOW binary output files;
- The path of the SFR2 Package input file, if needed; and
- The name of the auxiliary variable defined in the Drain Package that is used to identify the drain feature of interest in each drain cell, if needed.

After reading these data, MMProc performs a number of preprocessing steps:

- If simulated values generated by the SFR2 Package are required, MMProc reads data from the SFR2 input file that will enable it to extract the specified values from the appropriate MODFLOW binary output file.
- If the Well Package is being used to simulate managed stresses, the original Well Package input file containing unmanaged stresses is saved with a new name (generated by appending "_COPY.TXT" to the original file name). MMProc then combines unmanaged well stresses with managed stresses and overwrites the original Well Package input file with the combined well-stress information.

MMProc then starts the MODFLOW run by submitting the SimCommand to the operating system. When the MODFLOW run terminates, MMProc performs the following postprocessing steps:

- Model-simulated values are extracted from the binary head and cell-by-cell flow files as specified in the mmproc.in file;
- Model-simulated values are written to the SimulatedValues.out file;
- The model-convergence status is interpreted by reading the MODFLOW LIST file.
- The simulated values of head at the cells included in managed stresses simulated with the Well and MNW2 Packages are evaluated to determine if any of the cells converted to "dry" during the simulation;
- The modflow.status file is written; and
- If the Well Package input file was modified, the file is overwritten by the saved, original version of this file.

Execution of MMProc terminates when the postprocessing steps have been completed.

Error Handling and Importance of Maintaining Duplicate Well and MNW2 Input Files

GWM–VI, JRunnerM, and MMProc attempt to recognize common errors that may be encountered during execution. When errors are recognized, GWM–VI, JRunnerM, and MMProc continue execution and ensure that the original Well and(or) MNW2 Package input file(s) are restored with their original names. However, if an unrecognized error is encountered, one of these programs may terminate abnormally, leaving the modified Well and(or) MNW2 Package input file(s) in place. If this occurs, subsequent runs of MMProc will behave as if the modified file(s) is (are) the original file(s), and results will be misleading. For this reason, it is helpful to keep duplicate copies of the original Well and MNW2 Package input files, so that they can be used to replace the modified input files in the event of an abnormal termination of GWM–VI, JRunnerM, or MMProc.

JRunnerM

JRunnerM is a "runner" program, in the terminology of the JUPITER API (Banta and others, 2006). The purpose of a runner program is to generate model-input files, make model runs, and extract model-output values as required by a "dispatcher" program. In the GWM–VI suite of programs, GWM–VI is the dispatcher program. When a model run does not converge or causes managed-flow cells to convert to "dry," GWM–VI is designed to adjust the stresses at managed-flow cells and attempt another model run. The runner program used in conjunction with GWM–VI needs to be able to handle the non-convergence case. A runner program for GWM–VI needs to perform several steps to do its job:

1. Initialize itself;
2. Monitor a runner directory for a specially-named "signal" file (named "jdispatch.rdy") written by

GWM–VI. The signal file contains information needed by the runner, including model-input and -output file names, template file names, and instructions for extracting model-generated values of interest from model-output files;

3. When the signal file of step 2 is found, read the file;
4. Monitor the runner directory for another signal file ("jdispar.rdy") written by the dispatcher program containing values to be assigned as stresses in managed-flow cells;
5. When the signal file of step 4 is found, read it and use the stress values and template file to create a MMProc input file.
6. Make a model run by invoking MMProc;
7. Read the modflow.status file generated by MMProc;
8. (a) If the modflow.status file indicates convergence was reached and no managed-flow cells went dry, read model-output files to obtain model-calculated values needed by GWM–VI to evaluate head and streamflow constraints; or
8. (b) if the modflow.status file indicates that the model failed to converge or that any managed-flow cells went dry, generate meaningless placeholder values to take the place of model-calculated values corresponding to head and streamflow constraints;
9. Write a signal file ("jrundep.rdy") containing the values read from modflow.status and either the model-calculated values or the placeholder values generated in step 8; and
10. Prepare for another request for a model run by returning to step 4.

Banta and others (2006) describe a runner program called jrunner, which is distributed with the JUPITER API. Jrunner is capable of performing all of the steps outlined above except step 8; it is not designed to modify its behavior based on the contents of a model-output file (for example, modflow.status) or to generate placeholder values. To satisfy the requirements of step 8, JRunnerM was developed by making minor modifications to jrunner.

JRunnerM needs to be invoked such that its working directory is one of the runner directories listed in the Parallel_Runners input block of the CONTROL file listed in the GWM–VI input file. The simplest way to start JRunnerM under Microsoft Windows is to open a DOS (Disk Operating System) command window, use the DOS CD command as needed to ensure that the working directory is one of the runner directories, and start jrunnerm.exe, which is in the Bin directory of the GWM–VI distribution. When multiple processes need to run on the same computer running Windows, it may be helpful to lower the "Base Priority" of the process running JRunnerM by starting it with a command of the form:

```
start /low /wait jrunnerm
```

Alternatively, a script (for example a batch file) can be used to invoke instances of JRunnerM as needed and then invoke GWM–VI; this approach is demonstrated in the batch files provided in the Test folder and its subfolders in the GWM–VI distribution.

File names (or paths) for the command to invoke MODFLOW (identified by the keyword SimCommand in the Simulation input block of the CONTROL file) and for the command to invoke MMProc (identified by the keyword Command in the Model_Command_Lines input block) need to be accessible relative to the runner directory where JRunnerM is started. Care needs to be exercised to ensure that file names and relative paths reference the correct, up-to-date files from the directory where GWM–VI is started and from all the runner directories.

MODFLOW Input and Output Requirements

To properly function with GWM–VI, the version of MODFLOW being used must support input and output that meets certain specific requirements. This section provides detailed information on these requirements. The requirements are described using the notation and terminology of MODFLOW-2005 (Harbaugh, 2005).

Input Requirements for MODFLOW

GWM–VI uses the file identified by the NAM keyword in the GWM–VI input file as the MODFLOW Name file. It is assumed that the structure and contents of the Name file are consistent with MODFLOW-2005. GWM–VI will look for file types LIST, DIS, and BAS6. Depending on the flow package used, GWM–VI will look for file types BCF6, LPF, HUF, or UPW and will expect to find the OC file type and a file type that identifies the solver package. If streamflow or drains are part of the management problem, then GWM–VI will expect to find the SFR or DRN file types.

If well-type decision variables are used in the management formulation, then GWM–VI will expect to find a WEL file type. Even if there are no unmanaged wells, a WEL file must be present. Prior to each MODFLOW run, MMProc combines the original WEL file with managed well-type decision variables to create a new WEL file. If there are no unmanaged wells, an original WEL file must exist and be listed in the NAM file; however, the WEL Package input file should indicate no active wells.

Similarly, if MNW-type decision variables are used, then GWM–VI will expect to find an MNW2 file type. If no unmanaged MNW wells are present, then an MNW2 Package

input file that indicates that zero wells will be simulated must be provided. When MNW-type decision variables are used, MMProc extracts net flow values for each cell included in a managed MNW well and sums them for comparison with the flow specified for that well. If the sum of the net flows does not equal the specified flow, the well is considered to have dewatered. To ensure that flows recorded in the binary file are associated with the correct well, GWM–VI assigns a unique sequence number to each MNW well; it uses an auxiliary variable for this purpose. Because of this use of an auxiliary variable, it is necessary to specify "COMPACT BUDGET AUXILIARY" (or "AUX" in place of "AUXILIARY") in the MODFLOW Output Control file whenever the GWM problem includes one or more MNW-type decision variables.

The GWM Response Matrix Solution (RMS) Package performs a check on the success of each perturbation run using the number of significant digits in the difference between base and perturbed heads (Ahlfeld and others, 2005, p. 32–33). The number of digits is estimated using the ratio of the difference in heads and HCLOSE, the head convergence parameter used in many MODFLOW solvers. GWM–VI searches for a value of HCLOSE (or its equivalent) from the solver input file listed in the NAM file. GWM–VI is capable of searching the following solvers: SIP, PCG, PCGN (CLOSE_H), SOR, DE4, GMG, and NWT (HEADTOL). If a value of HCLOSE cannot be found, a default value that is an estimate of machine precision is used. The use of HCLOSE for perturbation testing can be deactivated by setting NPGNMX to zero in the SOLN input file.

Output Requirements for MODFLOW

MODFLOW is expected to provide output in certain forms so that it can be properly interpreted. GWM–VI uses the output to determine the success of the flow-process run and to obtain the values of heads, streamflows, and other quantities needed to evaluate constraints and state variables of the management problem. Much of the output is in the form of unformatted files. These files contain heads or cell-by-cell flow information. Control of the contents of the unformatted files is specified in the MODFLOW Output Control (OC) file. For many features, GWM–VI requires cell-by-cell data that are written only when the COMPACT BUDGET option is specified in the OC file; for this reason, COMPACT BUDGET must be specified whenever a GWM–VI run requires cell-by-cell flow data. Detailed instructions for producing unformatted output are given in following sections for each type of constraint and state variable. These are described using the notation and terminology of MODFLOW-2005 (Harbaugh, 2005).

Listing File

GWM–VI gauges the success of a run of the simulation model by examining the listing file produced by each run. GWM–VI does a case-sensitive search of the listing file looking for one of these phrases:

'FAILED TO MEET SOLVER CONVERGENCE'
'Failure to converge'
'FAILED TO CONVERGE'

To function properly, the MODFLOW version used must output one of these phrases to the Listing File when convergence has failed. Otherwise, GWM–VI will assume that the run has converged. At least one of these phrases is produced by the versions of MODFLOW with which GWM–VI has been tested. Other MODFLOW versions may need to be coded to include one of these phrases in the listing file.

Head Constraints and Head State Variables

When head constraints or head state variables are used, unformatted information about heads needs to be written to a file that is identified in the NAM file as a DATA(BINARY) file. The unit number assigned to this binary file must match the unit number specified in the Output Control file for the value of IHEDUN (HEAD SAVE UNIT).

GWM–VI requires heads for three cases:

(1) the time step at the end of a stress period at which each head constraint is imposed,
(2) the time step at the end of a stress period at which each head-type state variable is imposed,
(3) all time steps in any stress period in which a flow-rate decision variable is allowed to be active (these heads are used to evaluate if the cell containing a well has converted to "dry").

It is up to the user to ensure that the required head data are written to the binary file; MMProc has limited ability to recognize that required data are missing.

Streamflow Constraints and Streamflow State Variables

When a GWM–VI problem includes streamflow constraints or streamflow state variables, the SFR Package needs to write unformatted information about streamflow. Streamflow constraints and state variables can be either flow-type (representing flow in the stream leaving the specified reach) or leak-type (representing flow between the aquifer and stream in the specified reach). Flow and leakage information are written to separate unformatted files, each of which must be identified in the NAM file as a DATA(BINARY) file. The unit number assigned to the binary file that is to contain streamflow or leakage data must match the unit number specified in the SFR input file. Leakage data are read from a binary file associated with a positive value of ISTCB1; flow data are read from a binary file associated with a negative value of ISTCB2 (Niswonger and Prudic, 2006; appendix 1). When using streamflow constraints or streamflow state variables, it is necessary to use MODFLOW Output Control using words and to specify the COMPACT BUDGET option.

GWM–VI requires streamflow data for:

(1) the time step at the end of a stress period at which each streamflow constraint (either flow or leakage) is imposed, and
(2) the time step at the end of a stress period at which each streamflow-type state variable (either flow or leakage) is defined.

Storage State Variables

When storage state variables are used, unformatted information about groundwater-storage changes needs to be written to a file that is identified in the NAM file as a DATA(BINARY) file. The unit number assigned to this binary file must match the unit number specified in the BCF6 input file for the value of IBCFCB if using the Block-Centered Flow Package, in the LPF input file for the value of ILPFCB if using the Layer-Property Flow Package, in the HUF input file for the value of IHUFCB if using the Hydrologic-Unit Flow Package (Anderman and Hill, 2000), or in the UPW input file for the value of IUPWCB if using the Upstream Weighting Package (Niswonger and others, 2011). Storage-change information that is written to the binary file must include information at all the time steps required by GWM–VI. Data in the file for time steps not required by GWM–VI are ignored.

GWM–VI requires storage-change information at every time step spanned by the storage state variables SPSTRT and SPEND. For example, if a problem has 10 stress periods, each with 5 time steps, and the storage state variable is defined over stress periods 3 through 5, then GWM–VI will require that the binary file contain storage change information at time steps 11 through 25 for a total of 15 time steps. If multiple storage state variables are present, then the inclusive set of time steps must be written to the binary file.

Drain Package State Variables

When Drain-Package state variables are used, unformatted information about drain flow needs to be written to a file that is identified in the NAM file as a DATA(BINARY) file. The unit number assigned to this binary file must match the unit number specified in the DRN file for the value of IDRNCB. Drain information should be saved to the binary file (using ICBCFL (SAVE BUDGET) in the Output Control file) at the time steps needed by GWM–VI. When using drain-flow state variables, it is necessary to use MODFLOW Output Control using words and to specify the COMPACT BUDGET option.

GWM–VI requires drain information for the following:

(1) the time step at the end of a stress period at which each flow-type drain state variable is defined; and
(2) every time step spanned by a volume-type drain state variable as defined by the starting and ending stress periods, SPSTRT and SPEND.

The Drain Package allows multiple drains to be defined in a single cell of the MODFLOW grid. If a managed drain is present in a cell that has multiple drains, then a method must be available to distinguish among the drains in that cell. GWM–VI accomplishes this by using the AUXILIARY variable option in the Drain Package. The user defines the name of the AUX variable in data set 2 of the Drain Package input. The name used for GWM auxiliary variables must be “GWM-DR.” This name can appear with other auxiliary variables, which can be in any order. The user then defines a value for the auxiliary variable in data sets 4 and 6 of the Drain Package input. These values should be positive integer numbers; they serve as an index to identify the drain in that cell. In the GWM state variable input file, each drain cell is identified by its layer, row, column location and, optionally, its auxiliary variable index number. GWM–VI determines the correct drain, when multiple drains are present in a single cell, by matching the auxiliary variable values in the GWM state variable file and the Drain Package input file. When using GWM–VI and auxiliary variables are used to identify drains, ensure that the Output Control input file specifies “COMPACT BUDGET AUXILIARY” (or AUX in place of AUXILIARY). Otherwise, the required auxiliary variables of the Drain Package will not be written to the binary budget file, and GWM–VI will not be able to match managed drains with drain boundaries defined in the Drain Package.

Using GWM–VI

A GWM–VI run involves a series of program executions, file creations, and extractions of information from model-output files. When all input instructions are properly structured, this series of steps is automatic; the user simply invokes GWM–VI and a complete run is carried out. Any of the test files distributed with GWM–VI can be used to observe the program steps carried out automatically and in proper sequence; running these analyses in parallel-processing mode requires a computer with a minimum of two processors or processor cores. In setting up a new program run, it may be useful to understand this sequence of steps; they are described in detail here with reference, as an example, to the input file for the DEWATER test problem, which can be found in the set of problems distributed with GWM–VI.

GWM–VI may be run in serial-processing mode using a single processor, or in parallel-processing mode using multiple processors. This choice is indicated by parameters set in the CONTROL input file. The mode selected determines the directory structure required and the output generated. In both modes, the user creates a directory in which a single instance of GWM–VI is executed. When using either mode, this directory is the “master” directory.

When run in either serial-processing or parallel-processing mode, GWM–VI reads the GWM–VI input file described in the section titled “GWM–VI File Input Instructions.” Details of processing by the GWM–VI suite of

programs to perform a groundwater-flow simulation depend on the selected processing mode. The following sections describe the use of GWM–VI in serial- and parallel-processing modes.

Sequence of Program Execution—Serial Processing

Figure 3 illustrates the flow of data among programs when GWM–VI is used in serial-processing mode. Execution of GWM–VI in serial-processing mode begins with running the executable GWM–VI.exe, which uses as input a GWM–VI input file (for example, file dewater_serial.gwm). GWM–VI proceeds to prepare files for the rest of the execution. MMProc.in.jtf is a JUPITER API template file created by GWM–VI that contains information needed to generate input for MODFLOW (for example, the locations of flow-rate decision variables) and information needed to interpret the MODFLOW output (for example, the names of the binary files containing output and the cell locations for constraints). SimulatedValues.jif is a JUPITER API instruction file created by GWM–VI, which contains instructions on how to extract values of state variables and values subject to constraints defined in other GWM input from the SimulatedValues.out file written by MMProc. Modflow_status.jif is an instruction file created by GWM–VI, which contains instructions on how to extract values from the modflow.status file written by MMProc. If MNW-type decision variables are present, then a template file named mnw2_input.jtf is created, which is used by GWM–VI to generate the MNW2 input file used by MODFLOW. The resulting MNW2 input file will contain information to define managed MNW wells in addition to unmanaged MNW wells, if any are used.

In preparation for a run of MODFLOW, MMProc.in is created by GWM–VI from MMProc.in.jtf by replacing the placeholders for the flow rates for flow-rate decision variables with the actual flow rates to be used in the run. A new MNW2 Package input file is also created by GWM–VI using mnw2_input.jtf, if MNW-type flow-rate decision variables are present. Then, MMProc is run. MMProc creates a new WEL Package input file, if WEL-type flow-rate decision variables are present. MMProc then executes MODFLOW by submitting the SimCommand specified in the Simulation input block of the CONTROL file to the operating system. When the MODFLOW run has completed, MMProc extracts information of interest from MODFLOW output files and writes the modflow.status and SimulatedValues.out files.

Making a Serial Optimization Run

When GWM–VI is to be run in serial-processing mode, the CONTROL file listed in the GWM input file needs to contain a Simulation input block and a Model_Command_Lines input block, as described in the "CONTROL File Input Instructions" section. The CONTROL file also may contain an Options input block, a Parallel_Control input block, and(or) a Parallel_Runners input block. If the CONTROL file does not contain a Parallel_Control input block, the processing mode defaults to serial processing. If a Parallel_Control input block is included, it should specify PARALLEL as false. When PARALLEL is specified as false, other entries in the Parallel_Control input block are ignored, as is the Parallel_Runners input block. When making an optimization run in serial-processing mode, all processing is based in the directory where GWM–VI is invoked. In the event of an abnormal termination of MMProc, please refer to the "Error Handling and Importance of Maintaining Duplicate Well and MNW2 Input Files" section.

Setting Up Directories and Files—Parallel Processing

Before using GWM–VI in parallel-processing mode, the user must set up "runner" directories in which model runs are to be made, potentially simultaneously, for the purpose of populating the response matrix. The runner directories may be on the same computer as the one running GWM–VI or on other computers on a local area network. GWM–VI must be able to read and write files in all runner directories. The naming convention *\\server\directory\directory* (for Windows) or *//server/directory/directory/* (for Unix or Linux) may be used in the Parallel_Runners input block of the CONTROL file to reference directories on networked computers. If multiple computers on a network are to be used, the computers must use the same line-ending convention for text files. For example, if GWM–VI is run on a computer running the Windows operating system, then only computers running Windows are candidates for use in parallel processing. Similarly, if GWM–VI is run on a computer running any version of Unix or Linux, then only computers running Unix or Linux are candidates for parallel processing.

For successful parallel processing, care needs to be taken to ensure the following: (1) GWM–VI is able to read and write files in all runner directories; (2) The commands to invoke MODFLOW and MMProc, as specified in the GWM–VI CONTROL input file, are valid in the master directory where GWM–VI is invoked and in each of the runner directories; (3) All model-input files other than those that will be controlled by GWM–VI, JRunnerM, and MMProc are identical; (4) When WEL-type or MNW-type flow-rate decision variables are used, separate copies of WEL and(or) MNW2 Package input files for simulating unmanaged wells reside in both master and runner directories; and (5) Output files generated by instances of MODFLOW and MMProc running in the runner directories can be written without interfering with each other. Regarding item (4), if there are WEL-type or MNW-type flow rate decision variables, then the WEL and(or) MNW2 Package input files will be rewritten by GWM–VI, JRunnerM, or MMProc, in each case, combining managed and unmanaged flows into a single file.

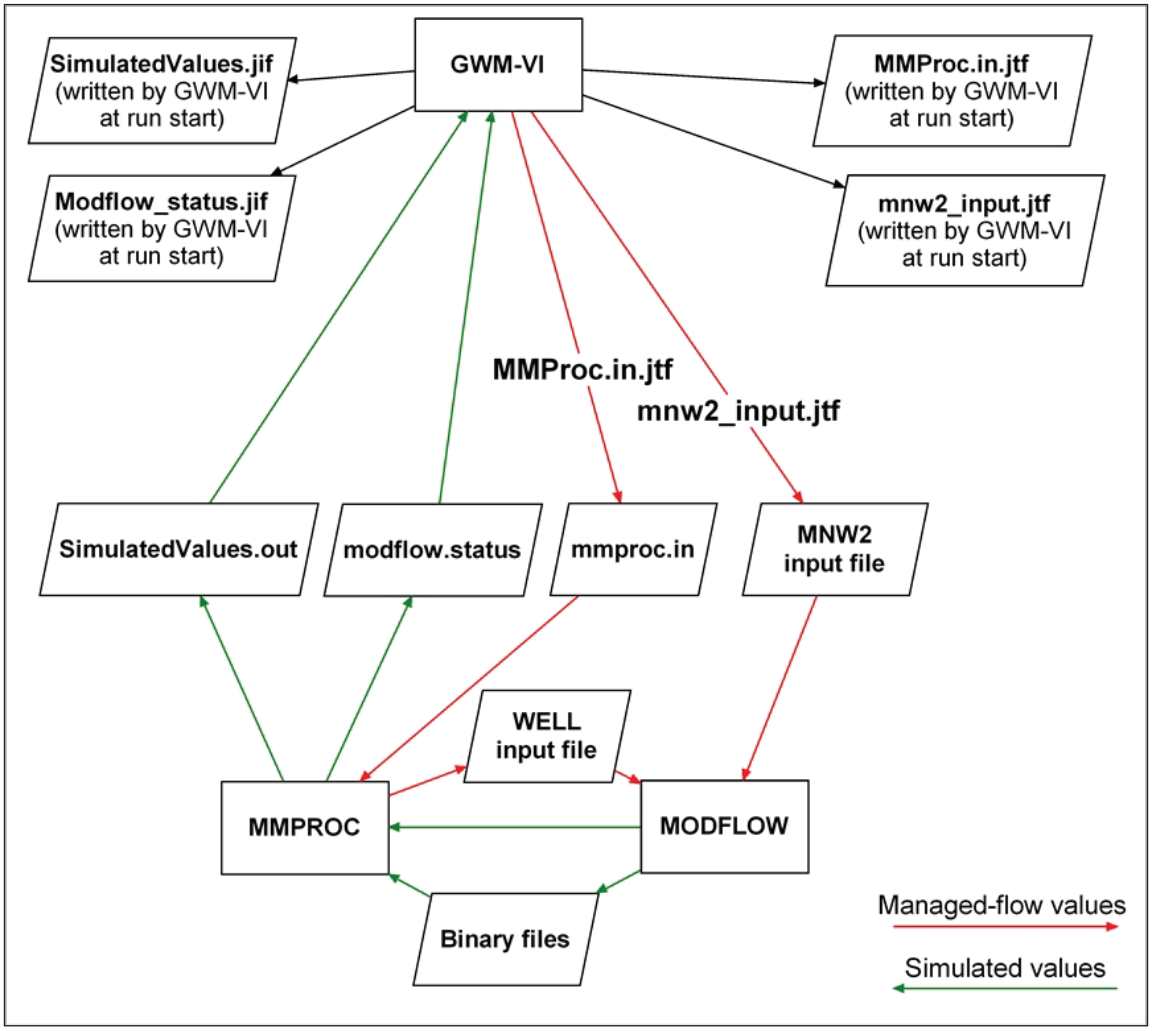

Figure 3. Flow of data used by the GWM–VI suite of programs during serial processing. Rectangles represent programs; rhomboids represent files.

Optimum Performance of Parallel Processing

For optimum performance, the number of programs (including GWM–VI and all instances of JRunnerM) in use at one time on a given computer generally needs to be limited to the number of processors or processor cores installed in the computer. The available random-access memory (RAM) installed in each computer also needs to be taken into account when determining how many instances of the groundwater model are to be run on a computer. When the RAM requirement of a single MODFLOW simulation is large, the number of simulations that can be simultaneously and efficiently run on a computer may be limited by the amount of installed RAM rather than the number of processors. Because of slow access times of virtual (disk-based) memory relative to RAM, it is worthwhile to ensure that GWM–VI and MODFLOW never use virtual memory. When deciding how many runners to use on a particular computer, ensure that the computer has enough RAM to accommodate MODFLOW runs for all runners simultaneously. If RAM is limiting, reduce the number of runners on that computer to avoid the use of virtual memory.

If the RAM requirement is not a limiting factor, the appropriate number of runner directories to use for a parallel-processing optimization run will depend on the number of processors or processor cores available, the number of flow-rate decision variables included in the analysis, and possibly, the speed of the local area network and the available processors. When the response matrix needs to be populated, one model run for each flow-rate decision variable is required in addition to a base model run. For the fastest possible execution time, a number of runners equal to the number of flow-rate decision variables should be used. When the number of flow-rate decision variables exceeds the number of runner directories in use, some or all of the runners will need to make multiple model runs. GWM–VI dispatches model runs to runners as they become available. While model runs are executing in the runner directories, GWM–VI remains

running and consuming processor time. If more than about four processors are available, processing may be facilitated by dedicating one processor to running GWM–VI. If the number of available processors is four or fewer, the advantage gained by dedicating a processor to run GWM–VI is reduced, and it may be advantageous to use as many runners as there are total available processors.

Sequence of Program Execution—Parallel Processing

The flow of information for model runs conducted in runner directories during parallel processing is illustrated in figure 4. When GWM–VI is used in parallel-processing mode, processing related to base groundwater-flow simulations is conducted in the master directory, and the procedure described in the "Sequence of Program Execution—Serial Processing" section applies. However, in parallel-processing mode, all model runs needed to populate the response matrix are conducted in the runner directories. This section describes the sequence of program execution for the potentially simultaneous model runs in the runner directories.

Before an optimization run can be made in parallel-processing mode, runner directories must be set up as described in the "Setting Up Directories and Files—Parallel Processing" section and populated as needed with current model-input files. The user starts an instance of JRunnerM in each runner directory to be used by GWM–VI, then starts GWM–VI.

Once GWM–VI has read all input, it attempts to communicate with each runner, using signal files as documented by Banta and others (2006). GWM–VI makes base groundwater-flow simulations using the SimCommand provided in the Model_Command_Lines input block of the CONTROL file. When the response matrix needs to be populated, GWM–VI instructs JRunnerM to make groundwater-flow simulations in the runner directories with perturbed flow-rate decision variables as needed by writing signal files named jdispar.rdy. When JRunnerM recognizes the presence of a file named jdispar.rdy, it reads the managed stress rates to be simulated by the Well and MNW2 Packages. For each run to be made by a runner, JRunnerM creates new versions of the MNW2 Package and MMProc input files, which include the stresses applied to the flow-rate decision variables for that run in addition to the unmanaged stresses. In each runner directory, JRunnerM invokes MMProc.

If there are managed stresses to be simulated with the Well Package, MMProc prepares a Well-Package input file by combining any unmanaged stresses in the existing Well-Package input file with managed stresses. When the Well-Package input file has been written, MMProc submits the

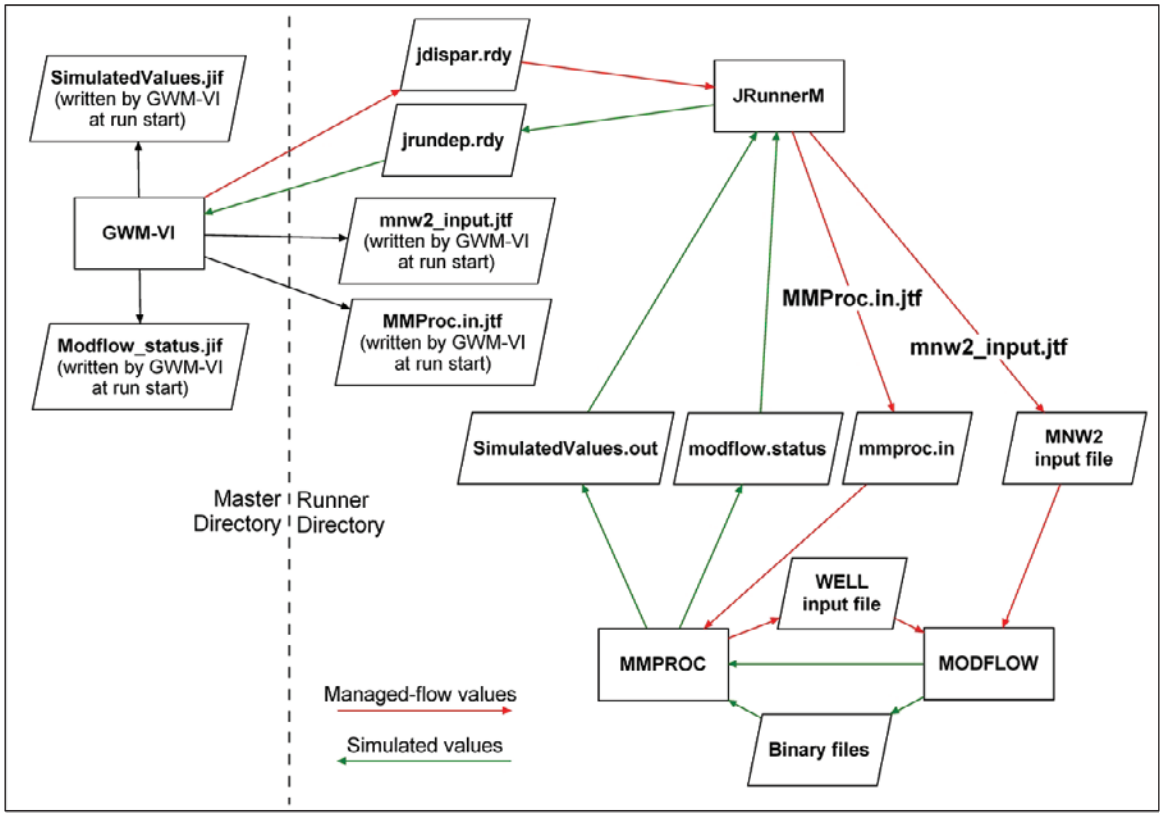

Figure 4. Flow of data used by the GWM–VI suite of programs during parallel processing. Rectangles represent programs; rhomboids represent files.

SimCommand provided in the Simulation input block of the CONTROL to the operating system to initiate the MODFLOW simulation. As for serial-processing, when the simulation has completed, MMProc extracts model-simulated values from MODFLOW binary output files and writes the modflow.status and SimulatedValues.out files. JRunnerM extracts values from modflow.status and, assuming the flags in modflow.status indicate that convergence was achieved and that no managed-flow cells went dry, it extracts model-calculated values needed for populating the response matrix from SimulatedValues.out. JRunnerM then writes the model-calculated values to a signal file named jrundep.rdy. When GWM–VI recognizes the presence of jrundep.rdy, it reads the model-calculated values and uses them to populate the response matrix. In the event of an abnormal termination of MMProc, please refer to the "Error Handling and Importance of Maintaining Duplicate Well and MNW2 Input Files" section.

If the number of flow-rate decision variables exceeds the number of available processors, GWM–VI will assign model runs to available processors as model runs complete and processors become available. GWM–VI will wait for a model run to complete, unless the execution time exceeds an expected run time multiplied by TimeOutFactor of the Parallel_Control input block of the CONTROL input file. The expected run time for each runner is set initially to the RunTime provided in the Parallel_Runners input block and is updated as model runs are completed.

Making a Parallel Optimization Run

When the master and runner directories have been properly set up with all required input files, a parallel-processing run of GWM–VI can be made. For most efficient operation, invoke JRunnerM in each runner directory listed in the Parallel_Runners input block of the CONTROL file before starting GWM–VI. If JRunnerM is started in only a subset of the runner directories, GWM–VI will run and use the functioning runners, but if JRunnerM has not been invoked in any of the runner directories, GWM–VI will stop with an appropriate error message. When the Parallel_Runners input block lists runner directories where JRunnerM has not been started, GWM–VI checks for signal files in those directories. This checking is necessary to allow runners to be started at a later time and to be included in the parallel processing, but if a runner is not to be used, efficiency is improved by eliminating it from the Parallel_Runners input block. When JRunnerM has been started in one or more runner directories, GWM–VI can be started, and processing proceeds as described in the "Sequence of Program Execution—Parallel Processing" section.

Output from the GWM–VI Suite of Programs

GWM–VI and the other programs in the suite produce: (1) output files describing the results of the groundwater-management problem, (2) files listing values extracted or interpreted from MODFLOW output, and (3) a variety of communication files. The communication files can normally be ignored by the user. However, if the program fails to function properly it may be useful to review these files. All output files are discussed in this section.

The file identified in the GWM input file with the OUT keyword is the primary output file from GWM–VI and contains the main output from the GWM Process. This file is nearly identical in content to the corresponding file produced by runs of GWM-2005.

MODFLOW is expected to produce a listing file and binary files that meet the requirements described in the "Output Requirements for MODFLOW" section of this report. MODFLOW may also produce other files depending on its input settings. These will be ignored by GWM–VI. If running GWM–VI in parallel-processing mode, it is recommended that the listing and binary files be written to the directory, either master or runner, in which MODFLOW is executed. This is demonstrated with the distribution test problems.

The GWM–VI suite of programs creates a set of communication files that convey information among GWM–VI, MMProc, and JRunnerM. These files are retained at the completion of a run of GWM–VI and can be viewed to diagnose problems. They may be discarded by the user after a run of GWM–VI has completed. The files are listed in table 1 along with brief descriptions of their contents. Files that are generated for the benefit of the user are listed in table 2.

Table 1. Files used to communicate information among GWM–VI, MMProc, and JRunnerM.

File name	Written by	Read by	Contents
MMProc.in.jtf	GWM-VI	GWM-VI or JRunnerM	Template file for creating input file for MMProc; copied to all Runner directories by GWM-VI
MMProc.in	GWM-VI or JRunnerM	MMProc	Input file for MMProc; written by GWM-VI in Master directory and by JRunnerM in Runner directories
Mnw2_input.jtf	GWM-VI	GWM-VI or JRunnerM	Template file used to create MNW2 input file
Modflow_status.jif	GWM-VI	GWM-VI or JRunnerM	Instructions for extracting values from Modflow.status file
Modflow.status	MMProc	GWM-VI or JRunnerM	Status report containing information about a MODFLOW run
SimulatedValues.jif	GWM-VI	MMProc	Instructions for extracting values from SimulatedValues.out file
SimulatedValues.out	MMProc	GWM-VI or JRunnerM	Values produced by MODFLOW

Table 2. Files generated by GWM–VI, MMProc, and JRunnerM for user.

File name	Written by	Contents
MMProc.out	MMProc	Echo of input read by MMProc
Run0000#._ext	JRunnerM	Values extracted from files written by MMProc; the 0000# part of the file name identifies the run number. These files can be useful when trying to identify problems that may be encountered during parallel processing.
__.mnw2_COPY.TXT	GWM-VI	Backup copy of the unmanaged MNW2 file; retained in case of unexpected program failure
__.wel_COPY.TXT	MMProc	Backup copy of the unmanaged WEL file; retained in case of unexpected program failure

Acknowledgments

The authors gratefully acknowledge the support of the USGS Groundwater Resources Program and the USGS California Water Science Center and the assistance of Kevin Mulligan of the University of Massachusetts, who performed substantial testing of GWM–VI.

References Cited

Ahlfeld, D.P., Baker, K.M., and Barlow, P.M., 2009, GWM-2005—A Groundwater-Management Process for MODFLOW-2005 with Local Grid Refinement (LGR) capability: U.S. Geological Survey Techniques and Methods, 6-A33, 65 p., *http://water.usgs.gov/nrp/gwsoftware/mf2005_gwm/MF2005-GWM.html*.

Ahlfeld, D.P., Barlow, P.M., and Baker, K.M., 2011, Documentation for the State Variables Package for the Groundwater-Management Process of MODFLOW-2005 (GWM-2005): U.S. Geological Survey Techniques and Methods 6-A36, 45 p., *http://water.usgs.gov/nrp/gwsoftware/mf2005_gwm/MF2005-GWM.html*.

Ahlfeld, D.P., and Barlow, P.M., 2013, Use of multi-node wells in the Groundwater-Management Process of MODFLOW-2005 (GWM-2005): U.S. Geological Survey Techniques and Methods 6-A47, *http://water.usgs.gov/nrp/gwsoftware/mf2005_gwm/MF2005-GWM.html*.

Ahlfeld, D.P., Barlow, P.M., and Mulligan, A.E., 2005, GWM—A Ground-Water Management Process for the U.S. Geological Survey modular ground-water model (MODFLOW-2000): U.S. Geological Survey Open-File Report 2005–1072, 124 p., *http://water.usgs.gov/nrp/gwsoftware/mf2k-gwm/MF2K-GWM.html*.

Anderman, E.R., and Hill, M.C., 2000, MODFLOW-2000, the U.S. Geological Survey modular ground-water model—Documentation of the Hydrogeologic-Unit Flow (HUF) Package: U.S. Geological Survey Open-File Report 00–342, 89 p., *http://water.usgs.gov/nrp/gwsoftware/modflow2005/modflow2005.html*.

Anderman, E.R., and Hill, M.C., 2003, MODFLOW-2000, the U.S. Geological Survey modular ground-water model—Three additions to the Hydrogeologic-Unit Flow (HUF) Package—Alternative storage for the uppermost active cells, Flows in hydrogeologic units, and the Hydraulic-conductivity depth-dependence (KDEP) capability: U.S. Geological Survey Open-File Report 03–347, 36 p., *http://pubs.er.usgs.gov/publication/ofr03347*.

Anderman, E.R., Kipp, K.L., Hill, M.C., Valstar, Johan, and Neupauer, R.M., 2002, MODFLOW-2000, the U.S. Geological Survey modular ground-water model—Documentation of the Model-Layer Variable-Direction Horizontal Anisotropy (LVDA) capability of the Hydrogeologic-Unit Flow (HUF) Package: U.S. Geological Survey Open-File Report 02–409, 60 p., *http://pubs.er.usgs.gov/publication/ofr02409*/.

Banta, E.R., Hill, M.C., Poeter, Eileen, Doherty, J.E., and Babendreier, Justin, 2008, Building model analysis applications with the Joint Universal Parameter IdenTification and Evaluation of Reliability (JUPITER) API: Computers & Geosciences, v. 34, no. 4, p. 310–319. [April, 2008]

Banta, E.R., Poeter, E.P., Doherty, J.E., and Hill, M.C., 2006, JUPITER—Joint Universal Parameter IdenTification and Evaluation of Reliability—An application programming interface (API) for model analysis: U.S. Geological Survey Techniques and Methods, book 6, chap. E1, 268 p., *http://pubs.er.usgs.gov/usgspubs/tm/tm6E1*.

Harbaugh, A.W., 2005, MODFLOW-2005, the U.S. Geological Survey modular ground-water model—The ground-water flow process: U.S. Geological Survey Techniques and Methods, book 6, chap. A16, variously paginated, *http://pubs.er.usgs.gov/usgspubs/tm/tm6A16*.

Harbaugh, A.W., Banta, E.R., Hill, M.C., and McDonald, M.G., 2000, MODFLOW-2000, the U.S. Geological Survey modular ground-water model—User guide to modularization concepts and the Ground-Water Flow Process: U.S. Geological Survey Open-File Report 00–92, 121 p., *http://pubs.er.usgs.gov/publication/ofr200092*.

Konikow, L.F., Hornberger, G.Z., Halford, K.J., and Hanson, R.T., 2009, Revised Multi-Node Well (MNW2) package for MODFLOW ground-water flow model: U.S. Geological Survey Techniques and Methods, book 6, chap. A30, 67 p., *http://pubs.usgs.gov/tm/tm6a30/*.

Langevin, C.D., Thorne, D.T., Jr., Dausman, A.M., Sukop, M.C., and Guo, Weixing, 2008, SEAWAT version 4—A computer program for simulation of multi-species solute and heat transport: U.S. Geological Survey Techniques and Methods, book 6, chap. A22, 39 p., *http://pubs.usgs.gov/tm/tm6a22/*.

Mehl, S.W., and Hill, M.C., 2005, MODFLOW-2005, The U.S. Geological Survey modular ground-water model—Documentation of shared node Local Grid Refinement (LGR) and the Boundary Flow and Head (BFH) Package: U.S. Geological Survey Techniques and Methods, book 6, chap. A12, 68 p., *http://pubs.usgs.gov/tm/2006/tm6a12/*.

Mehl, S.W., and Hill, M.C., 2007, MODFLOW-2005, The U.S. Geological Survey modular ground-water model—Documentation of the Multiple-Refined-Areas capability of Local Grid Refinement (LGR) and the Boundary Flow and Head (BFH) Package: U.S. Geological Survey Techniques and Methods, bk. 6, chap. A21, 13 p., *http://pubs.usgs.gov/tm/2007/06A21*.

Niswonger, R.G., Panday, Sorab, and Ibaraki, Motomu, 2011, MODFLOW–NWT, A Newton formulation for MODFLOW–2005: U.S. Geological Survey Techniques and Methods, book 6, chap. A37, 44 p., *http://pubs.usgs.gov/tm/tm6a37/*.

Niswonger, R.G., and Prudic, D.E., 2006, Documentation of the Streamflow-Routing (SFR2) Package to include unsaturated flow beneath streams—A modification to SFR1 (ver. 1.1, April 2006): U.S. Geological Survey Techniques and Methods, book 6, chap. A13, 48 p., *http://pubs.usgs.gov/tm/2006/tm6A13/*.

Schmid, Wolfgang, Hanson, R.T., Maddock, Thomas, III, Leake, S.A., 2006, User guide for the Farm Process (FMP1) for the U.S. Geological Survey's modular three-dimensional finite-difference ground-water flow model, MODFLOW-2000: U.S. Geological Survey Techniques and Methods 6-A17, 127 p., *http://pubs.usgs.gov/tm/2006/tm6A17/*.

Schmid, Wolfgang, and Hanson, R.T., 2009, The Farm Process Version 2 (FMP2) for MODFLOW–2005—Modifications and upgrades to FMP1: U.S. Geological Survey Techniques and Methods 6-A32, 102 p., *http://pubs.usgs.gov/tm/tm6a32/*.

Shoemaker, W.B., Kuniansky, E.L., Birk, S., Bauer, S., and Swain, E.D., 2007, Documentation of a Conduit Flow Process (CFP) for MODFLOW-2005: U.S. Geological Survey Techniques and Methods, book 6, chap. A24, 50 p., *http://water.usgs.gov/ogw/cfp/cfp.htm*.

Publishing support provided by:
Denver Publishing Service Center

For more information concerning this publication, contact:
Director, USGS Colorado Water Science Center
Box 25046, Mail Stop 415
Denver, CO 80225
(303) 236-4882

Or visit the Colorado Water Science Center Web site at:
http://co.water.usgs.gov/

Appendix 1: Example of Use of GWM–VI with the Dewater Problem

Input files and output for the Dewater sample problem described in Ahlfeld and others (2005), adapted for use with GWM–VI, are discussed in this section. All files referenced here can be found in the sample problems distributed with GWM–VI. The problem domain is illustrated in figure 1-1. Details of the model and the management problem are as described in Ahlfeld and others (2005). Briefly, the object of the management problem is to minimize the cost of withdrawing groundwater to lower heads to a suitable elevation in the construction area identified in figure 1-1, so that footings for a construction site can be installed. Only the linear formulation of the Dewater problem is presented in this appendix.

As with the original sample problem, the MODFLOW input files include a Name file. However, the contents of the Name file used for the GWM–VI sample problem are somewhat different from that used for the original sample problem to account for the additional files that are needed for the GWM–VI run. The Name file (dewater.nam) is to be read by MODFLOW-2005 and appears as follows:

```
LIST 10 dewater.lst
DIS 11 ..\data\dewater.dis
BAS6 12 ..\data\dewater.ba6
BCF6 13 ..\data\dewater.bc6
PCG 14 ..\data\dewater.pcg
OC 15 ..\data\dewater.oc
WEL 16 dewater.wel
data(binary) 50 dewaterhd.bin REPLACE
```

As with the Name file in the original problem, a Discretization (DIS) file, a Basic Package (BAS6) file, a Block-Centered Flow Package (BCF6) file, and a Preconditioned Conjugate-Gradient Package (PCG) file are included. The contents of these files are unchanged from the original problem. Unlike the Name file in the original sample problem, the Name file does not include the GWM keyword.

Three additional files are added to the Name file to meet the requirements of GWM–VI. A data(binary) file is identified. This file contains the heads calculated by MODFLOW. An

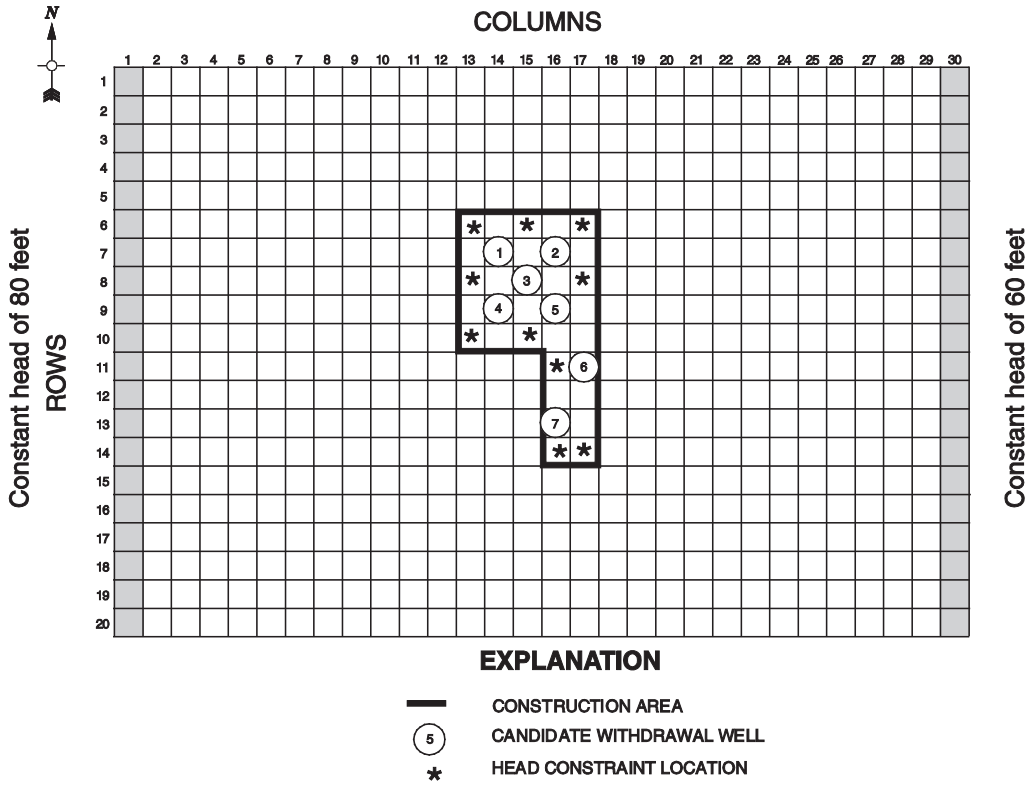

Figure 1-1. Model setup for Dewater example (modified from fig. 7 of Ahlfeld and others, 2005).

Output Control (OC) file is added to provide specific output instructions as follows:
OC file (dewater.oc):

```
HEAD PRINT FORMAT 20
head save unit 50
compact budget
PERIOD 1 STEP 1
PRINT HEAD
save budget
save head
```

Note that the OC file instructs MODFLOW to save heads to unit 50, which is identified in the Name file as a binary data file. A Well Package (WEL) file is added to provide a name for a file to which decision-variable flows can be written. The Dewater sample problem does not have any unmanaged wells, so the Well file indicates zero unmanaged wells.
WEL file (dewater.wel):

```
# Original WEL file for dewater case (no unmanaged wells)
0 0 # Item 1: MXACTW IWELCB
0 0 # Item 5: ITMP NP, stress period 1
```

The contents of the DECVAR, OBJFNC, VARCON, HEDCON, and SOLN files are unchanged from those documented in Ahlfeld and others (2005). As described in the "GWM–VI File Input Instructions" section of this report, the entries for CONTROL and NAM files in the main GWM–VI input file are new requirements. The CONTROL file, among other things, controls the mode (either serial or parallel) in which GWM–VI will run. The NAM file is the MODFLOW Name file, and is shown above.

Files for running the linear formulation of the Dewater management problem in either serial or parallel mode are provided in the distribution. Like all the example problems, the Dewater problem includes a batch file, which can be used to invoke GWM–VI in either serial or parallel model. Serial model is invoked by providing the word "serial" as a command-line option following the batch file name on the command line. Parallel mode is invoked when the batch file is run without a command-line option. Execution of GWM–VI in both modes is explained in the following sections.

Running the Dewater Problem in Serial Mode

The input file dewater_serial.gwm is provided for running the Dewater problem in serial mode. It contains:

```
OUT dewater.gwmout.serial
DECVAR ..\data\dewater.decvar
OBJFNC ..\data\dewater.objfnc
VARCON ..\data\dewater.varcon
HEDCON ..\data\dewater.hedcon
SOLN ..\data\dewater.soln
CONTROL ..\data\dewater_serial.ctrl
NAM ..\data\dewater.nam
```

The CONTROL file for the serial-mode run, dewater_serial.ctrl, contains:

```
BEGIN Options
 Verbose = 3
 MessageFile = dewater_messages.txt
END Options
BEGIN Simulation
 SimCommand = "..\..\bin\mf2005dbl ..\data\dewater.nam"
END Simulation
BEGIN Model_Command_Lines
 command = ..\data\dewater_cmr.bat
 commandid = Dewater_cmr
END Model_Command_Lines
BEGIN Parallel_Control
 parallel=false
END Parallel_Control
```

The Simulation input block includes the SimCommand that gives the path for the executable to be used for MODFLOW runs. The Command keyword in the Model_Command_Lines block references a DOS batch file that invokes the MMProc executable file.

Note that the default value of "parallel" in the Parallel_Control input block is "false" and that the Parallel_Control input block is optional. If the Parallel_Control input block is omitted from the dewater_serial.ctrl file, GWM–VI will execute in serial mode, as it will with the CONTROL file shown above.

If dewater.bat is invoked with the word "serial" as a command-line option following the batch-file name on the command line, the Dewater problem runs in serial mode, and, to avoid ambiguity, the output file "dewater.gwmout" is copied to a file with the name "dewater.gwmout.serial." The main output file for the serial-mode run of the Dewater problem follows. (Note that the file extends across several pages.):

```
                                GWM-VI
   U.S. GEOLOGICAL SURVEY GROUNDWATER MANAGEMENT VERSION INDEPENDENT PROGRAM
                Version  1.0.0



OPENING GWM FILE FOR GWM1 -- GROUNDWATER MANAGEMENT PROCESS
INPUT READ FROM UNIT   7

# DEWATER Sample Problem, GWM file - Serial


OPENING DECISION-VARIABLE FILE ON UNIT   12:
..\data\dewater.decvar

#DEWATER Sample Problem, DECVAR file
#August 14, 2006

NO. OF FLOW-RATE DECISION VARIABLES (NFVAR)       7
NO. OF EXTERNAL DECISION VARIABLES (NEVAR):       0
BINARY VARIABLES ARE NOT ACTIVE.

FLOW-RATE VARIABLES: WEL-TYPE
                                                    FRACTION
 NUMBER     NAME       TYPE      LAY   ROW   COL   OF FLOW
-----------------------------------------------------------
   1      Q1         WITHDRAWAL     1    7   14    1.0000
  AVAILABLE IN STRESS PERIODS: 1

   2      Q2         WITHDRAWAL     1    7   16    1.0000
  AVAILABLE IN STRESS PERIODS: 1

   3      Q3         WITHDRAWAL     1    8   15    1.0000
  AVAILABLE IN STRESS PERIODS: 1

   4      Q4         WITHDRAWAL     1    9   14    1.0000
  AVAILABLE IN STRESS PERIODS: 1

   5      Q5         WITHDRAWAL     1    9   16    1.0000
  AVAILABLE IN STRESS PERIODS: 1

   6      Q6         WITHDRAWAL     1   11   17    1.0000
  AVAILABLE IN STRESS PERIODS: 1

   7      Q7         WITHDRAWAL     1   13   16    1.0000
  AVAILABLE IN STRESS PERIODS: 1


     606 BYTES OF MEMORY ALLOCATED TO STORE DATA FOR DECISION VARIABLES

CLOSING DECISION-VARIABLE FILE
```

```
OPENING OBJECTIVE-FUNCTION FILE ON UNIT   12:
..\data\dewater.objfnc

#DEWATER Sample Problem, OBJFNC file
#February 20, 2005

OBJECTIVE TYPE: MIN   FUNCTION TYPE: WSDV

NO. OF FLOW-RATE DECISION VARIABLES IN OBJECTIVE FUNCTION (NFVOBJ):    7
NO. OF EXTERNAL DECISION VARIABLES IN OBJECTIVE FUNCTION (NEVOBJ):     0
NO. OF BINARY DECISION VARIABLES IN OBJECTIVE FUNCTION (NBVOBJ):       0
NO. OF STATE VARIABLES IN OBJECTIVE FUNCTION (NSVOBJ):                 0

OBJECTIVE FUNCTION: (excluding flow duration terms)

MIN    +  1.00E+00 Q1         +  1.00E+00 Q2         +  1.00E+00 Q3
       +  1.00E+00 Q4         +  1.00E+00 Q5         +  1.00E+00 Q6
       +  1.00E+00 Q7

      28 BYTES OF MEMORY ALLOCATED TO STORE DATA FOR OBJECTIVE-FUNCTION

CLOSING OBJECTIVE-FUNCTION FILE


OPENING DECISION-VARIABLE CONSTRAINTS FILE ON UNIT   12:
..\data\dewater.varcon

#DEWATER Sample Problem, VARCON file
#February 20, 2005

FLOW RATE VARIABLES:
                      MINIMUM        MAXIMUM        REFERENCE
 NUMBER   NAME        FLOW RATE      FLOW RATE      FLOW RATE
-------------------------------------------------------------
   1      Q1          0.000E+00      2.000E+04      0.000E+00
   2      Q2          0.000E+00      2.000E+04      0.000E+00
   3      Q3          0.000E+00      2.000E+04      0.000E+00
   4      Q4          0.000E+00      2.000E+04      0.000E+00
   5      Q5          0.000E+00      2.000E+04      0.000E+00
   6      Q6          0.000E+00      2.000E+04      0.000E+00
   7      Q7          0.000E+00      2.000E+04      0.000E+00

CLOSING DECISION-VARIABLE CONSTRAINTS FILE


OPENING HEAD CONSTRAINTS FILE ON UNIT   12:
..\data\dewater.hedcon

#DEWATER Sample Problem, HEDCON file
#February 20, 2005
```

```
HEAD CONSTRAINTS:
                    MNW WELLID or          RIGHT-HAND    STRESS
NUMBER  NAME        LAY  ROW  COL  TYPE       SIDE       PERIOD
------------------------------------------------------------------
    1   b_01          1    6   13   <    5.0000E+01         1
    2   b_02          1    6   15   <    5.0000E+01         1
    3   b_03          1    6   17   <    5.0000E+01         1
    4   b_04          1    8   13   <    5.0000E+01         1
    5   b_05          1    8   17   <    5.0000E+01         1
    6   b_06          1   10   13   <    5.0000E+01         1
    7   b_07          1   10   15   <    5.0000E+01         1
    8   b_08          1   11   16   <    5.0000E+01         1
    9   b_09          1   14   16   <    5.0000E+01         1
   10   b_10          1   14   17   <    5.0000E+01         1

     700 BYTES OF MEMORY ALLOCATED TO STORE DATA FOR HEAD CONSTRAINTS

CLOSING HEAD CONSTRAINTS FILE


OPENING SOLUTION FILE ON UNIT   12:
..\data\dewater.soln

#DEWATER Sample Problem, SOLN file
#February 20, 2005

SOLNTYP IS LP: GWM WILL COMPLETE A SINGLE ITERATION OF THE LINEAR PROBLEM.

IRM EQUALS 2: RESPONSE MATRIX WILL BE CALCULATED BY GWM
BUT NOT WRITTEN TO FILE

MAXIMUM NUMBER OF LP ITERATIONS:               1000
MAXIMUM NUMBER OF BRANCH AND BOUND ITER:       2000

PERTURBATION VALUE:                          0.50D+00

MAXIMUM NUMBER OF PERTURBATION ATTEMPTS:         10
PERTURBATION ADJUSTMENT FACTOR (PGFACT):    0.50000

OUTPUT FROM BRANCH-AND-BOUND ALGORITHM WILL NOT BE PRINTED.

CRITMFC SET TO                               0.000D+00
GWM WILL ACCEPT FLOW PROCESS RESULTS THAT MEET GWF
CONVERGENCE CRITERIA

BASE PUMPING RATES TAKEN FROM FVREF SPECIFIED IN VARCON INPUT FILE

    PROBLEM SIZE

 NUMBER OF VARIABLES (INCLUDING SLACKS)          17
 NUMBER OF CONSTRAINT EQUATIONS                  10


    6439 BYTES OF MEMORY ALLOCATED FOR RESPONSE MATRIX ALGORITHM
```

```
 CLOSING SOLUTION AND OUTPUT FILE


 Reading input from file: ..\data\dewater_serial.gwm
 CONTROL file: ..\data\dewater_serial.ctrl
 NAM file: ..\data\dewater.nam
-----------------------------------------------------------------
               Solution Algorithm
-----------------------------------------------------------------
  Begin Solution Algorithm
    Running Base Flow Process Simulation
      Status of Simulation-Based Constraints
      Constraint Type        Name       Status   Distance To RHS
      ---------------        ----       ------   ---------------
      Head Bound             b_01       Not Met       2.1724E+01
      Head Bound             b_02       Not Met       2.0345E+01
      Head Bound             b_03       Not Met       1.8966E+01
      Head Bound             b_04       Not Met       2.1724E+01
      Head Bound             b_05       Not Met       1.8966E+01
      Head Bound             b_06       Not Met       2.1724E+01
      Head Bound             b_07       Not Met       2.0345E+01
      Head Bound             b_08       Not Met       1.9655E+01
      Head Bound             b_09       Not Met       1.9655E+01
      Head Bound             b_10       Not Met       1.8966E+01

  Distance to RHS is the absolute value of the difference between the
    the right hand side of the constraint and the left side of the
    constraint evaluated using the current set of decision variable values.

    Calculating Response Matrix
      Perturb Flow Variable    1
       By Perturbation Value: -1.000000E+04
      Perturb Flow Variable    2
       By Perturbation Value: -1.000000E+04
      Perturb Flow Variable    3
       By Perturbation Value: -1.000000E+04
      Perturb Flow Variable    4
       By Perturbation Value: -1.000000E+04
      Perturb Flow Variable    5
       By Perturbation Value: -1.000000E+04
      Perturb Flow Variable    6
       By Perturbation Value: -1.000000E+04
      Perturb Flow Variable    7
       By Perturbation Value: -1.000000E+04

      Average Number of Significant Digits in Matrix  1.011429E+01

    Solving Linear Program
    Feasible Solution Found
  Optimal Solution Found
```

```
------------------------------------------------------------------------
               Groundwater Management Solution
------------------------------------------------------------------------

       OPTIMAL SOLUTION FOUND

       OPTIMAL RATES FOR EACH FLOW VARIABLE
       ------------------------------------
Variable           Withdrawal          Injection           Contribution
Name               Rate                Rate                To Objective
----------         --------------      ------------        ------------
 Q1                1.077390E+03                            1.077390E+06
 Q2                7.823877E+01                            7.823877E+04
 Q3                0.000000E+00                            0.000000E+00
 Q4                7.689506E+02                            7.689506E+05
 Q5                0.000000E+00                            0.000000E+00
 Q6                0.000000E+00                            0.000000E+00
 Q7                9.410751E+02                            9.410751E+05
                   ------------        ------------        ------------
TOTALS             2.865655E+03        0.000000E+00        2.865655E+06


       OBJECTIVE FUNCTION VALUE                            2.865655E+06

       BINDING CONSTRAINTS
Constraint Type        Name     Status      Shadow Price
---------------        ----     ------      ------------
Head Bound             b_01     Binding     -2.7273E+04
Head Bound             b_03     Binding     -3.2593E+04
Head Bound             b_06     Binding     -3.1185E+04
Head Bound             b_10     Binding     -5.1544E+04

  Binding constraint and range analysis values are determined from the linear
    program and based on the response matrix approximation of the flow-process.

       RANGE ANALYSIS


       Constraint Ranges

  Lower/Upper Bound are the values of the RHS beyond which basis will change.
    Leaving is the variable which will leave the basis.
    Entering is the variable which will enter the basis.
    If the entering or leaving variable is a constraint name,
      then the constraint slack variable is active

Constraint                Original      Lower/Upper
Name        Slack         RHS           Bound            Entering    Leaving
----------  ----------    ----------    ----------       ----------  ----------
b_01        0.0000E+00    5.0000E+01    4.9477E+01       b_03        Q2
                                        5.3228E+01       b_01        b_04

b_02        2.0745E+00    5.0000E+01    4.7926E+01       b_01        b_02
                                          Infinity       ----- No Change -----
```

```
b_03        0.0000E+00    5.0000E+01    4.3065E+01       b_01        Q1
                                        5.0317E+01       b_03        Q2

b_04        2.0528E+00    5.0000E+01    4.7947E+01       b_01        b_04
                                          Infinity      ----- No Change -----

b_05        1.1167E+00    5.0000E+01    4.8883E+01       Q3          b_05
                                          Infinity      ----- No Change -----

b_06        0.0000E+00    5.0000E+01    4.7939E+01       b_03        Q2
                                        5.2635E+01       b_06        Q4

b_07        2.6182E+00    5.0000E+01    4.7382E+01       Q3          b_07
                                          Infinity      ----- No Change -----

b_08        1.8584E+00    5.0000E+01    4.8142E+01       Q6          b_08
                                          Infinity      ----- No Change -----

b_09        1.0158E+00    5.0000E+01    4.8984E+01       b_10        b_09
                                          Infinity      ----- No Change -----

b_10        0.0000E+00    5.0000E+01    4.7205E+01       b_03        Q2
                                        5.0850E+01       b_10        b_09



      Objective-Function Coefficient Ranges

  Lower/Upper Bound are the values of the coefficients beyond which basis will change.
    Leaving is the variable which will leave the basis.
    Entering is the variable which will enter the basis.
    If the entering or leaving variable is a constraint name,
      then the constraint slack variable is active
    Basic variables are shown with zero reduced cost

Variable    Reduced       Original      Lower/Upper
Name        Cost          Coefficient   Bound            Entering    Leaving
----------  ----------    ----------    ----------       ----------  ----------
Q1          0.0000E+00    1.0000E+03    9.1368E+02       b_01        Q2
                                        1.0669E+03       Q3          Q2

Q2          0.0000E+00    1.0000E+03    8.6811E+02       b_03        Q1
                                        1.0438E+03       Q3          Q2

Q3          1.5770E+01    1.0000E+03    9.8423E+02       Q3          Q2
                                          Infinity      ----- No Change -----

Q4          0.0000E+00    1.0000E+03    8.9312E+02       b_06        Q2
                                        1.0471E+03       Q3          Q2

Q5          4.4085E+01    1.0000E+03    9.5592E+02       Q5          Q2
                                          Infinity      ----- No Change -----

Q6          7.4018E+01    1.0000E+03    9.2598E+02       Q6          Q2
                                          Infinity      ----- No Change -----
```

```
Q7          0.0000E+00    1.0000E+03    6.7387E+02       b_10        Q2
                                        1.1286E+03       Q6          Q2

------------------------------------------------------------
        Final Flow Process Simulation
------------------------------------------------------------
  Running Final Flow Process Simulation
    using Optimal Flow Variable Rates

      Status of Simulation-Based Constraints
        Using Optimal Flow Rate Variable Values

                                          Simulated    Specified
                                           By Flow        in
      Constraint Type        Name          Process    Constraints   Difference
      ---------------        ----         ----------   ----------   ----------
      Head Bound             b_01         5.0000E+01 < 5.0000E+01   9.4796E-10
      Head Bound             b_02         4.7926E+01 < 5.0000E+01  -2.0745E+00
      Head Bound             b_03         5.0000E+01 < 5.0000E+01   1.9886E-09
      Head Bound             b_04         4.7947E+01 < 5.0000E+01  -2.0528E+00
      Head Bound             b_05         4.8883E+01 < 5.0000E+01  -1.1167E+00
      Head Bound             b_06         5.0000E+01 < 5.0000E+01  -9.0949E-10
      Head Bound             b_07         4.7382E+01 < 5.0000E+01  -2.6182E+00
      Head Bound             b_08         4.8142E+01 < 5.0000E+01  -1.8584E+00
      Head Bound             b_09         4.8984E+01 < 5.0000E+01  -1.0158E+00
      Head Bound             b_10         5.0000E+01 < 5.0000E+01  -2.8596E-09

  Difference is computed by subtracting right hand side of the constraint
    from the left side of the constraint.
  Precision limitations and nonlinear response may cause the
    values of the binding constraints computed directly by the flow process
    to differ from those computed using the linear program.

 Run end date and time (yyyy/mm/dd hh:mm:ss): 2013/03/28 14:12:12
 Elapsed run time:  2.155 Seconds
```

Running the Dewater Problem in Parallel Mode

The main input file for running the Dewater problem in parallel mode is dewater_pll.gwm, which differs from dewater_serial.gwm only in the CONTROL entry. It contains:

```
OUT dewater.gwmout.parallel
DECVAR ..\data\dewater.decvar
OBJFNC ..\data\dewater.objfnc
VARCON ..\data\dewater.varcon
HEDCON ..\data\dewater.hedcon
SOLN ..\data\dewater.soln
CONTROL ..\data\dewater_pll.ctrl
NAM ..\data\dewater.nam
```

The CONTROL file for the parallel-mode run, dewater_pll.ctrl, contains:

```
BEGIN Options
  Verbose = 3
  MessageFile = dewater_messages.txt
END Options

BEGIN Simulation
  SimCommand = "..\..\bin\mf2005dbl ..\data\dewater.nam"
END Simulation

BEGIN Model_Command_Lines
  command = ..\data\dewater_cmr.bat
  commandid = Dewater_cmr
END Model_Command_Lines

BEGIN Parallel_Control
  parallel=true wait=0.01
  VerboseRunner=4
  AutoStopRunners = true
  TimeOutFactor = 4.0
END Parallel_Control

BEGIN Parallel_Runners table
  nrow=2 ncol=3 columnlabels
  RunnerName RunnerDir RunTime
  Runner1 ..\runner1\ 2.0
  Runner2 ..\runner2\ 2.0
END Parallel_Runners
```

Before invoking GWM–VI to run the Dewater problem in parallel mode, the runner program JRunnerM is invoked in each of the runner directories. The Dewater example is then run by executing GWM–VI with input file dewater_pll.gwm. When invoked without a command-line option, dewater.bat starts JRunnerM in the runner directories and then invokes GWM–VI in parallel mode. After GWM_VI terminates, dewater.bat copies the output file (dewater.gwmout) to a file with the name "dewater.gwmout.parallel." The main output file for the parallel-mode run of the Dewater problem is nearly identical to the output for the serial-mode run shown above. The only meaningful differences are in the names of input files and in text echoed from the input file, where "serial" is replaced by "parallel."

Appendix 2: Programmers' Guide to GWM–VI

GWM–VI is a model-analysis application, in that it is a tool designed to perform a specific type of analysis (optimization) with a specific process-model code (MODFLOW). The JUPITER API (Banta and others, 2006) is a set of Fortran-90 modules containing data structures, subroutines, and functions specifically designed to facilitate development of model-analysis applications. The primary goals in developing the GWM–VI suite of programs were to add support for a parallel-processing capability to GWM and to enable GWM to work with any MODFLOW-based executable that meets certain conditions. The JUPITER API is well suited to the requirements of GWM. To take advantage of the parallel-processing capability of the JUPITER API, the optimization-related code of the GWM Process as implemented in GWM-2005 was adapted into a structure consistent with the requirements of the JUPITER API; the result is the main program unit of GWM–VI.

The main program unit of GWM–VI has the structure of a model-analysis application designed according to the principles of the JUPITER API; in figure 2-1 the boxes with white background represent GWM–VI. Tasks defined in the JUPITER API (Banta and others, 2006) are numbered in figure 2-1. The ellipse with a gray background represents user-prepared input files for GWM–VI. Ellipses with a blue background represent JUPITER API template and instruction files generated by GWM–VI. Black ellipses with white text represent the input files, executable file, and output files that make up a MODFLOW simulation.

Each task shown in figure 2-1 has one or more subroutine calls in GWM–VI. The tasks, their purposes, and the corresponding subroutines are shown in table 2-1.

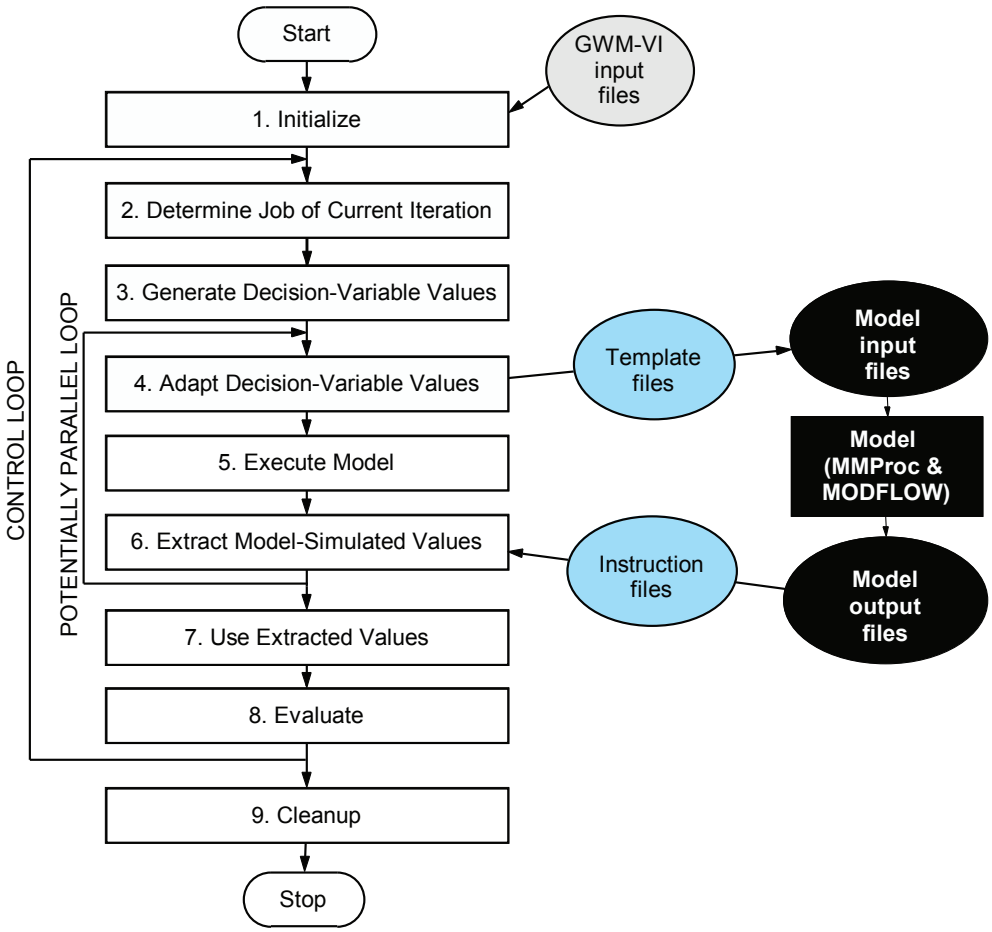

Figure 2-1. Schematic flowchart for GWM–VI.

Table 2-1. JUPITER API tasks and GWM-VI and JUPITER API subroutines that perform them. Names in bold are subroutines written or adapted for GWM-VI; names beginning with GWM1 are subroutines from GWM-2005; non-bold names are subroutines of the JUPITER API (Banta and others, 2006)

JUPITER API Task	Purpose	Subroutine(s) called from GWM-VI main program unit to perform task	
Initialize	Read input; allocate and populate arrays	**MAIN_INI**, **GWM1BAS3AR**, **READ_NAME_FILE_DIS**, BAS_INI_MODELEXEC, **WRMMPROCIN**, **MAIN_WRITE_MF_STATUS_JIF**, **MAIN_INI_PARS**, **MAIN_INI_DEP**, DEP_INI_ALLOC, DEP_INI_STORE, MIO_INI_ALLOC, **MAIN_WRITE_MIF_BLOCK**, MIO_INI_INPUTFILES, **MAIN_WRITE_MOF_BLOCK**, MIO_INI_OUTPUTFILES, MIO_INI_TEMPLATE, MIO_INI_INSTRUCT1, MIO_INI_INSTRUCTALLOC, MIO_INI_INSTRUCT2, MIO_INI_DIMENSION, MIO_INI_ARRAYS, **PLLM_INI_DISPATCHER**	
Determine job of current iteration	Determine job of current iteration of control loop	**MAIN_DEF**	
Generate decision-variable values	Perturb decision variables or, for a nonlinear management formulation, generate updated decision-variable values	**GWM1RMS3PL**, BAS_GEN, **MAIN_GEN**, **DECVARS_TO_PVAL**	
Adapt decision-variable values	Use current values of decision variables and template files to generate model-input file(s)	Serial Processing	Parallel Processing
		GWM1RMS3PP, MIO_ADA_WRITEFILES	BAS_EXE_SELECT, **PLLGWM_MAKE_RUNS**
Execute model	Execute the model (using current values of decision variables)	BAS_EXE_SELECT, BAS_EXE	
Extract model-simulated values	Use instruction files to extract model-calculated values related to hydraulic-head and streamflow constraints	MIO_EXT	
Use extracted values	Calculate response matrix or, for nonlinear formulation, update decision-variable values	**MAIN_UEV, GWM1RMS3FP**	
Evaluate	Evaluate effectiveness of solution to management problem; write results of analyses to output file	**GWM1RMS3FM, GWM1RMS3AP, GWM1RMS3OT**	
Cleanup	Deallocate arrays and close files	**CLEAN_UP**, BAS_CLN, DEP_CLN, **PLLM_CLN**, **MAIN_CLN**	

GWM–VI, like GWM-2005, adjusts its behavior based on results of perturbation simulations made to populate the response matrix. The Parallel-Processing Module of the JUPITER API does not support such intervention. To implement this capability, GWM–VI includes a modified parallel-processing implementation, which is derived from and replaces the Parallel-Processing Module of the JUPITER API. The new module includes all the capabilities of the JUPITER Parallel-Processing Module documented by Banta and others (2006). In addition, the GWM–VI parallel-processing module includes code to evaluate the simulation results and adapt GWM–VI behavior accordingly. In particular, response precision and model non-convergence and dewater status as reported in the modflow.status file generated by MMProc are evaluated, and values of flow-rate decision variables are adjusted as needed to obtain usable results using procedures from the GWM Response Matrix Solution (RMS) Package (Ahlfeld and others, 2005).

Of the 44 subroutines listed in table 2-1, 20 are part of the JUPITER API and documented by Banta and others (2006). The other 24 subroutines are described briefly in table 2-2.

GWM–VI uses the same GWM Packages as GWM-2005. As a result, both GWM-2005 and GWM–VI are compiled with code from MODFLOW-2005, version 1.10. GWM–VI uses this code to access data from the GLOBAL and GWFBASMODULE modules of MODFLOW's Basic Package, and from the GWFMNW2MODULE module of the MNW2 Package (Konikow and others, 2009). GWM–VI also invokes the GWF2MNW27RP subroutine to read the MNW2 input file when MNW2-based decision variables are used. If a model to be analyzed by GWM–VI uses more recent versions of either the Basic Package or the MNW2 Package, it may be necessary to make corresponding updates to the GWM–VI code and recompile the programs to allow GWM–VI to function correctly.

Table 2-2. Subroutines written or adapted for GWM–VI and their purposes.

Subroutine	Purpose
MAIN_INI	Read MODFLOW DIS file and store required dimensions and array data; read Options and Simulation input blocks from Control file.
GWM1BAS3AR	Open GWM input and output files; read GWM file; call routines to read DECVAR, OBJFNC, VARCON, SUMCON, HEDCON, STRMCON, and SOLN files
READ_NAME_FILE_DIS	Read MODFLOW Name file to get pathname of Discretization (DIS) file, then open DIS file and read model discretization data
WRMMPROCIN	Create three files: a JUPITER API template file for generating the mmproc.in file, another template file for generating the mnw2_input.jtf file, and a JUPITER API instruction file for extracting values from the SimulatedValues.out file
MAIN_WRITE_MF_STATUS_JIF	Create a JUPITER API instruction file (modflow_status.jif) for extracting values from the modflow. status file
MAIN_INI_PARS	Generate names for parameters used by JUPITER subroutines from flow-rate decision variables
MAIN_INI_DEP	Determine number of dependent variables
MAIN_WRITE_MIF_BLOCK	Write a Model_Input_Files input block to a scratch file
MAIN_WRITE_MOF_BLOCK	Write a Model_Output_Files input block to a scratch file
PLLM_INI_DISPATCHER	Read Parallel_Control and Parallel_Runners input blocks from Control file; initialize parallel-processing module of GWM-VI
MAIN_DEF	Define job of current iteration of control loop
GWM1RMS3PL	Prepare perturbation loop controls, size of perturbations
MAIN_GEN	Generate parameter set with perturbed value
DECVARS_TO_PVAL	Utility to distribute decision-variable values to parameter-value array used by JUPITER subroutines
GWM1RMS3PP	Prepare each individual perturbation
PLLGWM_MAKE_RUNS	Make multiple perturbation runs in parallel
MAIN_UEV	Use extracted values to populate head-constraint state array and other variables
GWM1RMS3FP	Evaluate perturbation results; test for failure
GWM1RMS3FM	Formulate the GWM problem
GWM1RMS3AP	Solve the GWM problem
GWM1RMS3OT	Write GWM solution output
CLEAN_UP	Restore original MNW2 Package input file to original file name
PLLM_CLN	Deallocate arrays of parallel-processing module of GWM-VI
MAIN_CLN	Close an output file and deallocate arrays used by GWM-VI