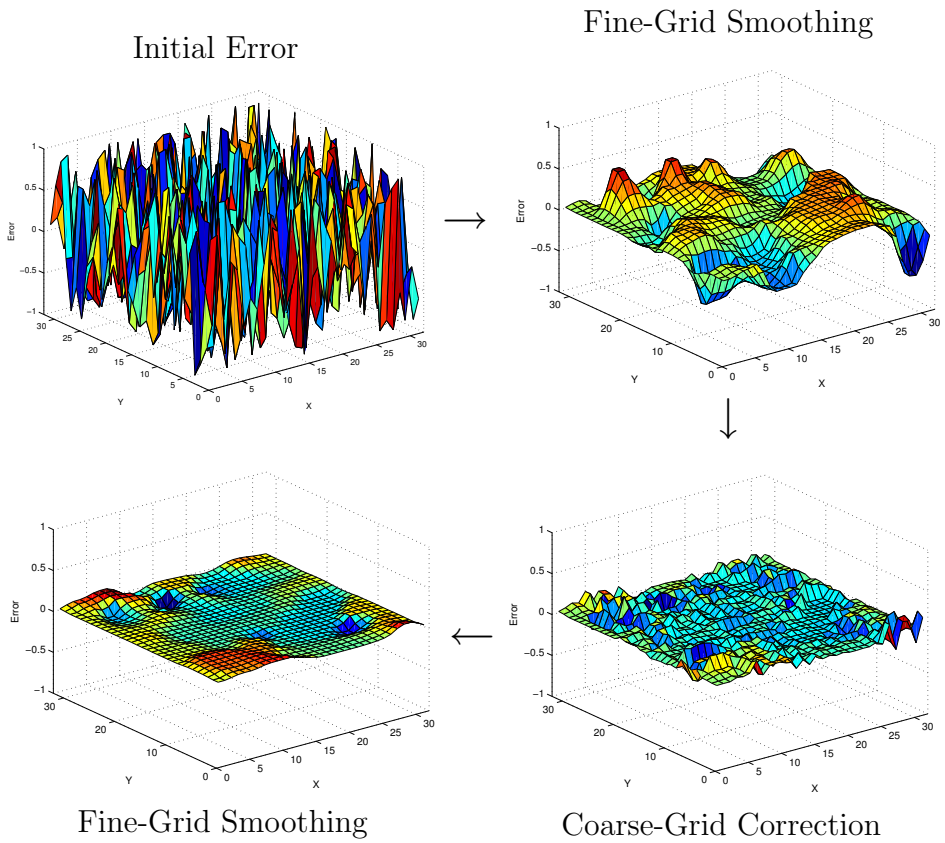
MODFLOW-2000, The U.S. Geological Survey Modular Ground-Water Model – GMG Linear Equation Solver Package Documentation



U.S. GEOLOGICAL SURVEY
Open-File Report 2004-1261

The U.S. Geological Survey Modular Ground-Water Model – GMG Linear Equation Solver Package Documentation

John D. wilson Richard L. Naff

U.S. GEOLOGICAL SURVEY
Open-File Report 2004-1261

Denver, Colorado
2004

U.S. DEPARTMENT OF THE INTERIOR
GALE A. NORTON, *Secretary*

U.S. GEOLOGICAL SURVEY
Charles G. Groat, *Director*

The use of brand, trade™, or firm names in this report is for identification purposes only and does not constitute endorsement by the U.S. Geological Survey.

For additional information write to:

Chief, Branch of Regional Research
U.S. Geological Survey
Box 25046, Mail Stop 418
Denver Federal Center
Denver, CO 50225-0046

Copies of this report can be purchased from:

U.S. Geological Survey
Branch of Information Services
Box 25286
Denver, CO 50225-0425

Preface

This report describes the linear equation solver based in cell-centered multigrid and linked with the U.S. Geological Survey (USGS) MODFLOW-2000 (MF2K) computer program. The computer program implementing the linear equation solver is named GMG and is written by the authors of this report. The GMG solver is based on a method known in the literature as geometric multigrid; the implementation herein is as a preconditioner for the conjugate gradient method.

Comparisons are made between the GMG solver and the algebraic multigrid (AMG) solver. The AMG solver was developed by the German National Research Center for Information Technology (GMD) and is linked to MF2K using the LINK-AMG (LMG) package described in Mehl and Hill (2001). The AMG source code is named AMG1R6 and has a release number 1.6, date July, 2002. The AMG code is a freeware program but has some licensing restrictions; currently (2004), use of this code by the USGS is intended for research purposes only. More recent developments in AMG software are available from GMD but not as freeware programs. The commercial versions of AMG are likely to be more sophisticated and require less computer memory than the freeware versions.

The performance of GMG has been tested on a variety of ground-water models. It is possible, and even likely, that GMG will not be sufficiently robust for all future ground-water modeling problems. In the event that a ground-water modeling problem cannot be solved using the GMG package in MF2K with sufficient efficiency, then the authors of this report should be notified and given a detailed description of the problem.

Contents

Abstract **1**

Introduction **1**

The GMG Solver and MF2K **2**
Finite-Difference Matrix . . . 3
Specified Heads and Inactive Cells . . . 4
Linear Solution . . . 6
Nonlinear Solution . . . 8
Convergence Criteria . . . 9

Multigrid **9**
Smoothers . . . 10
ILU Smoother . . . 11
Coarse-Grid Correction . . . 13
ν-Cycle . . . 19

Comparison of GMG with AMG **21**

Input Instructions for the GMG Solver **23**
Adaptive Damping Example . . . 26
Semi-Coarsening Example . . . 29
ILU PCG Example . . . 31

Description of GMG Interface **32**
Module GMG1ALG . . . 34
Module GMG1AP . . . 34

Description of GMG Library **35**
Vector Library . . . 36
Solver Library . . . 39
PCG Operator . . . 39
MG Operator . . . 40
CCFD Library . . . 42
MF2KGMG Library . . . 43

List of Figures

1. Smoothing the error . . . 14
2. Coarse-Grid Correction . . . 18
3. Grid Schedules . . . 20
4. Fractures with entrapped NAPL . . . 23

List of Tables

1. Computational differences between GMG and AMG 21

MODFLOW-2000, THE U.S. GEOLOGICAL SURVEY MODULAR GROUND-WATER MODEL – GMG LINEAR EQUATION SOLVER PACKAGE DOCUMENTATION

By John D. Wilson and Richard L. Naff

Abstract

A geometric multigrid solver (GMG), based in the preconditioned conjugate gradient algorithm, has been developed for solving systems of equations resulting from applying the cell-centered finite difference algorithm to flow in porous media. This solver has been adapted to the U.S. Geological Survey ground-water flow model MODFLOW-2000. The documentation herein is a description of the solver and the adaptation to MODFLOW-2000.

Introduction

The U.S. Geological Survey (USGS) modular ground-water model MODFLOW-2000 (MF2K) is a computer program that simulates three-dimensional transient ground-water flow through a porous medium. For the purposes of this report, it is only necessary to consider the steady-state case. The elliptic partial-differential equation (PDE) of steady-state ground-water flow used in MF2K is given by

$$-\nabla \cdot \mathbf{K} \nabla p = W, \tag{1}$$

where p is the hydraulic head (L), $\mathbf{K}$ is a hydraulic conductivity tensor (L/T), and W is a source/sink term (T^{-1}). An approximate solution to equation (1) is obtained by using a finite-difference method (Harbaugh and others, 2000; McDonald and Harbaugh, 1988). The hydraulic heads are approximated at cell-centers. In this report, this method is referred to as the cell-centered finite difference (CCFD) method, and the finite-difference matrix that results from this method is referenced as the CCFD matrix. The modular structure of MF2K enables the user to select from a number of solvers to evaluate the CCFD matrix equation.

Multigrid methods are generally accepted as being among the fastest numerical methods for the solution of elliptic PDE's (Trottenberg and others, 2001; Briggs, 1987; McCormick, 1987; Wesseling, 1991). Algebraic multigrid (AMG) methods (Ruge and Stüben, 1987; Mehl and Hill, 2001) are especially robust in handling problems with large variations in the hydraulic-conductivity coefficient but are complex and can require large amounts of computer memory. Simpler multigrid methods, using significantly less memory, can be effective when used in conjunction with the preconditioned conjugate gradient (PCG) method. The solver was developed by the USGS based on a conjugate gradient method preconditioned by cell-centered multigrid and is referred to herein as the GMG solver. As opposed to AMG, the preconditioning in GMG is

based on a solver method known as geometric multigrid (Briggs, 1987; McCormick, 1987; Trottenberg and others, 2001; Wesseling, 1991; Hackbush, 1985); precepts of this method, as it applies to a CCFD matrix and the conjugate gradient scheme, are described in this report.

The GMG solver consists of two primary computer codes; the GMG library and the GMG interface. The GMG library is a collection of modules written in the C language. The GMG interface is a FORTRAN 95 code that links the GMG library with the MF2K program. The GMG library implements a generic operator and a collection of generic algorithms that are problem independent and can facilitate a modular approach for solving a variety of programming problems.

The purpose of this report is to document the GMG solver as applied to the evaluation of the linear equation in the MF2K program. The development of the GMG linear equation solver should be of interest to ground-water modelers who work with complex and memory-intensive simulations on workstations. The reduction in computer execution time, relative to other solvers using a comparable amount of memory, achieved by the GMG solver makes this solver an important and useful computer code for the USGS and the research community.

Detailed descriptions of the matrix equation and the multigrid preconditioner are given, along with a discussion of nonlinear iterations and convergence criteria. Comparisons between the GMG solver and the AMG solver for specific problems are presented so that relative efficiencies of the two methods can be evaluated. Algorithms are presented throughout this report to describe the methods used by the GMG solver; they coincide closely with the actual implementation of the GMG library.

For those readers interested only in using the GMG package in MF2K, a description of the input file is provided in the "Input Instructions for the GMG Solver" section. Sufficient information for each input item is provided to enable a user to implement this package without reading a description of the GMG method.

A description of the GMG interface program is provided. This description explains how the GMG library is linked to the MF2K program and will help in linking GMG to older versions of MODFLOW. It also provides a high-level description of the GMG method.

The GMG library implements the algorithms described in this report. Those familiar with the C programming language can gain an understanding of the generic algorithms by reading the "Description of the GMG Library" section. Modifications to the methods described in this report and future enhancements can be facilitated within the framework of these generic algorithms.

The GMG Solver and MF2K

The GMG solver is a computer code that consists of a library of functions written in the C language and an interface program written in FORTRAN. The interface program links MF2K to the GMG library. The GMG interface is activated by including file type "GMG" in the name file (Harbaugh and others, 2000, p. 43-44).

Hydraulic conductances, specified heads, inactive cells, and the right-hand side

are passed into the GMG interface. The GMG interface calls GMG library functions to assemble the CCFD matrix equation and preconditioner, approximate a solution using the PCG method, and determine convergence of the solution. Arrays of data and variables assembled by MF2K and passed into the GMG interface are referred to as internal. Arrays of data and variables used by the GMG solver that are not internal are allocated in the GMG library.

Throughout the remainder of this report, the number of columns, rows, and layers in the finite-difference grid are denoted as l, m, and n respectively. The finite-difference grid contains $N = l \times m \times n$ cells. The solution vector is of length N and represents values of the hydraulic head in each cell center. The heads and cell centers have an (i, j, k) ordering with $0 \leq i \leq l-1$, $0 \leq j \leq m-1$, and $0 \leq k \leq n-1$. This ordering is consistent with arrays in the C language where the indexing always starts at 0. A sequential ordering of the cells is given by the multi-index J, where $J = i + jl + klm$.

Remark 1. The indices for variables in this report are inconsistent with what are used in the MODFLOW documentation. The MODFLOW documentation uses i, j, k (row, column, layer) indexing; for efficiency, the MODFLOW computer code uses j, i, k (column, row, layer) indexing. The GMG code uses i, j, k (column, row, layer) indexing. The meaning of i and j is interchanged with i indexing columns instead of rows and j indexing rows instead of columns. Also, the indices in MODFLOW all start at 1 while the indices in GMG start at 0.

Finite-Difference Matrix

When applying the cell-centered finite-difference scheme on a regular grid, conservation of mass implies that, for each cell node J, the following equation is applicable:

$$\begin{aligned}
&-\mathtt{CV}(J-lm) * p(J-lm) - \mathtt{CC}(J-l) * p(J-l) - \mathtt{CR}(J-1) * p(J-1) \\
&+\mathtt{DD}(J) * p(J) \\
&-\mathtt{CR}(J) * p(J+1) - \mathtt{CC}(J) * p(J+l) - \mathtt{CV}(J) * p(J+lm) = \mathtt{RHS}(J)
\end{aligned} \tag{2}$$

where

$$\begin{aligned}
&\mathtt{DD}(J) = \mathtt{CV}(J-lm) + \mathtt{CC}(J-l) + \mathtt{CR}(J-1) \\
&+\mathtt{CR}(J) + \mathtt{CC}(J) + \mathtt{CV}(J) - \mathtt{HCOF}(J)
\end{aligned}$$

and `CR`, `CC` and `CV` are MF2K internal arrays storing the hydraulic conductances between adjacent columns, rows, and layers, respectively. The internal array `HCOF` (Harbaugh and others, 2000, p. 11,22,33,35,37) contains the storage terms, the sum of coefficients of head from source and sink terms, and correction terms applied under dewatered conditions. Storage terms are reflections of parameters in the transitive flow equation, which is fully discussed in McDonald and Harbaugh (1988). The internal array `RHS` is the right-hand side contribution corresponding to the hydraulic head p at the center of cell J. A system of equations based in equation (2) defines an $N \times N$ CCFD matrix equation. This matrix equation is denoted as

$$Ap = f. \tag{3}$$

The GMG package allocates and assembles the DD array, but does not allocate memory for any of the other coefficients in the CCFD matrix. Under normal compilation of MF2K, the coefficients and the right-hand side are single precision. The solution vector p is stored as an internal array that is double precision. The GMG package automatically detects the precision of internal arrays. Thus, linkage to the GMG library is possible even if MF2K is compiled in forced double precision.

Specified Heads and Inactive Cells

The internal integer array IBOUND indicates which cells have a specified head or are inactive. If $\texttt{IBOUND}(J) = 0$, then an inactive cell or a cell that has gone dry is indicated. If $\texttt{IBOUND}(J) < 0$, then a specified head is indicated, and the corresponding value of the specified head is stored in p. In the case of an inactive or dry cells, the corresponding value in p is set to the value of the internal variable HNOFLO. Any cell J with $\texttt{IBOUND}(J) \leq 0$ can be treated as a specified head. It is necessary to apply the specified heads before solving the matrix problem for the unknown heads. Let C be the matrix defined by

$$[Cp](J) = \begin{cases} p(J) & \text{if } \texttt{IBOUND}(J) \leq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

The specified heads are moved to the right-hand side by subtracting ACp from both sides of equation (3), resulting in

$$A(I - C)p = f - ACp \tag{5}$$

where I is the identity matrix. Equation (5) is then multiplied by $(I - C)$ to form a symmetric system of equations given by

$$(I - C)A(I - C)p = (I - C)(f - ACp). \tag{6}$$

Finally, Cp is added to both sides to make the system of equations non-singular, resulting in

$$((I - C)A(I - C) + C)p = (I - C)(f - ACp) + Cp. \tag{7}$$

In summary, equations (5) and (6) zero out the columns and rows, respectively, of A corresponding to specified heads, while equation (7) places a value of one on the corresponding diagonal elements.

If p_0 is an initial guess satisfying $Cp - Cp_0 = 0$, then the residual corresponding to equation (7) is given by

$$r_0 = (I - C)(f - Ap_0). \tag{8}$$

Elements of r_0 corresponding to specified heads have value zero; otherwise the values are the same as the residual for the original problem equation (3). Thus, an equivalent expression for equation (7) is

$$((I - C)A(I - C) + C)e = r_0 \tag{9}$$

where $p = p_0 + e$. Only the diagonal DD of the matrix $(I - C)A(I - C) + C$ and the residual r_0 are explicitly assembled. This assembly is given by the following algorithm:

Algorithm 1 **CCFD_assemble**(r_0, p_0, A):

1. For $k = 0, \ldots, n-1$
   For $j = 0, \ldots, m-1$
   For $i = 0, \ldots, l-1$

   (a) $J = i + jl + klm$

   (b) If $\texttt{IBOUND}(J) \leq 0$, then
   $r_0(J) = 0,\ \texttt{DD}(J) = 1$

   (c) Else

   i. $a = 0,\ b = \texttt{RHS}(J)$

   ii. If $k > 0$, then
   $\texttt{COND} = \texttt{CV}(J - lm)$,
   $a = a + \texttt{COND}$,
   $b = b - \texttt{COND} * p_0(J - lm)$

   iii. If $j > 0$, then
   $\texttt{COND} = \texttt{CC}(J - l)$,
   $a = a + \texttt{COND}$,
   $b = b - \texttt{COND} * p_0(J - l)$

   iv. If $i > 0$, then
   $\texttt{COND} = \texttt{CR}(J - 1)$,
   $a = a + \texttt{COND}$,
   $b = b - \texttt{COND} * p_0(J - 1)$

   v. If $i < l - 1$, then
   $\texttt{COND} = \texttt{CR}(J)$,
   $a = a + \texttt{COND}$,
   $b = b - \texttt{COND} * p_0(J + 1)$

   vi. If $j < m - 1$, then
   $\texttt{COND} = \texttt{CC}(J)$,
   $a = a + \texttt{COND}$,
   $b = b - \texttt{COND} * p_0(J + l)$

   vii. If $k < n - 1$, then
   $\texttt{COND} = \texttt{CV}(J)$,
   $a = a + \texttt{COND}$,
   $b = b - \texttt{COND} * p_0(J + lm)$

   viii. If $a = 0$, then
   $\texttt{IBOUND}(J) = 0$,
   $p_0(J) = \texttt{HNOFLO}$,
   $\texttt{DD}(J) = 1$,
   $r_0(J) = 0$

ix. Else
$\texttt{DD}(J) = a - \texttt{HCOF}(J)$,
$r_0(J) = b - \texttt{DD}(J) * p_0(J)$.

Note that step 1(c)viii sets a cell to inactive if all the surrounding conductances are zero, indicating no flow in or out of the cell. The evaluation of the matrix-vector product in equation (9) is computed using the following procedure:

Algorithm 2 **CCFD_eval**(p_1, p_2, A):

1. For $k = 0, \ldots, n-1$
   For $j = 0, \ldots, m-1$
   For $i = 0, \ldots, l-1$

   (a) $J = i + jl + klm$

   (b) $a = 0$

   (c) If $\texttt{IBOUND}(J) > 0$, then

      i. If $k > 0$ and $\texttt{IBOUND}(J - lm) > 0$, then
      $a = a - \texttt{CV}(J - lm) * p_2(J - lm)$

      ii. If $j > 0$ and $\texttt{IBOUND}(J - l) > 0$, then
      $a = a - \texttt{CC}(J - l) * p_2(J - l)$

      iii. If $i > 0$ and $\texttt{IBOUND}(J - 1) > 0$, then
      $a = a - \texttt{CR}(J - 1) * p_2(J - 1)$

      iv. If $i < l - 1$ and $\texttt{IBOUND}(J + 1) > 0$, then
      $a = a - \texttt{CR}(J) * p_2(J + 1)$

      v. If $j < m - 1$ and $\texttt{IBOUND}(J + l) > 0$, then
      $a = a - \texttt{CC}(J) * p_2(J + l)$

      vi. If $k < n - 1$ and $\texttt{IBOUND}(J + lm) > 0$, then
      $a = a - \texttt{CV}(J) * p_2(J + lm)$

   (d) $p_1(J) = \texttt{DD}(J) * p_2(J) - a$.

Linear Solution

Given an approximate solution $\hat{e}$ of equation (9), the approximate hydraulic head $\hat{p}$ is returned as

$$\hat{p} = p_0 + \hat{e}. \tag{10}$$

An approximate solution to equation (9) is obtained using the PCG algorithm. The PCG algorithm approximates the solution to a linear equation $Ax = b$, applying a preconditioner B such that $B^{-1}Ax = B^{-1}b$ is, in some sense, better conditioned and easier to solve than the original problem (Golub and Van Loan, 1989;

Xu, 1992). The A matrix and the preconditioner B need not be explicitly assembled. A generic PCG operator is defined to contain the actions of these operators. Let PCG.A_eval$(x_1, x_2, \text{PCG.A})$ define the operation $x_1 = Ax_2$, and let PCG.B_eval$(x_1, x_2, \text{PCG.B})$ define the operation $x_1 = B^{-1}x_2$. Furthermore, let $(x_1, x_2) = x_1^T x_2$ define the Euclidean inner product with associated l2-norm $\|x\| = \sqrt{(x, x)}$. Given a convergence criterion `BIGR` $\leq$ `DRCLOSE`, where `BIGR` is the l2-norm of the residual, and a maximum number of iterations `IITER`, the generic PCG algorithm is given by

Algorithm 3 **PCG_eval**$(x, b, \mathbf{PCG})$:

1. $x_0 = 0$, $r_0 = b$
2. `BIGR`$= \|r_0\|$
3. If `BIGR`> 0, then
   (a) PCG.B_eval$(p_0, r_0, \text{PCG.B})$
   (b) $\beta_{0,N} = \beta_{0,D} = (r_0, p_0)$
   (c) PCG.A_eval$(z_0, p_0, \text{PCG.A})$
   (d) $\alpha_0 = \beta_{0,D}/(z_0, p_0)$
   (e) $x_1 = x_0 + \alpha_0 p_0$
   (f) $r_1 = r_0 - \alpha_0 z_0$
   (g) $k = 1$
   (h) `BIGR`$= \|r_1\|$
   (i) While `BIGR`>`DRCLOSE` and $k <$`IITER`, do
      i. PCG.B_eval$(z_k, r_k, \text{PCG.B})$
      ii. $\beta_{k,N} = (r_k, z_k)$
      iii. $\beta_k = \beta_{k,N}/\beta_{k,D}$
      iv. $\beta_{k,D} = \beta_{k,N}$
      v. $p_k = z_k + \beta_k p_{k-1}$
      vi. PCG.A_eval$(z_k, p_k, \text{PCG.A})$
      vii. $\alpha_k = \beta_{k,N}/(p_k, z_k)$
      viii. $x_{k+1} = x_k + \alpha_k p_k$
      ix. $r_{k+1} = r_k - \alpha_k z_k$
      x. $k = k + 1$
      xi. `BIGR`$= \|r_{k+1}\|$
4. $x = x_k$

Algorithm 3 corresponds closely to the PCG algorithm described in Golub and Van Loan (1989). An initial guess of zero is not required in the PCG algorithm. However, if a zero initial guess is used, then the right-hand side can be overwritten by the residual, eliminating the need for one work vector.

Nonlinear Solution

In cases where conductances or sink/source terms are head dependent, the flow equation is nonlinear and the ground-water flow equation (1) is written as follows:

$$-\nabla \cdot \mathbf{K}(p)\nabla p = f(p). \tag{11}$$

Equation (11) is linearized using the following simple substitution method:

$$-\nabla \cdot \mathbf{K}(p_{i-1})\nabla p_i = f(p_{i-1}).$$

The indices i and $i-1$ imply an iterative scheme whereby $\mathbf{K}$ and f are updated using hydraulic heads from a previous solution, p_{i-1}, before obtaining the current solution p_i. The equivalent matrix equation is given by

$$A(p_{i-1})p_i = f(p_{i-1})$$

and the resulting Picard iteration is given by

$$p_i = \hat{A}^{-1}(p_{i-1})f(p_{i-1}), \tag{12}$$

where $\hat{A}^{-1}$ is an approximate inverse obtained form the PCG algorithm. Each Picard iteration results in a new CCFD problem requiring an approximate linear solution. Each Picard iteration is referred to as a nonlinear iteration, or outer iteration, and solutions of the CCFD problem in equation (12) are referred to as as a linear iteration, or inner iteration. The inner iteration uses initial guess p_{i-1}, with an initial residual given by

$$r_{i-1} = f_{i-1} - A_{i-1}p_{i-1} \tag{13}$$

where $A_{i-1} = A(p_{i-1})$ and $f_{i-1} = f(p_{i-1})$. The nonlinear solution is advanced by

$$p_i = p_{i-1} + \hat{e}_{i-1} \tag{14}$$

where the head change $\hat{e}_{i-1}$ is obtained from the PCG algorithm applied to the linear equation

$$A_{i-1}e_{i-1} = r_{i-1}. \tag{15}$$

A problem is strongly nonlinear if the iterates p_i change dramatically from one outer iteration to the next. Convergence of a strongly nonlinear problem may be accelerated by a damping in the head change:

$$p_{\delta_i} = p_{i-1} + \delta_i \hat{e}_{i-1} \tag{16}$$

where $0 < \delta_i \leq 1$. The damping parameter δ_i may be fixed for all iterations, or it may vary adaptively from one outer iteration to the next. The GMG solver implements either a fixed damping or Cooley's method (Mehl and Hill, 2001) of adaptive damping based on a head-change criterion.

Convergence Criteria

The convergence criterion of the Picard iteration is based on the head change measured by the max-norm:

$$\texttt{BIGH} = \max_J |\hat{e}_{i-1}(J)|. \tag{17}$$

The convergence criterion used in the PCG algorithm for the linear iteration is based on the l2-norm of the residual:

$$\texttt{BIGR} = \|r_i\|, \tag{18}$$

where

$$r_i = f_{i-1} - A_{i-1}p_i.$$

The convergence criterion can be adaptively relaxed to avoid unnecessary PCG iterations in the early stages of the nonlinear approximation. The strategy for an adaptive convergence criterion is based on the residual of the damped solution. The residual r_{δ_i} of the damped solution is given by

$$r_{\delta_i} = f_{i-1} - A_{i-1}p_{\delta_i} = (1-\delta_i)r_{i-1} + \delta_i r_i \tag{19}$$

Based on equation (19), an adaptive convergence criterion `DRCLOSE` for the PCG algorithm is defined as

$$\texttt{DRCLOSE} = (1-\delta_{i-1}) * \|r_{i-1}\| + \delta_{i-1} * \texttt{RCLOSE} \tag{20}$$

where `RCLOSE` is a specified bound on `BIGR` and δ_{i-1} is used as an approximation to δ_i.

Remark 2. Users of the GMG solver need to be aware that the convergence criteria of this package are not necessarily analogous to criteria of other MF2K solver packages.

Remark 3. If the nonlinear problem cannot be accelerated by damping, then an adaptive PCG convergence criterion can be simulated by limiting the maximum number of PCG iterations.

Remark 4. The residual r_{i-1} given in equation (13) also can be used as a stopping criterion for the Picard iteration. The LMG package (Mehl and Hill, 2001) uses a scaled l2-norm of r_{i-1} for the convergence criterion.

Multigrid

The cell-centered multigrid algorithm described in this report is based in the cell-centered multigrid method presented in Ewing and Shen (1993); Bramble and others (1996). In turn, these results follow from a non-variational multigrid analysis developed in Bramble and others (1991). Only a qualitative presentation of the cell-centered multigrid method is given here, followed with some numerical examples.

Smoothers

The multigrid method begins with a smoothing procedure (also known as relaxation) in the form of a stationary iteration (Kelley, 1995) with initial guess p_0. The smoother is denoted as S_{μ_0} where μ_0 is the number of iterations. The action of S_{μ_0} is given as follows:

Algorithm 4 $\quad p = S_{\mu_0}(p_0, f)$:

1. $r_0 = f - Ap_0$
2. For $k = 0, \ldots, \mu_0 - 1$
   (a) $p_{k+1} = p_k + B^{-1} r_k$
   (b) $r_{k+1} = f - Ap_{k+1}$
3. $p = p_{\mu_0}$

where B is a preconditioner. Note that, while Algorithm 4 does contain a preconditioner, it is a stationary method and not a conjugate gradient method. The error e_k after k iterations of Algorithm 4 is given by

$$e_k = (I - B^{-1}A)e_{k-1}. \tag{21}$$

The spectral radius of the iteration matrix $M = (I - B^{-1}A)$ is defined as

$$\rho(M) = \{\max |\lambda| : \lambda \text{ eigenvalue of } M\}.$$

If $\rho(M) \leq \alpha < 1$, then there exists $\alpha < 1$ such that $\|e_k\| \leq \alpha_k \|e_0\|$; α is defined as the convergence factor.

Algorithm 4 is a generic algorithm and different types of preconditioners result in different types of iterations. A family of standard iterations, as defined in Golub and Van Loan (1989) and Atkinson (1988), result from the following preconditioners:

- Jacobi Iteration: $B = D$ where $D = \text{diag}(A)$.
- Gauss-Seidel Iteration: $B = L+D$, or $B = D+U$ where L is the lower triangular part of A and U is the upper triangular part.
- Symmetric Gauss-Seidel (SGS): $B = (L + D)D^{-1}(D + U)$.
- Symmetric Successive Over-Relaxation (SSOR): For some real-valued $0 \leq \omega < 1$

$$B = \frac{1}{1 - \omega^2}((1 - \omega)L + D)D^{-1}(D + (1 - \omega)U).$$

Standard iterations have different convergence factors for different types of problems. For example, the Jacobi Iteration is convergent for diagonally dominant problems while Gauss-Seidel is convergent for symmetric positive definite problems. For $\omega = 0$, the SSOR iteration is equivalent to SGS. The SSOR iteration can be significantly faster than SGS, but it is difficult to derive an optimal ω.

Remark 5. It can be shown (Xu, 1992) that if A is a symmetric positive definite matrix and $\alpha < 1$, then the PCG method with preconditioner B will have a convergence factor strictly less than α. Thus, PCG serves as an acceleration of the stationary iteration in Algorithm 4.

ILU Smoother

Another type of preconditioning is incomplete factorization (Benzi, 2002; Barrett and others, 1994). Incomplete factorization is an approximation to the exact factorization $A = \hat{L}\hat{D}\hat{U}$, where $\hat{L}$ is lower triangular, $\hat{D}$ is diagonal, and $\hat{U}$ is upper triangular. Given a complete factorization, the system of equations is solved by a backward substitution, followed by diagonal scaling, followed by a forward substitution. The full factorization generates fill-in, resulting in a dense matrix. The strategy behind incomplete factorization is to limit the amount of fill-in while approximating the complete factorization. A zero fill-in strategy results in an incomplete factorization with the same number of non-zeros as the original matrix. The zero fill-in strategy, when applied to the CCFD matrix, requires only assembling the diagonal $\hat{D}$; the upper and lower triangular parts are assembled as needed in the evaluation process. This type of incomplete factorization is referred to as ILU0-D and is expressed in matrix form as

$$B = (L + \hat{D})\hat{D}^{-1}(\hat{D} + U).$$

Henceforth, the ILU0-D smoother will be referred to as simply ILU. The diagonal $\hat{D}$ is assembled by the following procedure:

Algorithm 5 **CCFD_ILU_assemble**($\hat{D}, A$):

1. For $i = 0, \ldots, l-1$,
   For $j = 0, \ldots, m-1$,
   For $k = 0, \ldots, n-1$
   (a) $J = i + jl + klm$
   (b) $a = 0$
   (c) If $\texttt{IBOUND}(J) > 0$, then
      i. If $i > 0$ and $\texttt{IBOUND}(J-1) > 0$, then
         $a = a + \texttt{CR}(J-1) * \texttt{CR}(J-1)/\hat{D}(J-1)$
      ii. If $j > 0$ and $\texttt{IBOUND}(J-l) > 0$, then
         $a = a + \texttt{CC}(J-l) * \texttt{CC}(J-l)/\hat{D}(J-l)$

iii. If $k > 0$ and $\texttt{IBOUND}(J - lm) > 0$, then
$a = a + \texttt{CV}(J - lm) * \texttt{CV}(J - lm)/\hat{D}(J - lm)$

(d) $\hat{D}(J) = \texttt{DD}(J) - a$.

The evaluation of the ILU smoother is given by

Algorithm 6 **CCFD_ILU_eval**$(u, p, \mathbf{ILU})$:

1. Backward substitution
For $i = 0, \ldots, l - 1$
For $j = 0, \ldots, m - 1$
For $k = 0, \ldots, n - 1$

   (a) $J = i + jl + klm$

   (b) $a = 0$

   (c) If $\texttt{IBOUND}(J) > 0$, then

      i. If $k > 0$ and $\texttt{IBOUND}(J - lm) > 0$, then
      $a = a + \texttt{CV}(J - lm) * u(J - lm)$

      ii. If $j > 0$ and $\texttt{IBOUND}(J - l) > 0$, then
      $a = a + \texttt{CC}(J - l) * u(J - l)$

      iii. If $i > 0$ and $\texttt{IBOUND}(J - 1) > 0$, then
      $a = a + \texttt{CR}(J - 1) * u(J - 1)$

   (d) $u(J) = (p(J) + a)/\hat{D}(J)$

2. Forward Substitution
For $i = l - 1, \ldots, 0$
For $j = m - 1, \ldots, 0$
For $k = n - 1, \ldots, 0$

   (a) $J = i + jl + klm$

   (b) $a = 0$

   (c) If $\texttt{IBOUND}(J) > 0$, then

      i. If $k < n - 1$ and $\texttt{IBOUND}(J + lm) > 0$, then
      $a = a + \texttt{CV}(J) * u(J + lm)$

      ii. If $j < m - 1$ and $\texttt{IBOUND}(J + l) > 0$, then
      $a = a + \texttt{CC}(J) * u(J + l)$

      iii. If $i < l - 1$ and $\texttt{IBOUND}(J + 1) > 0$, then
      $a = a + \texttt{CR}(J) * u(J + 1)$

   (d) $u(J) = u(J) + a/\hat{D}(J)$.

Remark 6. The PCG2 package in MF2K (Hill, 1990) allows the use of a modified ILU preconditioner. The modified ILU preconditioner adds terms to the diagonal $\hat{D}$ of the factorization that would otherwise have presence in the off-diagonals of the full factorization. In the PCG2 package, the factors are weighted by a RELAX parameter between 0 and 1. If the value of RELAX is 1, then the row-sums of the incomplete factorization are equal to the row-sums of the CCFD matrix. This results in a better spectral condition number for the PCG method (Dupont and others, 1968; Gustaffson, 1978). However, the modified ILU preconditioner can be sensitive to rounding error and can break down during factorization (Van der Vorst, 1990). Our experience is that modified ILU does not function well for stationary iterations such as Algorithm 4.

Remark 7. The right-hand side in Algorithm 6 can be overwritten by the solution, eliminating the need to store an additional work vector for the ILU smoothing operation.

Remark 8. The evaluation of the symmetric Gauss-Seidel smoother is identical to the evaluation of the ILU smoother with `DD` *replacing $\hat{D}$. The symmetric Gauss-Seidel smoother eliminates the need to store the diagonal of the factorization, but it is not as robust a smoother as ILU. However, problems where savings in computer memory is beneficial may warrant use of this smoother.*

Coarse-Grid Correction

In this section the fine-grid space is denoted as Ω^h and the coarse-grid space as Ω^H, where h and H represent the average cell diameters of the fine grid and coarse grid respectively. Although not necessary in general, the number of coarse-grid cells is assumed to be given by $N_H = (1/H)^d$, where $d = 1, 2, 3$ is the spatial dimension. Under standard coarsening, $H = 2h$ so that the number of fine-grid cells is $N_h = 2^d N_H$. The solution p is in the fine-grid space and is denoted as $p^h \in \Omega^h$. Henceforth, vectors and matrices (linear operators) in the fine-grid and coarse-grid spaces will be denoted by superscript h and superscript H respectively.

Algorithm 4 tends to be good for smoothing the error e_k^h. To illustrate the effect of smoothing, a CCFD matrix A^h is assembled from a two-dimensional boundary-value problem discretized on a 32×32 grid. A random hydraulic-conductivity field, which has a variance of up to five orders of magnitude, is defined on a 16×16 coarse grid. A random right-hand side is assembled by letting $f^h = A^h u^h$, where u^h is a random vector. The initial guess p_0^h is chosen to be zero so that the initial error is u^h. The result of the smoothing procedure is illustrated in figure 1. After three iterations, only smooth error generally remains and further smoothing becomes less effective at reducing the error.

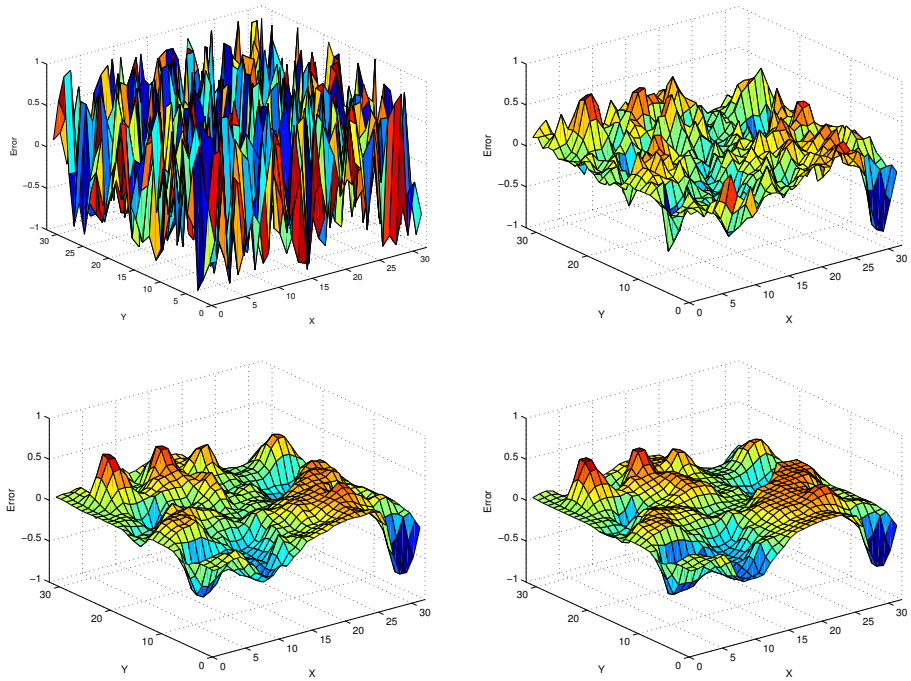


Figure 1: Going from top left clockwise, the initial error, the error after one smoothing, after two smoothings, and after three smoothings using Algorithm 4.

The idea behind the coarse-grid correction is that the smooth error can be accurately approximated on a coarser grid and used to correct the fine-grid solution. Coarse-grid nodal values are transfered to the fine-grid through a prolongation operator P using the natural embedding of Ω^H in to Ω^h. If $p^H \in \Omega^H$ is a vector of nodal values, then P is a $N_h \times N_H$ matrix. Let $N_H = l_0 \times m_0 \times n_0$ and $N_h = 2l_0 \times 2m_0 \times 2n_0$. The three-dimensional prolongation is given by

Algorithm 7 **CCFD_P_eval**$(p^h, p^H, l_0, m_0, n_0)$:

1. $l = 2l_0$, $m = 2m_0$, $n = 2n_0$.
2. For $k_0 = 0, \ldots, n_0 - 1$
   For $j_0 = 0, \ldots, m_0 - 1$
   For $i_0 = 0, \ldots, l_0 - 1$
   (a) $i = 2i_0$, $j = 2j_0$, $k = 2k_0$
   (b) $J_H = i_0 + j_0 l_0 + k_0 l_0 m_0$
   (c) $J_h = i + jl + klm$

$$
\text{(d)} \quad \begin{array}{l}
p^h(J_h) = p^H(J_H) \\
p^h(J_h+1) = p^H(J_H) \\
p^h(J_h+l) = p^H(J_H) \\
p^h(J_h+1+l) = p^H(J_H) \\
p^h(J_h+lm) = p^H(J_H) \\
p^h(J_h+1+lm) = p^H(J_H) \\
p^h(J_h+l+lm) = p^H(J_H) \\
p^h(J_h+1+l+lm) = p^H(J_H).
\end{array}
$$

For $e_k^h \in \Omega^h$, the coarse-grid error is defined to be the unique function $e_k^H \in \Omega^H$ such that

$$(A^H e_k^H, w) = (A^h e_k^h, Pw), \quad \text{for all } w \in \Omega^H. \tag{22}$$

where $(\cdot,\cdot)$ is the usual algebraic inner product. Equation (22) implies that

$$e_k^H = A^{H^{-1}} P^T A^h e_k^h = A^{H^{-1}} P^T r_k^h.$$

The $N_H \times N_h$ restriction operator $R = P^T$ is given algorithmically as

Algorithm 8 **CCFD_R_eval**$(p^H, p^h, l_0, m_0, n_0)$:

1. $l = 2l_0$, $m = 2m_0$, $n = 2n_0$
2. For $k_0 = 0, \ldots, n_0 - 1$
   For $j_0 = 0, \ldots, m_0 - 1$
   For $i_0 = 0, \ldots, l_0 - 1$
   (a) $i = 2i_0$, $j = 2j_0$, $k = 2k_0$
   (b) $J_H = i_0 + j_0 l_0 + k_0 l_0 m_0$
   (c) $J_h = i + jl + klm$
   (d)
$$
\begin{array}{rcl}
p^H(J_H) & = & p^h(J_h) \\
 & + & p^h(J_h+1) \\
 & + & p^h(J_h+l) \\
 & + & p^h(J_h+1+l) \\
 & + & p^h(J_h+lm) \\
 & + & p^h(J_h+1+lm) \\
 & + & p^h(J_h+l+lm) \\
 & + & p^h(J_h+1+l+lm),
\end{array}
$$

If it is assumed the hydraulic conductances are constant within each coarse-grid cell and that the conductances across cell faces result from harmonic averaging (Harbaugh and others, 2000, p. 22-27), then it can be shown that

$$(A^h Pv, Pw) = 2(A^H v, w), \quad \text{for all } v, w \in \Omega^H. \tag{23}$$

From equation (23), the $N_H \times N_H$ coarse-grid CCFD matrix A^H is defined as

$$A^H = \frac{1}{2} R A^h P = \frac{1}{2} P^T A^h P. \tag{24}$$

Special attention must be given when applying cells with specified heads to equation (24). Accounting for these special cells in the coarsening makes the GMG solver effective for problems with complex geometries modeled in MF2K by the use of inactive cells. A temporary fine-grid array $\mathtt{EE}^h$ is first assembled to eliminate contributions of conductances corresponding to cells with specified heads to the diagonal. This array is assembled as

Algorithm 9 **EE_assemble**($\mathbf{EE}^h, A^h$):

1. For $k = 0, \ldots, n-1$
   For $j = 0, \ldots, m-1$
   For $i = 0, \ldots, l-1$
   (a) $J_h = i + jl + klm$
   (b) If $\mathtt{IBOUND}^h(J_h) \leq 0$, then
      i. $\mathtt{EE}^h(J_h) = 0$
   (c) Else,
      i. $a = 0$
      ii. If $k > 0$ and $\mathtt{IBOUND}^h(J_h - lm) > 0$, then
         $a = a + \mathtt{CV}^h(J_h - lm)$
      iii. If $j > 0$ and $\mathtt{IBOUND}^h(J_h - l) > 0$, then
         $a = a + \mathtt{CC}^h(J_h - l)$
      iv. If $i > 0$ and $\mathtt{IBOUND}^h(J_h - 1) > 0$, then
         $a = a + \mathtt{CR}^h(J_h - 1)$
      v. If $i < l - 1$ and $\mathtt{IBOUND}^h(J_h + 1) > 0$, then
         $a = a + \mathtt{CR}^h(J_h)$
      vi. If $j < m - 1$ and $\mathtt{IBOUND}^h(J_h + l) > 0$, then
         $a = a + \mathtt{CC}^h(J_h)$
      vii. If $k < n - 1$ and $\mathtt{IBOUND}^h(J_h + lm) > 0$, then
         $a = a + \mathtt{CV}^h(J_h)$
      viii. $\mathtt{EE}^h(J_h) = \mathtt{DD}^h(J_h) - a$.

Using $\mathtt{EE}^h$, the coarse-grid matrix A^H can be assembled as

Algorithm 10 **CCFD_RAP_assemble**(A^H, A^h, l_0, m_0, n_0):

1. $l = 2l_0$, $m = 2m_0$, $n = 2n_0$

2. For $k_0 = 0, \ldots, n_0 - 1$
   For $j_0 = 0, \ldots, m_0 - 1$
   For $i_0 = 0, \ldots, l_0 - 1$

(a) $i = 2i_0$, $j = 2j_0$, $k = 2k_0$

(b) $J_H = i_0 + j_0 l_0 + k_0 l_0 m_0$

(c) $J_h = i + jl + klm$

(d) Excluding conductances for cells corresponding to specified heads

$$\begin{aligned}
\mathtt{CR}^H(J_H) &= 1/2(\mathtt{CR}^h(J_h+1) + \mathtt{CR}^h(J_h+1+l)) \\
&+ 1/2(\mathtt{CR}^h(J_h+1+lm) + \mathtt{CR}^h(J_h+1+l+lm)) \\
\mathtt{CC}^H(J_H) &= 1/2(\mathtt{CC}^h(J_h+l) + \mathtt{CC}^h(J_h+1+l)) \\
&+ 1/2(\mathtt{CC}^h(J_h+l+lm) + \mathtt{CC}^h(J_h+1+l+lm)) \\
\mathtt{CV}^H(J_H) &= 1/2(\mathtt{CV}^h(J_h+lm) + \mathtt{CV}^h(J_h+1+lm)) \\
&+ 1/2(\mathtt{CV}^h(J_h+l+lm) + \mathtt{CV}^h(J_h+1+l+lm)) \\
\mathtt{DD}^H(J_H) &= 1/2(\mathtt{EE}^h(J_h) + \mathtt{EE}^h(J_h+1)) \\
&+ 1/2(\mathtt{EE}^h(J_h+l) + \mathtt{EE}^h(J_h+1+l)) \\
&+ 1/2(\mathtt{EE}^h(J_h+lm) + \mathtt{EE}^h(J_h+1+lm)) \\
&+ 1/2(\mathtt{EE}^h(J_h+l+lm) + \mathtt{EE}^h(J_h+1+l+lm)) \\
&+ \mathtt{CR}^H(J_H-1) + \mathtt{CR}^H(J_H) \\
&+ \mathtt{CC}^H(J_H-l_0) + \mathtt{CC}^H(J_H) \\
&+ \mathtt{CV}^H(J_H-l_0m_0) + \mathtt{CV}^H(J_H)
\end{aligned}$$

(e) If $\mathtt{DD}^H(J_H) = 0$, then

i. $\mathtt{DD}^H(J_H) = 1$

ii. $\mathtt{IBOUND}^H(J_H) = 0$

(f) Else,

i. $\mathtt{IBOUND}^H(J_H) = 1$.

The coarse-grid correction algorithm is a stationary iteration preconditioned by μ_0 iterations of a smoother $S^h_{\mu_0}$, followed by a coarse-grid correction $P(A^H)^{-1}R$, followed by another μ_0 iterations of the smoother $S^h_{\mu_0}$. Iterating μ_1 times, the coarse-grid correction algorithm is given by

Algorithm 11 **CCFD_CG_eval**(p^h, p^h_0, f^h):

1. For $k = 0, \ldots, \mu_1 - 1$

   (a) $p^h_{k+1/3} = S^h_{\mu_0}(p^h_k, f_h)$

   (b) $r^h_{i+1/3} = f^h - A^h p^h_{k+1/3}$

   (c) Solve
   $A^H e^H_{i+1/3} = R r^h_{i+1/3}$

   (d) $p^h_{k+2/3} = p^h_{k+1/3} + P e^H_{i+1/3}$

   (e) $p^h_{k+1} = S^h_{\mu_0}(p^h_{k+2/3}, f^h)$

2. $p^h = p^h_{\mu_1}$

The error $e^h_{k+1/3} = p^h - p^h_{k+1/3}$ after smoothing in step 1a is the same as the error shown in figure 1. The error $e^h_{k+2/3} = p^h - p^h_{k+2/3}$, after a coarse-grid correction in step 1d and the error e^k_h, after post-smoothing in step 1e, are illustrated in figure 2. The total error $e^h_{k+1} = p^h - p^h_{k+1}$ is given by

$$e^h_{k+1} = (I^h - (B^h)^{-1}A^h)^{\mu_0}(I^h - PA^{H^{-1}}RA^h)(I^h - (B^h)^{-1}A^h)^{\mu_0}e^h_k.$$

The coarse-grid solution in step 1b of algorithm 11 is a cell-centered finite difference approximation of the error $e^h_{k+1/3} = p^h - p^h_{k+1/3}$ on the coarse-grid space Ω^H.

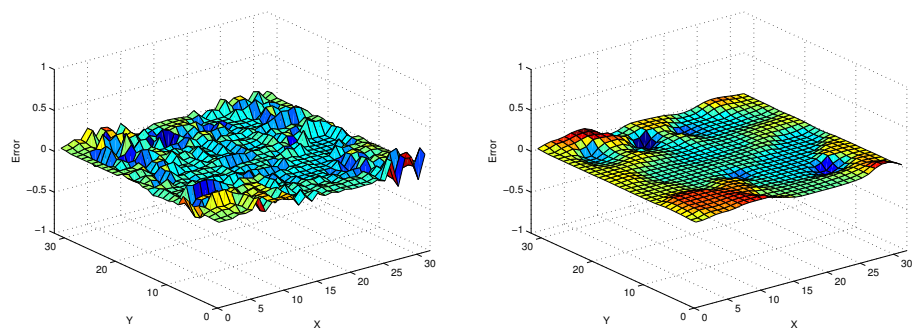


Figure 2: The error $e^h_{k+2/3}$ after after a coarse-grid correction (step 1d of Algorithm 11) is shown on the left. The error e^h_{k+1} after subsequent smoothing (step 1e of Algorithm 11) is shown on the right.

Remark 9. Ewing and Shen (1993) indicate that smooth variations of the hydraulic conductivity coefficient within a coarse-grid cell does not seriously effect the convergence of the coarse-grid correction scheme. In cases where there are large jumps in the coefficient, a semi-coarsening approach can be used. For example, there may be large differences in the vertical conductivity from one layer to the next. In such cases, it is best that coarsening be limited to only the horizontal directions.

Remark 10. Bramble and others (1996) note that the cell-centered multigrid method violates many of the standard heuristics for multigrid algorithms. For example, it is believed that the sum of the orders of the prolongation/restriction pair should be greater than the order of the PDE being approximated (Hackbush, 1985; Wesseling, 1991). The order of the flow equation is two, while the sum of the orders of the prolongation/restriction pair in the cell-centered multigrid method also is two.

Remark 11. The three-dimensional prolongation/restriction operator is implemented in the computer code as a series of two-dimensional prolongations/restrictions that are, in turn, implemented as a series of one-dimensional prolongations/restrictions.

Special consideration for specified heads in the prolongation/restriction is not necessary. Simple modifications to the prolongation/restriction algorithms allow for grids with odd numbers of cells.

The computer code to assemble the coarse-grid matrix implements a three-dimensional coarsening based on a series of two-dimensional coarsenings that in turn are based on a series of one-dimensional coarsenings. Simple modifications are made to allow for grids with odd numbers of cells.

ν-Cycle

If the coarse-grid problem is too large to solve directly, then another coarse-grid correction scheme is applied recursively until a sufficiently small problem is obtained. The GMG solver coarsens until a one-dimensional problem is obtained. At this point, the ILU smoother becomes an exact factorization of the coarse-grid problem. The computational complexity of multigrid is independent of the number of levels. The only advantage in limiting the number of levels is a possible improvement in the convergence factor. In this case, a sparse matrix solver or an iterative method would be needed to solve the larger coarse-grid problem. The GMG solver does not implement this approach; numerical results have not indicated any advantage in using a solver on a larger coarse-grid.

Recursive application of the coarse-grid correction scheme is referred to as the ν-Cycle algorithm. Assume there are L levels of the multigrid algorithm numbered by $s = L-1, \ldots, 0$ where level $s = 0$ is the coarsest level. Superscript s is used on vectors and operators to indicate the space they are in. The multigrid components are defined as follows:

1. Restriction Operator $R^s : \Omega^s \to \Omega^{s-1}$.
2. Prolongation Operator $P^s : \Omega^{s-1} \to \Omega^s$.
3. Smoother $S^s_{\mu_0} : \Omega^s \to \Omega^s$.
4. Coefficient Matrix $A^s : \Omega^s \to \Omega^s$.

The multilevel CCFD matrix is defined by

$$A^{L-1} = A^h \tag{25}$$

$$A^{s-1} = \frac{1}{2} R^s A^s P^s, \quad \text{for } s = L-1, \ldots 1 \tag{26}$$

The recursive ν-Cycle algorithm is given by

Algorithm 12 **ν-Cycle_eval**(p^s, f^s, A^s):

1. If $s = 0$, then
   (a) $p^s = (A^s)^{-1} f^s$

2. Else

   (a) $p^s = S^s_{\mu_0}(p^s, f^s)$

   (b) If $s = L - 1$, then $\nu = 1$

   (c) For $m = 0, \ldots, \nu - 1$

      i. $r^{s-1} = R^s(f^s - A^s p^s)$

      ii. $p^{s-1} = 0$

      iii. ν-Cycle_eval$(p^{s-1}, r^{s-1}, A^{s-1})$

      iv. $p^s = p^s + P^s p^{s-1}$

      v. $p^s = S^s_{\mu_0}(p^s, f^s)$

Algorithm 12 represents one cycle of the multigrid algorithm. The PCG preconditioner in the GMG package (Algorithm 3) is given by μ_1 cycles of Algorithm 12 as follows:

Algorithm 13 $p^h = \mathbf{MG_eval}(p^h_0, f^h, A^h)$:

1. For $k = 0, \ldots, \mu_1 - 1$

   (a) ν-Cycle_eval(p^h, f^h, A^h)

The parameter ν on line 2c of Algorithm 12 represents the number of iterations used for the approximate coarse-grid correction on each level. The value of ν is typically one or two and results in various grid schedules. The different grid schedules are illustrated in figure 3. For $\nu = 1$, a so-called V-Cycle results, while $\nu = 2$ produces a W-Cycle (Briggs, 1987; Trottenberg and others, 2001). For both the V-Cycle and W-Cycle, a value of $\nu = 1$ is used for level $s = L - 1$; otherwise, two cycles would result. The GMG package uses ILU smoothing with $\mu_0 = \mu_1 = \nu = 2$.

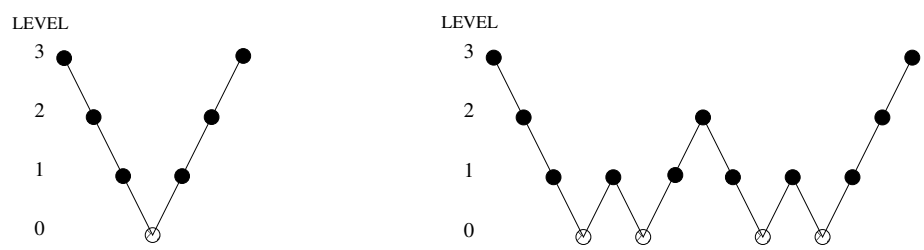


Figure 3: The grid schedules for a 4-level multigrid cycle. The grid schedule on the left is for $\nu = 1$ (V-Cycle) and the grid schedule for $\nu = 2$ (W-Cycle) is shown on the right.

Remark 12. Bramble and others (1996) show that the cell-centered multigrid algorithm converges with an iterative rate of convergence bounded independently of the

number of levels. That the simplest prolongation/restriction pairs are able to be used (see Remark 10) and accurately approximate the coarse-grid CCFD discretization using (24) (see Remark 9) is the reason why the geometric multigrid preconditioner in the GMG solver is remarkably efficient in terms of memory requirements and computer execution time.

Comparison of GMG with AMG

The AMG solver is linked to MF2K using the LMG package described in Mehl and Hill (2001). Comparisons are made between GMG and AMG for several large two-dimensional and three-dimensional ground-water flow problems. The LMG package uses a scaled l2-norm of the residual equation (13) as a stopping criterion. To make the comparisons equal, the l2-norm in LMG is unscaled and the same stopping criterion are used in GMG. With these modifications, the stopping criterion for both solvers is presumed to be about equal. Conjugate gradient acceleration is implemented in the AMG solver for all the test problems. The test problems were run on a Linux work station with a 1.4 gigahertz (GHz) processor and 1 gigabyte (GB) of random access memory (RAM).

The computational efficiencies of GMG and AMG are compared in Table 1 using seven different test problems. Test problems 1 through 5 are taken from Mehl and Hill (2001). Test problem 6 is an enlargement of the test problem given in (Harbaugh and others, 2000, p. 89). Test problem 7 is from Detwiler and others (2002).

Table 1: Comparison of computational differences between the GMG (cell-centered geometric multigrid preconditioner) and AMG (algebraic multigrid) solvers for different ground-water flow problems.

Test Problem	[1] Type	[2] (l, m, n)	Memory (megabytes)		CPU Time (seconds)		Iterations	
			GMG	AMG	GMG	AMG	GMG	AMG
1	SS, NL	$(120, 24, 60)$	195	925	219	371	48	71
2	SS, L	$(700, 1500, 1)$	132	367	33	37	7	12
3	TR, NL	$(194, 190, 4)$	21	64	671	875	1740	886
4	SS, L	$(160, 194, 15)$	66	208	29	31	13	21
5	SS, NL	$(153, 163, 3)$	12	33	6	18	25	89
6	SS, L	$(160, 160, 40)$	115 [3] 125	596	72 [3] 27	115	22 [3] 6	13
7	SS, L	$(855, 1952, 1)$	209	595	32	38	4	4

[1] (SS, steady-state; TR, transient; L, linear, NL, nonlinear)
[2] (columns, rows, layers)
[3] Coarsening along columns and rows only (SC=1).

Problem 1 is a large three-dimensional steady-state nonlinear problem. The variance of the hydraulic-conductivity field is approximately 4.0. The nonlinearity is

weak and requires only a few Picard iterations. The residual stopping criterion is set at `RCLOSE` = 10^{-5}. The AMG solver requires about five times as much memory as the GMG solver.

Problem 2 is a large two-dimensional steady-state linear problem with a complex hydraulic-conductivity field. The residual stopping criterion is set at `RCLOSE` = 10^{-10}. The GMG solver and the AMG solver perform equally well in terms of CPU time, with the AMG solver requiring about three times as much memory.

Problem 3 is a moderately sized nonlinear transient model with 49 stress periods resulting in a large number of total iterations. The problem is also nonlinear in each stress period, but the nonlinearities are weak and require only a few Picard iterations. This problem is a case where the head change becomes small, but the residual remains relatively large. The stopping criterion for the residual is set at `RCLOSE` = 10^{-2}. The GMG solver takes about twice as many iterations per stress period as the AMG solver. The CPU time per iteration for the GMG solver is about half that of the AMG time per iteration, making the total CPU times about equal. The memory used by the AMG solver is about three times the memory used by GMG.

Problem 4 is a large three-dimensional model with a complex heterogeneous hydraulic-conductivity field. The problem is steady-state and linear. The convergence criterion is set at `RCLOSE` = 10^{-5}. The CPU times are about equal for the GMG solver and AMG solver with the AMG solver, using about three times the memory.

Problem 5 is a moderately sized three-dimensional steady-state model with a hydraulic-conductivity field that is fairly complex and strong nonlinearities resulting from evaportranspiration. An adaptive damping strategy is used in both solvers for this problem. The convergence criterion is set at `RCLOSE` = 10^{-5}. The relative efficiency of the GMG solver is due to the adaptive convergence criterion for the inner iteration as described in the "Convergence Criteria" section.

Problem 6 is an enlargement of the steady-state example problem given in (Harbaugh and others, 2000, p. 89). The enlargement consists of increasing both the row and column count to 160, and the layer count to 40. Five different hydraulic conductivity zones were associated with the 40 layers; these zones were created using the Layer-Property Flow (LPF) package (Harbaugh and others, 2000, p. 59). To ensure that some fluid flow occurred in all zones, additional wells were situated at more levels in these layers. Constant heads were imposed in a few columns along one side of the discretized domain, as in the original example problem. For the linear simulation, the drain package was eliminated, and the unconfined layers were changed to confined.

The AMG solver requires about five times as much memory as the GMG solver for this test problem. If coarsening is restricted only to the columns and rows,

then the performance of the GMG solver improves dramatically. The residual convergence criterion is set at RCLOSE $= 10^{-5}$.

Problem 7 is a steady-state linear problem used in Detwiler and others (2002) to illustrate the ability of AMG to efficiently solve the system of equations resulting from complex, irregularly shaped domains as part of a two-phase two-dimensional flow and transport problem. The nonaqueous phase liquid (NAPL) is entrapped in fractures as illustrated in Figure 4. Cells of entrapped NAPL are modeled in MF2K by inactive cells. These inactive cells are accounted for, in the coarsening of the CCFD matrix, by Algorithms 9 and 10. The residual convergence criterion is set at RCLOSE $= 10^{-5}$.

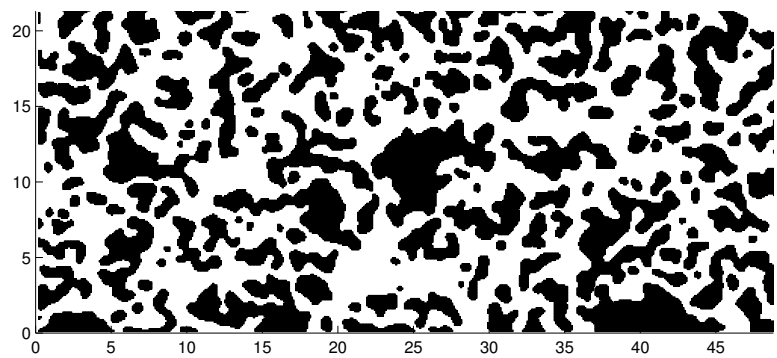


Figure 4: Section (22×50 cells) of 1952×855 cell grid containing fractures (dark areas) with entrapped nonaqueous phase liquid.

Input Instructions for the GMG Solver

The GMG solver package has both an inner loop and outer loop. The inner loop is set up for the convergence of the linear problem and is controlled by the parameters RCLOSE and IITER. A nonlinear problem is controlled through the outer loop and uses parameters HCLOSE and MXITER. As a solution of the linear problem is used as an approximate solve for the nonlinear problem, all four parameters are needed when solving a nonlinear problem. The operation of these parameters is further explained in the section titled "**Module GMG1AP**".

Input for the GMG package is read from the file that is type "GMG" in the name file. Free format is used for reading all values on the input list. The GMG data file should contain the following data items:

```
0. RCLOSE IITER HCLOSE MXITER
1. DAMP IADAMP IOUTGMG
2. ISM ISC
3. RELAX
```

An optional [#Text] item can be inserted multiple times before any of the above items; the symbol # must appear in the first column. These optional items are treated as comments. The convergence criteria for the GMG package may look similar to other packages, such as the PCG2 package described in Hill (1990), but their meaning may be quite different. The reader is encouraged to refer back to the "Convergence Criteria" section for more detail. The GMG data items represent the following quantities:

`RCLOSE` is the residual convergence criterion for the inner iteration. The PCG algorithm computes the l2-norm of the residual and compares it against `RCLOSE`. Typically, `RCLOSE` is set to the same value as `HCLOSE` (see below). If `RCLOSE` is set too high, then additional outer iterations may be required due to the linear equation not being solved with sufficient accuracy. On the other hand, a too restrictive setting for `RCLOSE` for nonlinear problems may force an unnecessarily accurate linear solution. This may be alleviated with the `IITER` parameter or with damping.

`IITER` is the maximum number of PCG iterations for each linear solution. A value of 100 is typically sufficient. It is frequently useful to specify a smaller number for nonlinear problems so as to prevent an excessive number of inner iterations.

`HCLOSE` is the head-change convergence criterion for nonlinear problems. After each linear solve (inner iteration), the max-norm of the head change is compared against `HCLOSE`. `HCLOSE` can be set to a large number for linear problems; `HCLOSE` is ignored if `MXITER`=1.

`MXITER` is the maximum number of outer-iterations. For linear problems, `MXITER` can be set to 1. For nonlinear problems, `MXITER` needs to be larger, but rarely more than 100.

`DAMP` is the value of the damping parameter. For linear problems, a value of 1.0 should be used. For nonlinear problems, a value less than 1.0 but greater than 0.0 may be necessary to achieve convergence. A typical value for nonlinear problems is 0.5. Damping also helps control the convergence criterion of the linear solve to alleviate excessive PCG iterations (see equation (20).

`IADAMP` is a flag that controls adaptive damping. The possible values of `IADAMP` and their meanings are as follows:

If `IADAMP` $= 0$, then the value assigned to `DAMP` is used as a constant damping parameter.

If `IADAMP` $\neq 0$, then the value of `DAMP` is used for the first nonlinear iteration (see "Nonlinear Solution" section). The damping parameter is adaptively varied on the basis of the head change, using Cooley's method as described in Mehl and Hill (2001), for subsequent iterations.

`IOUTGMG` is a flag that controls the output of the GMG solver. The possible values of `IOUTGMG` and their meanings are as follows:

If `IOUTGMG` = 0, then only the solver inputs are printed.

If `IOUTGMG` = 1, then for each linear solve, the number of PCG iterations, the value of the damping parameter, the l2-norm of the residual, and the max-norm of the head-change and its location (column, row, layer) are printed. At the end of a time/stress period, the total number of GMG calls, PCG iterations, and a running total of PCG iterations for all time/stress periods are printed.

If `IOUTGMG` = 2, then the convergence history of the PCG iteration is printed, showing the l2-norm of the residual and the convergence factor for each iteration.

`IOUTGMG` = 3 is the same as `IOUTGMG` = 1 except output is sent to the terminal instead of the MF2K LIST output file.

`IOUTGMG` = 4 is the same as `IOUTGMG` = 2 except output is sent to the terminal instead of the MF2K LIST output file.

`ISM` is a flag that controls the type of smoother used in the multigrid preconditioner. The possible values for `ISM` and their meanings are as follows:

If `ISM` = 0, then ILU(0) smoothing is implemented in the multigrid preconditioner. This smoothing requires an additional vector on each multigrid level to store the pivots in the ILU factorization.

If `ISM` = 1, then Symmetric Gauss-Seidel (SGS) smoothing is implemented in the multigrid preconditioner. No additional storage is required for this smoother; users may want to use this option if available memory is exceeded or nearly exceeded when using `ISM`=0. Using SGS smoothing is not as robust as ILU smoothing; additional iterations are likely to be required in reducing the residuals. In extreme cases, the solver may fail to converge as the residuals cannot be reduced sufficiently.

`ISC` is a flag that controls semi-coarsening in the multigrid preconditioner. The possible values of `ISC` and their meanings are given as follows:

If `ISC` = 0, then the rows, columns and layers are all coarsened.

If `ISC` = 1, then the rows and columns are coarsened, but the layers are not.

If `ISC` = 2, then the columns and layers are coarsened, but the rows are not.

If `ISC` = 3, then the rows and layers are coarsened, but the columns are not.

If `ISC` = 4, then there is no coarsening.

Typically, the value of `ISC` should be 0 or 1. In the case that there are large vertical variations in the hydraulic conductivities, , then a value of 1 should be used (see Remark 9 in "Coarse-Grid Correction" section). If no coarsening is implemented (`ISC` = 4), then the GMG solver is comparable to the PCG2 ILU(0) solver described in Hill (1990) and uses the least amount of memory.

RELAX is a relaxation parameter for the ILU preconditioned conjugate gradient method. The RELAX parameter can be used to improve the spectral condition number of the ILU preconditioned system. The value of RELAX should be approximately one. However, the relaxation parameter can cause the factorization to break down. If this happens, then the GMG solver will report an assembly error and a value smaller than one for RELAX should be tried. This item is read only if ISC = 4.

Adaptive Damping Example

A sample input file for the GMG package is given below for test problem 5:

```
#==============================
# Test 5
#==============================
# RCLOSE  IITER  HCLOSE  MXITER
#==============================
  1.0E-5  100    1.0E-5  100
#==============================
# DAMP  IADAMP  IOUTGMG
#==============================
  0.5   1       4
#==============================
# ISM  SC
#==============================
  0    0
```

The convergence criterion for the l2-norm of the residual in the inner iteration and the head-change criterion for the outer iteration are both 10^{-5}. The maximum number of outer-iterations is 100. The maximum number of PCG iterations for a given outer-iteration is 100. Adaptive damping is implemented with an initial damping value of 0.5. The output for the solver, including reduction histories, is written to standard out. ILU smoothing is implemented in the multigrid preconditioner and full coarsening is used for the columns, rows, and layers. The GMG output from this test problem is given below:

```
                              MODFLOW-2000
   U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
                         Version 1.12.01 10/03/2003

 Using NAME file: dvr.nam
 Run start date and time (yyyy/mm/dd hh:mm:ss): 2004/06/28 14:53:04

 --------------------------------------------------
 GMG -- PCG GEOMETRIC MULTI-GRID SOLUTION PACKAGE:
 --------------------------------------------------
 RCLOSE  = 1.00E-05; INNER CONVERGENCE CRITERION
 IITER   =      100; MAX INNER ITERATIONS
```

```
HCLOSE  = 1.00E-05; OUTER CONVERGENCE CRITERION
MXIITER =      100; MAX OUTER ITERATIONS
DAMP    = 5.00E-01; DAMPING PARAMETER
IADAMP  =        1; ADAPTIVE DAMPING FLAG
IOUTGMG =        4; OUTPUT CONTROL FLAG
ISM     =        0; SMOOTHER FLAG
ISC     =        0; COARSENING FLAG
RELAX   = 0.00E+00; RELAXATION PARAMETER
-------------------------------------------------
COOLEY'S ADAPTIVE DAMPING METHOD IMPLEMENTED
ILU SMOOTHING IMPLEMENTED
FULL COARSENING
-------------------------------------------------
   5 MEGABYTES OF MEMORY ALLOCATED BY GMG
-------------------------------------------------

ITER:   0  RES: 1.9373E+07  CFAC: 1.000
ITER:   1  RES: 7.7200E+04  CFAC: 0.004
---------------------------------------------------------
PCG ITERATIONS                    : 1
DAMPING                           : 0.500
L2-NORM OF RESIDUAL               : 7.7200E+04
MAX HEAD CHANGE                   : 2.3431E+03
MAX HEAD CHANGE AT (COL,ROW,LAY)  : (32,63,1)
---------------------------------------------------------

ITER:   0  RES: 9.6758E+06  CFAC: 1.000
ITER:   1  RES: 4.8186E+04  CFAC: 0.005
---------------------------------------------------------
PCG ITERATIONS                    : 1
DAMPING                           : 1.000
L2-NORM OF RESIDUAL               : 4.8186E+04
MAX HEAD CHANGE                   : 1.2412E+03
MAX HEAD CHANGE AT (COL,ROW,LAY)  : (32,63,1)
---------------------------------------------------------

ITER:   0  RES: 4.9663E+04  CFAC: 1.000
ITER:   1  RES: 8.1997E+03  CFAC: 0.165
ITER:   2  RES: 4.1679E+03  CFAC: 0.508
ITER:   3  RES: 4.5382E+02  CFAC: 0.109
ITER:   4  RES: 2.2520E+01  CFAC: 0.050
ITER:   5  RES: 4.8002E-01  CFAC: 0.021
ITER:   6  RES: 2.2607E-02  CFAC: 0.047
ITER:   7  RES: 1.6610E-03  CFAC: 0.073
ITER:   8  RES: 1.8021E-04  CFAC: 0.108
ITER:   9  RES: 4.5522E-06  CFAC: 0.025
---------------------------------------------------------
PCG ITERATIONS                    : 9
DAMPING                           : 0.966
L2-NORM OF RESIDUAL               : 4.5522E-06
MAX HEAD CHANGE                   : 6.4636E+01
MAX HEAD CHANGE AT (COL,ROW,LAY)  : (55,72,1)
---------------------------------------------------------
```

```
ITER:   0  RES: 6.4034E+03  CFAC: 1.000
ITER:   1  RES: 2.1542E+03  CFAC: 0.336
ITER:   2  RES: 2.3452E+02  CFAC: 0.109
ITER:   3  RES: 3.1895E+01  CFAC: 0.136
-------------------------------------------------------
PCG ITERATIONS                    : 3
DAMPING                           : 0.500
L2-NORM OF RESIDUAL               : 3.1895E+01
MAX HEAD CHANGE                   : 7.7927E+01
MAX HEAD CHANGE AT (COL,ROW,LAY)  : (55,72,1)
-------------------------------------------------------

ITER:   0  RES: 3.1059E+03  CFAC: 1.000
ITER:   1  RES: 5.4095E+02  CFAC: 0.174
-------------------------------------------------------
PCG ITERATIONS                    : 1
DAMPING                           : 0.704
L2-NORM OF RESIDUAL               : 5.4095E+02
MAX HEAD CHANGE                   : 2.0304E+01
MAX HEAD CHANGE AT (COL,ROW,LAY)  : (55,72,1)
-------------------------------------------------------

ITER:   0  RES: 1.8445E+03  CFAC: 1.000
ITER:   1  RES: 1.3377E+02  CFAC: 0.073
-------------------------------------------------------
PCG ITERATIONS                    : 1
DAMPING                           : 0.777
L2-NORM OF RESIDUAL               : 1.3377E+02
MAX HEAD CHANGE                   : 5.3870E+00
MAX HEAD CHANGE AT (COL,ROW,LAY)  : (80,104,1)
-------------------------------------------------------

ITER:   0  RES: 1.5616E+03  CFAC: 1.000
ITER:   1  RES: 6.9107E+01  CFAC: 0.044
-------------------------------------------------------
PCG ITERATIONS                    : 1
DAMPING                           : 0.500
L2-NORM OF RESIDUAL               : 6.9107E+01
MAX HEAD CHANGE                   : 5.1697E+00
MAX HEAD CHANGE AT (COL,ROW,LAY)  : (80,104,1)
-------------------------------------------------------

ITER:   0  RES: 2.3341E+02  CFAC: 1.000
ITER:   1  RES: 3.5979E+01  CFAC: 0.154
-------------------------------------------------------
PCG ITERATIONS                    : 1
DAMPING                           : 1.000
L2-NORM OF RESIDUAL               : 3.5979E+01
MAX HEAD CHANGE                   : 6.9644E-01
MAX HEAD CHANGE AT (COL,ROW,LAY)  : (57,66,1)
-------------------------------------------------------

ITER:   0  RES: 3.5979E+01  CFAC: 1.000
ITER:   1  RES: 1.3609E+01  CFAC: 0.378
```

```
ITER:   2  RES: 1.5037E+00  CFAC: 0.110
ITER:   3  RES: 1.0185E-01  CFAC: 0.068
ITER:   4  RES: 8.9790E-03  CFAC: 0.088
ITER:   5  RES: 3.9983E-04  CFAC: 0.045
ITER:   6  RES: 1.8652E-05  CFAC: 0.047
ITER:   7  RES: 5.4129E-07  CFAC: 0.029
-----------------------------------------------------------
PCG ITERATIONS                    : 7
DAMPING                           : 1.000
L2-NORM OF RESIDUAL               : 5.4129E-07
MAX HEAD CHANGE                   : 1.5882E-01
MAX HEAD CHANGE AT (COL,ROW,LAY)  : (13,43,3)
-----------------------------------------------------------

ITER:   0  RES: 7.4818E-07  CFAC: 1.000
ITER:   1  RES: 1.1912E-07  CFAC: 0.159
-----------------------------------------------------------
PCG ITERATIONS                    : 1
DAMPING                           : 1.000
L2-NORM OF RESIDUAL               : 1.1912E-07
MAX HEAD CHANGE                   : 5.7020E-10
MAX HEAD CHANGE AT (COL,ROW,LAY)  : (6,36,2)
-----------------------------------------------------------

-------------------------------
TIME STEP            :      1
STRESS PERIOD        :      1
GMG CALLS            :     10
PCG ITERATIONS       :     26
-------------------------------
TOTAL PCG ITERATIONS :     26
-------------------------------

Run end date and time (yyyy/mm/dd hh:mm:ss): 2004/06/28 14:53:10
Elapsed run time:  5.370 Seconds

Normal termination of MODFLOW-2000
```

The above listing shows the effects of adaptive damping and an adaptive inner-iteration convergence criterion. If the damping parameter is less than 1.0, then the effects of nonlinearity are strong and the accuracy for the PCG solution is decreased. When the damping parameter is equal to 1.0, then the nonlinear problem is converging and the accuracy of the PCG solution is increased.

Semi-Coarsening Example

A sample input file for the GMG package is given below for test problem 6:

```
#==============================
# Test 6
#==============================
# RCLOSE  IITER  HCLOSE  MXITER
```

```
#==============================
  1.0E-5  100    1.0E-5  1
#==============================
# DAMP  IADAMP  IOUTGMG
#==============================
  1.0   0       4
#==============================
# ISM  SC
#==============================
  0    1
```

Because this is a linear problem, `MXITER` is set to 1, the damping parameter is set to 1.0, and adaptive damping is disabled. The level of coarsening for the columns and rows is set at maximum, and no coarsening is done for the layers. The GMG output from this test problem is given below:

```
                                MODFLOW-2000
   U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
                           Version 1.12.01 10/03/2003

 Using NAME file: twri_large.nam
 Run start date and time (yyyy/mm/dd hh:mm:ss): 2004/06/28 15:00:03

 -------------------------------------------------
 GMG -- PCG GEOMETRIC MULTI-GRID SOLUTION PACKAGE:
 -------------------------------------------------
 RCLOSE  = 1.00E-05; INNER CONVERGENCE CRITERION
 IITER   =      100; MAX INNER ITERATIONS
 HCLOSE  = 1.00E-05; OUTER CONVERGENCE CRITERION
 MXIITER =        1; MAX OUTER ITERATIONS
 DAMP    = 1.00E+00; DAMPING PARAMETER
 IADAMP  =        0; ADAPTIVE DAMPING FLAG
 IOUTGMG =        4; OUTPUT CONTROL FLAG
 ISM     =        0; SMOOTHER FLAG
 ISC     =        1; COARSENING FLAG
 RELAX   = 0.00E+00; RELAXATION PARAMETER
 -------------------------------------------------
 ILU SMOOTHING IMPLEMENTED
 COARSENING ALONG COLUMNS AND ROWS ONLY
 -------------------------------------------------
   91 MEGABYTES OF MEMORY ALLOCATED BY GMG
 -------------------------------------------------

 ITER:   0  RES: 4.0399E+01  CFAC: 1.000
 ITER:   1  RES: 2.8635E+00  CFAC: 0.071
 ITER:   2  RES: 3.7051E-02  CFAC: 0.013
 ITER:   3  RES: 5.1541E-03  CFAC: 0.139
 ITER:   4  RES: 6.7234E-04  CFAC: 0.130
 ITER:   5  RES: 8.8153E-05  CFAC: 0.131
 ITER:   6  RES: 5.4711E-06  CFAC: 0.062
 ---------------------------------------------------------
 PCG ITERATIONS                    : 6
```

```
DAMPING                          : 1.000
L2-NORM OF RESIDUAL              : 5.4711E-06
MAX HEAD CHANGE                  : 5.5764E+01
MAX HEAD CHANGE AT (COL,ROW,LAY)  : (95,15,20)
-------------------------------------------------------------

-------------------------------
TIME STEP            :      1
STRESS PERIOD        :      1
GMG CALLS            :      1
PCG ITERATIONS       :      6
-------------------------------
TOTAL PCG ITERATIONS :      6
-------------------------------

Run end date and time (yyyy/mm/dd hh:mm:ss): 2004/06/28 15:00:28
Elapsed run time: 25.169 Seconds

Normal termination of MODFLOW-2000
```

ILU PCG Example

Test problem 6 is repeated without any coarsening. Double precision is forced as a compiler option to minimize the effects of rounding error. The reduction histories are suppressed. The input file and the output are given below as follows:

```
#==============================
# Test 6
#==============================
# RCLOSE  IITER  HCLOSE  MXITER
#==============================
  1.0E-5  500    1.0E-5  1
#==============================
# DAMP  IADAMP  IOUTGMG
#==============================
  1.0   0       3
#==============================
# ISM  SC
#==============================
  0    4
#==============================
# RELAX
#==============================
  1.00
```

```
                              MODFLOW-2000
    U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
                          Version 1.12.01 10/03/2003
```

```
Using NAME file: twri_large.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2004/06/28 15:14:55

--------------------------------------------------
GMG -- PCG GEOMETRIC MULTI-GRID SOLUTION PACKAGE:
--------------------------------------------------
RCLOSE  = 1.00E-05; INNER CONVERGENCE CRITERION
IITER   =      500; MAX INNER ITERATIONS
HCLOSE  = 1.00E-05; OUTER CONVERGENCE CRITERION
MXIITER =        1; MAX OUTER ITERATIONS
DAMP    = 1.00E+00; DAMPING PARAMETER
IADAMP  =        0; ADAPTIVE DAMPING FLAG
IOUTGMG =        3; OUTPUT CONTROL FLAG
ISM     =        0; SMOOTHER FLAG
ISC     =        4; COARSENING FLAG
RELAX   = 1.00E+00; RELAXATION PARAMETER
--------------------------------------------------
ILU SMOOTHING IMPLEMENTED
NO COARSENING
--------------------------------------------------
  49 MEGABYTES OF MEMORY ALLOCATED BY GMG
--------------------------------------------------

-------------------------------------------------------
PCG ITERATIONS                    : 452
DAMPING                           : 1.000
L2-NORM OF RESIDUAL               : 9.7500E-06
MAX HEAD CHANGE                   : 5.5764E+01
MAX HEAD CHANGE AT (COL,ROW,LAY)  : (95,15,20)
-------------------------------------------------------

-------------------------------
TIME STEP            :      1
STRESS PERIOD        :      1
GMG CALLS            :      1
PCG ITERATIONS       :    452
-------------------------------
TOTAL PCG ITERATIONS :    452
-------------------------------

Run end date and time (yyyy/mm/dd hh:mm:ss): 2004/06/28 15:17:07
Elapsed run time:  2 Minutes, 12.242 Seconds

Normal termination of MODFLOW-2000
```

Description of GMG Interface

The GMG interface program links the GMG library to MF2K and is activated by file type "GMG" in the name file. The interface program consists of two primary modules: the allocation module (GMG1ALG) and the solver module (GMG1AP). The GMG1ALG module reads the input for the solver and calls the GMG library

to allocate the solver. The GMG1AP module calls the GMG library to assemble and evaluate the linear equation. The interface contains an additional module (RESPRINT) for printing the convergence history of the PCG iteration.

The MF2K main program is modified to recognize the "GMG" file type and call the GMG interface program to allocate, assemble, and evaluate the CCFD problem. These modifications can be made to other versions of MODFLOW. The modifications to the MF2K program Version 1.12 are presented here as a guide.

1. After line 2302 insert the following:

   ```
   IF(IUNIT(42).GT.0) CALL MF2KGMG_FREE()
   ```

   This deallocates the GMG package after each simulation when MF2K is run in batch mode.

2. After line 1900 insert the following:

   ```
              IF (IUNIT(42).GT.0)
   &            CALL GMG1AP(Z(LCSNEW),GX(LCRHS),GX(LCCR),GX(LCCC),
   &                        GX(LCCV),GX(LCHCOF),HNOFLO,IG(LCIBOU),
   &                        IITER,MXITER,RCLOSES,HCLOSES,
   &                        KKITER,KKSTP,KKPER,
   &                        ICNVG,DAMP,IADAMP,IOUTGMG,IOUT)
   ```

   This provides a call to the GMG interface for solving the linear equation in the sensitivity loop.

3. After line 1226 insert the following:

   ```
              IF (IUNIT(42).GT.0)
   &            CALL GMG1AP(GZ(LCHNEW),GX(LCRHS),GX(LCCR),GX(LCCC),
   &                        GX(LCCV),GX(LCHCOF),HNOFLO,IG(LCIBOU),
   &                        IITER,MXITER,RCLOSE,HCLOSE,
   &                        KKITER,KKSTP,KKPER,
   &                        ICNVG,DAMP,IADAMP,IOUTGMG,IOUT)
   ```

   This provides a call to the GMG interface for solving the linear equation for each outer-iteration in the time/stress period loop.

4. After line 305 insert the following:

   ```
      IF(IUNIT(42).GT.0)
   1    CALL GMG1ALG(NCOL,NROW,NLAY,MXITER,IITER,
   2                 RCLOSE,HCLOSE,DAMP,IADAMP,
   3                 IOUTGMG,IUNIT(42),IOUTG)
   ```

   This provides a call to the GMG interface for reading input and allocating the GMG library.

5. Add the GMG file type specifier to position 42 of the CUNIT array on line `105` as follows:

```
 DATA CUNIT/'BCF6', 'WEL ', 'DRN ', 'RIV ', 'EVT ', '    ', 'GHB ',  !  7
&           'RCH ', 'SIP ', 'DE4 ', 'SOR ', 'OC  ', 'PCG ', 'LMG ',  ! 14
&           'gwt ', 'FHB ', 'RES ', 'STR ', 'IBS ', 'CHD ', 'HFB6',  ! 21
&           'LAK ', 'LPF ', 'DIS ', 'SEN ', 'PES ', 'OBS ', 'HOB ',  ! 28
&           'ADV2', 'COB ', 'ZONE', 'MULT', 'DROB', 'RVOB', 'GBOB',  ! 35
&           'STOB', 'HUF2', 'CHOB', 'ETS ', 'DRT ', 'DTOB', 'GMG ',  ! 42
&           'HYD ', 'sfr ', 'SFOB', 'GAGE', 'LVDA', '    ', 'LMT6',  ! 49
&           'MNW1', 'DAF ', 'DAFG', 'KDEP', 'SUB ', '    ', '    ',  ! 56
&           44*'    '/
```

Module GMG1ALG

The GMG1ALG module reads the input file and calls the GMG library to allocate the solver.

1. Read values for `RCLOSE`, `HCLOSE`, `MXITER`, `IITER`, `DAMP`, `IADAMP`, `IOUTGMG`, `ISM`, and `ISC`. If `ISC` $= 4$, then read value for `RELAX`.

2. Check for forced double precision.

3. Call GMG library module `MF2KGMG_ALLOCATE` to allocate the solver.

4. If `IERR` < 0, then call MF2K `USTOP` subroutine to report error and stop execution.

5. If `IOUTGMG` $\neq 0$, then print information about GMG solver.

6. Return.

The GMG library is deallocated by calling the GMG library function `MF2KGMG_FREE()`. When running MF2K in batch mode, deallocation of the library is necessary at the end of each simulation.

Single-precision real-valued variables in MF2K can be extended to double precision at compile time. The GMG library needs to know the size of the internal data being passed in from MF2K so that appropriate pointer arithmetic can be performed. By using the FORTRAN 95 KIND function, GMG is able to automatically detect whether MF2K was compiled as single or double precision.

Module GMG1AP

The GMG1AP module calls the GMG library to assemble the linear equation and the solver, compute a head change, and add the head change to the current approximation. The GMG1AP module also checks for convergence and calculates adaptive damping and adaptive PCG convergence criterion.

1. Initialize variables:

(a) `ICNVG=0`

(b) `IIOUT=IOUT`

(c) `IF(IOUTGMG .GT. 2) IIOUT=6`

(d) `IF(KITER .EQ. 1) DDAMP=DAMP`

(e) `DAMP0=DDAMP`

2. Call GMG library module `MF2KGMG_ASSEMBLE` to assemble the CCFD matrix, the initial residual, and the multigrid preconditioner. The l2-norm of the residual is returned in the variable `BIGR0`.

3. Set PCG convergence criterion:
`DRCLOSE=DDAMP*RCLOSE+(1-DDAMP)*BIGR0`.

4. Calculate the head-change by calling GMG module `MF2KGMG_EVAL`. The l2-norm of the residual is returned in the variable `BIGR`.

5. Calculate maximum head change `BIGH` by calling GMG module `MF2KGMG_BIGH`.

6. Check for convergence:

   (a) If `MXITER`=1 and `BIGR` $\leq$ `RCLOSE`, then set `ICNVG` to 1, `DDAMP` to 1.0, and go to 8.

   (b) If `BIGH` $\leq$ `HCLOSE` and `BIGR` $\leq$ `RCLOSE`, then set `ICNVG` to 1, `DDAMP` to 1.0, and go to 8.

7. If adaptive damping is implemented and it is not the first outer iteration, then compute a new damping parameter using Cooley's method.

8. Add damped head-change to current approximation by calling GMG module `MF2KGMG_UPDATE`.

9. If `IOUTGMG` $\neq 0$, then print linear iteration results.

10. If `ICNVG`=1 and `IOUGMG` $\neq 0$, then print outer iteration results.

11. Return.

Description of GMG Library

The GMG library contains four primary computer codes: the vector library, the solver library, the CCFD library, and the MF2KGMG library. The vector library defines real-valued arrays and a generic operator. The solver library defines a generic PCG algorithm and a generic multigrid algorithm. The CCFD library defines a CCFD matrix, an ILU smoother, and the components used for the CCFD multigrid method. The MF2KGMG library acts as a secondary interface allocating and assembling operators from MF2K data (internal arrays) passed in through the GMG interface modules.

These libraries can be viewed as forming a hierarchy of operators and methods. The vector library forms the base of the hierarchy and is used by the solver library to define its operators and methods. The vector library also includes built-in C definitions used by all the libraries. The CCFD library uses operators and methods from the solver library and vector library. The MF2KGMG library uses the vector library, the solver library, and the CCFD library.

Some of the topics in the following sections have been described in previous sections. In particular, the "Input Instructions for the GMG Solver" section and the "Description of GMG Interface" section should be referred to as needed.

Vector Library

Vectors represent nodal values on an $l \times m \times n$ grid. The number of nodal values, or equations, depends on the application. For CCFD problems, the number of nodal-values is given by `neq`$= l * m * n$. An `r_data` object contains values for `neq`, l, m, and n. The `r_data` structure is defined in the C language as follows:

```
typedef struct r_data
{
  int l,m,n;
  int neq;
}r_data;
```

An `r_vector` object contains an array of nodal values and a pointer to an `r_data` object. The definition of an `r_vector` is given by

```
typedef struct r_vector
{
  double* vec; /* Array of double */
  r_data* rdp; /* Pointer to r_data object */
}r_vector;

int r_allocate(r_vector* r_ptr, r_data* rdp);
void r_free(r_vector* r_ptr);
```

Level 1 basic linear algebra subroutines (BLAS) also are defined in the vector library. An example of a Level 1 BLAS method is the AXPY (alpha times x plus y) operation given by $y \leftarrow \alpha x + y$. The Level 1 BLAS methods use loop unrolling for cache optimization.

The generic operator defined in the vector library computes the action of a generic object on a `r_vector` object, storing the results in another `r_vector` object. This is facilitated by the ability of the C language to define pointers to functions as well as objects. The generic operator contains a pointer to an operator and a pointer to the operator's evaluation method. It also contains a pointer to the operator's deallocation method. The generic operator definition is given by the following structure:

```
typedef struct GEN_operator
{
```

```
  void* A_ptr;
  int (*A_eval)(r_vector*,r_vector*,void*);
  void (*A_free)(void*);
}GEN_operator;
```

The generic operator is assembled by

```
int GEN_assemble(GEN_operator* GEN_ptr, void* A_ptr,
                 int (*A_eval)(r_vector*,r_vector*,void*),
                 void (*A_free)(void*))
{
  GEN_ptr->A_ptr=A_ptr;
  GEN_ptr->A_eval=A_eval;
  GEN_ptr->A_free=A_free;

  return 0;
}
```

The `void` datatype is a special datatype used commonly as a return type for functions that do not return a value. Normally, a variable cannot be declared as type `void`, but declaring a pointer to `void` is allowed. A pointer to any type can be converted to a pointer to void and, conversely, a pointer to void can safely be converted to pointer to any other type. The generic operator is evaluated and deallocated by the following:

```
int GEN_eval(r_vector* r2_ptr, r_vector* r1_ptr, GEN_operator* G_ptr)
{
  if(G_ptr->A_eval==NULL)
  {
    r_copy(r2_ptr,r1_ptr);
    return 0;
  }
  return (*G_ptr->A_eval)(r2_ptr,r1_ptr,G_ptr->A_ptr);
}

void GEN_free(GEN_operator* G_ptr)
{
  if(G_ptr->A_free != NULL)
    (*G_ptr->A_free)(G_ptr->A_ptr);
}
```

Any or all of the data members of the generic operator can be set to NULL. If `A_eval` is set to NULL, then it is equivalent to the identity operator. The `A_ptr` data member can be set to NULL if there is no object associated with the operation. The `A_free` data member can be set to NULL if the operator is allocated and assembled outside the normal context of the operator. The generic deallocation method is convenient because every operator can be deallocated using the same `GEN_free` method.

The generic operator acts as a container for other operators. Operators have four primary methods; allocation, deallocation, assembly, and evaluation. The first argument in the allocation and assembly methods of an operator is a pointer to a `GEN_operator`. Other arguments are provided depending on the requirements of the operator. The allocation and assembly methods for a hypothetical operator `X` follow:

```
int X_allocate(GEN_operator* GEN_ptr, arg2,...)
{
  X_operator* X_ptr;
  int size,total_size=0;

  X_ptr=(X_operator*)calloc(1,sizeof(X_operator));
  if(X_ptr==NULL)
    return -1;
  total_size+=sizeof(X_operator);

  GEN_assemble(GEN_ptr,X_ptr,X_eval,X_free);

  /* Allocate the data members of X. */

  return total_size;
}

int X_assemble(GEN_operator* GEN_ptr, arg2,...)
{
  X_operator* X_ptr=GEN_ptr->A_ptr;

  /* Assemble the data members of X. */

  return 0;
}
```

Most methods return an integer value. For allocation methods, the return value indicates the number of bytes allocated. However, a negative return value indicates a failure in the method.

The evaluation method of an operator takes exactly three arguments. The first and second arguments are pointers to `r_vector` objects and the third argument is a pointer to `void`. The deallocation method takes exactly one argument; a pointer to `void`. The `X` operator evaluation and deallocation methods are defined as follows:

```
int X_eval(r_vector* r2_ptr, r_vector* r1_ptr, void* A_ptr)
{
  X_operator* X_ptr=A_ptr;

  /* Compute the action of X on a vector. */

  return 0;
}

void X_free(void* A_ptr)
{
  X_operator* X_ptr=A_ptr;

  /* Deallocate the data members of X. */

  free(X_ptr);
}
```

Note that the pointer to `void` is converted to a pointer to an `X` object in the above examples. A pointer should be thought of simply as an integer value representing the address of some specific location in memory. Prior to accessing the data at that address, the type of data must be resolved. This is accomplished at run time by casting the pointer to reference a specific datatype.

Solver Library

The generic operator definition given in the vector library is used to define a generic PCG operator in the solver library. Multilevel `r_vector` objects and multilevel generic operators are defined in the solver library to construct a generic multigrid operator.

PCG Operator

A generic PCG operator was introduced in Algorithm 3. The C language definition of this operator follows:

```
typedef struct PCG_operator
{
  /* Work vectors. */
  r_vector r,p,z;

  /* Iteration parameters. */
  double RCLOSE;
  int IITER,IOUT,IOUTGMG;

  /* Pointers to generic operators. */
  GEN_operator *A_ptr,*B_ptr;

  int ow; /* Overwrite flag. */

  double BIGR; /* Final Residual */
}PCG_operator;

int PCG_allocate(GEN_operator* PCG_GEN_ptr, int ow, r_data* rdp);
void PCG_free(void* A_ptr);
int PCG_assemble(GEN_operator* PCG_GEN_ptr,
                 GEN_operator* A_GEN_ptr,
                 GEN_operator* B_GEN_ptr);
int PCG_set(GEN_operator* PCG_GEN_ptr, int IITER,
            double RCLOSE, int IOUTGMG, int IOUT);

int PCG_eval(r_vector* d2_ptr, r_vector* d1_ptr, void* A_ptr);
```

The overwrite flag `ow` indicates whether the right-hand side is to be overwritten by the residual, saving the memory cost of one work vector. In applying GMG to MF2K, the right-hand side is the initial residual and is allowed to be overwritten by the PCG operator. The CCFD and multigrid operators are given to the PCG operator as pointers to generic operators. The maximum number of iterations, the residual

stopping criterion, the print flag, and FORTRAN unit number are passed to the PCG operator through the `PCG_set` method. The PCG operator can be defined recursively allowing a PCG operator to be a preconditioner in the PCG operator.

Multigrid Operator

Multilevel `r_vector` objects and multilevel generic operators are defined in the solver library as follows:

```
typedef struct mg_data
{
  r_data* rd_list; /* Array of r_data objects */
  int levels;
}mg_data;

int mg_data_allocate(mg_data* mgdp, levels);

void mg_data_free(mg_data* mgdp);

typedef struct mg_vector
{
  mg_data* mgdp;    /* Pointer to mg_data object */
  int i0,i1;
  r_vector* r_list; /* Array of r_vector objects */
}mg_vector;

int mg_vector_allocate(mg_vector* mgp, int i0, int i1, mg_data* mgdp);
void mg_vector_free(mg_vector* mgp);

typedef struct MG_GEN_operator
{
  mg_data* mgdp;          /* Pointer to mg_data object */
  GEN_operator* GEN_list; /* Array of generic operators */
}MG_GEN_operator;

int MG_GEN_allocate(MG_GEN_operator* MG_GEN_ptr, mg_data* mgdp);
void MG_GEN_free(MG_GEN_operator* MG_GEN_ptr);
```

The `mg_data` object contains an array of `r_data` objects (one for each level) and the number of multigrid levels. The number of levels and the contents of the `r_data` array are problem dependent and assembled outside the context of the generic multigrid (MG) operator. In the application of GMG to MF2K, the number of levels and the contents of the `r_data` array are assembled in the CCFD library.

The `mg_vector` object contains a pointer to an `mg_data` object and an array of `r_vector` objects, where an `r_vector` object is allocated on each level s for `i0`$\leq s \leq$`i1`. Specifying a range of levels eliminates unnecessary allocation of work vectors on certain levels. The `MG_GEN_operator` object contains a pointer to an `mg_data` object and an array of `GEN_operator` objects.

Generic versions of the ν-Cycle (Algorithm 12) and the smoother (Algorithm 4) are implemented as methods in the MG operator. The MG operator contains multilevel

vectors and pointers to multilevel generic operators. The multilevel generic operators are assembled outside the context of the MG operator and passed as pointers to the MG object. In applying GMG to MF2K, the multilevel generic operators are defined in the CCFD library; the MG operator accesses these methods indirectly. The MG definition follows:

```
typedef struct MG_operator
{
  mg_data* mgdp; /* Pointer to mg_data object */

  /* Overwrite Flag */
  int ow;

  /* Multilevel vectors */
  mg_vector mgp; /* Multilevel solution vector */
  mg_vector mgb; /* Multilevel right-hand side */
  mg_vector mgr; /* Multilevel residual vector */

  MG_GEN_operator* A_ptr;   /* Pointer to multilevel coefficient matrix */
  MG_GEN_operator* B_ptr;   /* Pointer to multilevel Smoother */
  MG_GEN_operator* P_ptr;   /* Pointer to multilevel prolongation operator */
  MG_GEN_operator* R_ptr;   /* Pointer to multilevel restriction operator */

  int mu0;        /* Number of smoothing iterations. */
  int mu1;        /* Number of multigrid cycles. */
  int nu;         /* V-Cycle(nu=1), W-Cycle(nu=2) */

}MG_operator;

int MG_allocate(GEN_operator* MG_ptr, int ow, mg_data* mgdp);
void MG_free(void* MG_free);
int MG_assemble(GEN_operator* MG_GEN_ptr,
                MG_GEN_operator* A_ptr,
                MG_GEN_operator* B_ptr,
                MG_GEN_operator* P_ptr,
                MG_GEN_operator* R_ptr);
void MG_set(GEN_operator* MG_ptr, int mu0, int mu1, int nu);

int MG_eval(r_vector* p2_ptr, r_vector* p1_ptr, void* A_ptr);
```

A recursive application of the MG operator can be used to implement multigrid smoothers in the multigrid method. For example, planar relaxation is a smoother described in Trottenberg and others (2001) for three-dimensional problems with anisotropic coefficients. Planar relaxation involves approximating a series of two-dimensional problems in the x, y-planes, y, z-planes, and z, x-planes; each may be approximated by the multigrid method using line relaxation for the smoother.

If the overwrite flag `ow` is non-zero, then it is assumed that the smoothing method is able to overwrite its right-hand side. This eliminates the need for an additional work vector on each level. The ILU smoother is able to overwrite the right-hand side.

CCFD Library

The CCFD library allows for either single-precision or double-precision conductances. This makes the CCFD library compatible with different compilations of MF2K where double precision may or may not be forced. The structure of the CCFD operator is given by

```
typedef struct CCFD_operator
{
  r_data* rdp;       /* Pointer to r_data object */
  void *CC,*CR,*CV;  /* Conductance arrays */
  double *DD;        /* Diagonal */
  int *IBOUND;       /* Indicates specified heads */
  int prec;          /* prec not zero indicates double precision */
}CCFD_operator;
```

The GMG interface program determines the precision of the conductance arrays at run time and passes this information into the GMG library so that it can resolve pointers to internal MF2K data. The data members `CC`, `CR`, and `CV` are defined as pointers to `void` and are resolved to reference the appropriate datatype, depending on the value of the `prec` data member. On the finest grid, the `DD` data member is allocated and assembled in the MF2KGMG library, outside the context of the CCFD library, and the `CC`, `CR`, `CV`, and `IBOUND` data members are given the addresses of MF2K internal arrays. This eliminates unnecessary duplication of internal MF2K data.

The CCFD_ILU operator (Algorithms 5 and 6) contains a pointer to a CCFD operator. The diagonal of the factorization `pivots` is stored as an `r_vector` object. The CCFD_ILU operator is defined as follows:

```
typedef struct CCFD_ILU_operator
{
  r_vector pivots;
  CCFD_operator* CCFD_ptr; \* Pointer to CCFD operator *\
}CCFD_ILU_operator;

int CCFD_ILU_allocate(GEN_operator* GEN_ptr, r_data* rdp);
void CCFD_ILU_free(void* A_ptr);
int CCFD_ILU_assemble(GEN_operator* GEN_ILU_ptr,
                      GEN_operator* GEN_CCFD_ptr,
      double RELAX);
int CCFD_ILU_eval(r_vector* p2_ptr, r_vector* p1_ptr, void* A_ptr);
```

The `RELAX` parameter is used to assemble a modified ILU factorization (see Remark 6).

The CCFD_MG operator contains multilevel generic operators defined in the solver library. These multilevel operators are allocated and assembled within the context of the CCFD_MG operator and passed as pointers to the MG operator. The CCFD_MG operator definition follows:

```
typedef struct CCFD_MG_operator
{
  /* Multigrid Operators */
  mg_data mgd;            /* Multilevel Data */
  MG_GEN_operator MGCCFD; /* Multilevel CCFD Matrix */
  MG_GEN_operator MGB;    /* Multilevel Smoother */
  MG_GEN_operator MGP;    /* Multilevel Prolongation */
  MG_GEN_operator MGR;    /* Multilevel Restriction */
  GEN_operator MG;        /* Multigrid Operator */
  int SM;                 /* Smoother, SM=0->ILU; Otherwise, SGS */
  double RELAX;           /* Relaxation parameter for ILU smoother. */
}CCFD_MG_operator;

int CCFD_MG_allocate(GEN_operator* GEN_ptr,
                     GEN_operator* GEN_CCFD_ptr,
                     r_data* rdp, int ISM, int ISC, double RELAX);

void CCFD_MG_free(void* A_ptr);

int CCFD_MG_assemble(GEN_operator* GEN_CCFD_MG_ptr);

void CCFD_MG_set(GEN_operator* GEN_CCFD_MG_ptr,
                 int mu0, int mu1, int nu);

int CCFD_MG_eval(r_vector* p2_ptr, r_vector* p1_ptr, void* A_ptr);
```

On the finest grid, the multilevel CCFD operator `MGCCFD` references the fine-grid CCFD operator assembled in the MF2KGMG library. On coarser levels, the CCFD operators are assembled using Algorithms 9 and 10. Once the multilevel CCFD operator has been assembled, the multilevel smoothing operators are assembled. The smoothers can be either ILU or symmetric Gauss-Seidel. The `ISC` parameter controls the coarsening (see "Input Instruction for the GMG Solver").

The prolongation/restriction operators, defined by Algorithms 7 and 8, do not require a structure, or allocation/deallocation methods. The prolongation/restriction operators are assembled as follows:

```
  P_list=CCFD_MG_ptr->MGP.GEN_list;
  R_list=CCFD_MG_ptr->MGR.GEN_list;

  for(i=0;i<levels;i++)
  {
    GEN_assemble(&P_list[i],NULL,CCFD_P_eval,NULL);
    GEN_assemble(&R_list[i],NULL,CCFD_R_eval,NULL);
  }
```

MF2KGMG Library

The MF2KGMG library uses MF2K internal arrays and data, passed in from the GMG interface program, for assembling and evaluating the CCFD problem. Vector

objects, the CCFD operator, the PCG operator, and the CCFD multigrid operator are defined as static variables in the MF2KGMG library as follows:

```
/* Static global variables for CCFD problem. */

static r_data rd;               /* Vector Data */
static r_vector r;              /* Residual */
static r_vector z;              /* Head-Change */
static GEN_operator CCFD;       /* Cell-Centered Finite Difference Matrix */
static GEN_operator CCFDMG;     /* CCFD Multigrid Operator */
static GEN_operator PCG;        /* Preconditioned Conjugate Gradient */
```

Four primary methods for allocating and evaluating the CCFD problem are defined in the MF2KGMG library. A brief description of each method follows:

```
void MF2KGMG_ALLOCATE(int* NCOL, int* NROW, int* NLAY,
                      int* IPREC, int* ISM, int* ISC,
                      int* ISIZ, int* IERR);
```

- The `MF2KGMG_ALLOCATE` method allocates and initializes the GMG solver. The `NCOL`, `NROW`, and `NLAY` arguments represent the number of columns, rows, and layers, respectively, and are used to initialize the `rd` object. The value pointed to by `IPREC` is given to the `CCFD` object where a value of 0 indicates single precision; otherwise, double precision is indicated. The values pointed to by `ISM` and `ISC` are given to the `CCFDMG` object to indicate the type of smoothing and the type of coarsening, respectively (see "Input Instruction for the GMG Solver"). The number of bytes allocated (in MB) and an error flag are passed back to the GMG interface program through `ISIZ` and `IERR`, respectively. If all the operators are allocated successfully, then the value of `IERR` is zero; otherwise, it has a negative value, indicating an assembly error.

```
void MF2KGMG_ASSEMBLE(void* CR, void* CC, void* CV,
                      void* HCOF, double* HNEW, void* RHS,
                      void* HNOFLO, int* IBOUND, double* RES0, int* IERR);
```

- The `MF2KGMG_ASSEMBLE` method assembles the fine-grid `CCFD` object and residual using Algorithm 1. The addresses of the conductance arrays `CR`, `CC`, and `CV` and the address of the `IBOUND` array are given to the `CCFD` object. The precision of the `HCOF`, `RHS`, and `HNOFLO` arguments are resolved by testing the precision flag stored in the `CCFD` object. Once the `CCFD` object is assembled, then the other objects are assembled using methods from the CCFD library and the solver library. The l2-norm of the residual is returned in the variable `RES0`. If all the operators are assembled successfully, then the value of `IERR` is zero; otherwise, it has a negative value, indicating an assembly error.

```
void MF2KGMG_EVAL(int* ITER, double* BIGR, double* BIGH, double* RCLOSE,
                  int* IITER, int* IOUTGMG, int* IOUT);
```

- The MF2KGMG_EVAL method computes the head change by calling the PCG method. The number of iterations performed by the PCG method is returned in the variable ITER. The maximum head change is returned in the variable BIGH. The value of BIGH is used in the GMG interface program to compute the adaptive damping parameter and to check for convergence of the Picard iteration. The absolute value of BIGH is the max-norm of the head change. The l2-norm of the residual is returned in the variable BIGR. The values pointed to by the RCLOSE, IITER, IOUTGMG, and IOUT arguments are passed to the PCG operator to set the stopping criterion, maximum iterations, level of output, and the FORTRAN unit number respectively.

```
void MF2KGMG_BIGH(int* BIGHC, int* BIGHR, int* BIGHL, double* BIGH);
```

- The MF2KGMG_BIGH method computes the maximum head change BIGH. The location of the maximum head change is given by COL, ROW, and LAY (column, row, layer). The absolute value of BIGH is the max-norm of the head change.

```
void MF2KGMG_UPDATE(double* HNEW, double* DAMP);
```

- The MF2KGMG_UPDATE method takes the current head approximation HNEW and adds the damped head change to it.

An additional method in the MF2KGMG library for deallocating the GMG solver uses the following function:

```
void MF2KGMG_FREE()
{
  CCFD_operator *CCFD_ptr=CCFD.A_ptr;

  GEN_free(&MF2KGMG_ptr->CCFDMG);
  GEN_free(&MF2KGMG_ptr->PCG);
  r_free(&MF2KGMG_ptr->r);
  r_free(&MF2KGMG_ptr->z);

  /* CCFD operator deallocated outside of normal context */
  free(CCFD_ptr->DD);
  free(CCFD_ptr);

  return;
}
```

References

Atkinson K., 1988, An Introduction to Numerical Analysis: New York, John Wiley & Sons Inc., 693 p.

Barrett R., Berry M., Chan T., Demmel J., Donato J., Dongarra J., Eijkhout V., Pozo R., Romine C., and Van der Vorst H., 1994, Templates for the Solution of Linear Systems — Building Blocks for Iterative Methods, 2nd Edition: Philadelphia, SIAM, 112 p.

Benzi M., 2002, Preconditioning techniques for large linear systems: A survey: Journal of Computational Physics, v. 182, p. 418–477.

Bramble J., Pasciak J., and Xu J., 1991, The analysis of multigrid algorithms with nonnested spaces or noninherited quadratic forms: Mathematics of Computation, v. 56, no. 193, p. 1–34.

Bramble J., Ewing R., Pasciak J., and Shen J., 1996, The analysis of multigrid algorithms for cell centered finite difference methods: Advances in Computational Mathematics, v. 5, no. 1, p. 15–29.

Briggs W., 1987, A Multigrid Tutorial: Philadelphia, SIAM, 88 p.

Detwiler L., Mehl S., Rajaram H., and Cheung W., 2002, Comparisions of an algebraic multigrid algorithm to two iterative solvers used for modeling ground water flow and transport: Ground Water, v. 40, no. 3, p. 267–272.

Dupont T., Kendall R., and Rachford H., 1968, An approximate factorization procedure for solving self-adjoint elliptic difference equations: SIAM Journal on Numerical Analysis, v. 5, p. 559–573.

Ewing R. and Shen J., 1993, A multigrid algorithm for the cell-centered finite difference scheme, *in* The Proceedings of the Sixth Copper Mountain Conference on Multigrid Methods, Copper Mountain Colorado, April, 1993: NASA Conference Publication 3224.

Golub G. and Van Loan C., 1989, Matrix Computations: Baltimore, London, John Hopkins University Press, 642 p.

Gustaffson I., 1978, A class of first-order factorization methods: BIT, v. 18, p. 142–156.

Hackbush W., 1985, Multi-Grid Methods and Applications: New York, Springer-Verlag, 377 p.

Harbaugh A., Banta E., Hill M., and McDonald M., 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.

Hill M., 1990, Preconditioned Conjugate-Gradient 2 (PCG2), A computer program for solving ground-water flow equations: U.S. Geological Survey Water-Resources Investigations Report 90-4048, 43 p.

Kelley C., 1995, Iterative Methods for Linear and Nonlinear Equations: Frontiers in Applied MathematicsPhiladelphia, , SIAM, 166 p.

McCormick S., ed., 1987, Multigrid Methods: SIAM, 286 p.

McDonald M. and Harbaugh A., 1988, A modular three-dimensional finite-difference ground-water flow model: U.S. Geological Survey Techniques of Water-Resources Investigations, Book 6, Chapter A1, 548 p.

Mehl S. and Hill M., 2001, Modflow-2000, The U.S. Geological Survey modular ground-water model user guide to the LINK-AMG (LMG) package for solving matrix equations using an algebraic multigrid solver: U.S. Geological Survey Open-File Report 01-177, 34 p.

Ruge J. and Stüben K., 1987, Algebraic multigrid, *in* S. McCormick, ed., Multigrid Methods: SIAM, p. 73–130.

Trottenberg U., Oosterlee C., and Schüller, A., 2001, Multigrid: San Diego, Academic Press, 631 p.

Van der Vorst H., 1990, The convergence behavior of preconditioned cg and cg-s in the presence of rounding error, *in* O. Axelsson and L. Kolotilina, eds., Lecture notes in mathematics: New York, London, Springer Verlag, 1457.

Wesseling P., 1991, An Introduction to Multigrid Methods: New York, John Wiley & Sons, 284 p.

Xu J., 1992, Iterative methods by space decomposition and subspace corrections: SIAM Review, v. 34, no. 4, p. 581–613.