Three-Dimensional Steady-State Groundwater Flow Modeling with Full Tensor Conductivities Using Finite Differences

Master Thesis submitted by
Liangping Li

Advisor:
J. Jaime Gómez-Hernández

Group of Hydrogeology
Universidad Politécnica de Valencia
September 2009

Three-Dimensional Steady-state Groundwater Flow Modeling with Full Tensor Conductivities Using Finite Differences

Master Thesis submitted by
Liangping Li

Advisor:
J. Jaime Gómez-Hernández

Master Program:
Ingeniería Hidráulica y Medio Ambiente

Department:
Ingeniería Hidráulica y Medio Ambiente

Group of Hydrogeology
Universidad Politécnica de Valencia

September 2009

Abstract

A new three-dimensional steady-state groundwater-flow forward-simulator with full conductivity tensors using a nineteen-points block-centered finite-difference method is presented. Hydraulic conductivity tensors are defined at the block interfaces eliminating the need to average conductivity tensors at adjacent blocks to approximate their values at the interfaces. The capabilities of the code are demonstrated in three heterogeneous formulations, two of the examples are two-dimensional, and the third one is three-dimensional and uses a nonuniform discretization grid. A benchmark, in the context of conductivity upscaling, is carried out with the MODFLOW LVDA module, which uses hydraulic conductivity tensors at block centers and then approximates their values at the interfaces. The results show that the code developed outperforms the MODFLOW LVDA module when the block conductivity principal directions are not parallel to the Cartesian axis.

Resumen

Esta tesis presenta un nuevo algoritmo, conjuntamente con un código numérico para la solución de la ecuación de flujo estacionario de agua en el subsuelo en tres dimensiones usando el método de diferencias finitas con un esquema de diecinueve puntos centrado en los bloques. La conductividad hidráulica es representada como un tensor que hay que definir en las interfaces entre bloques; de esta manera, se evita la necesidad de promediar tensores en bloques adyacentes para obtener su valor en la interfaz. Las capacidades del código numérico se demuestran en tres formaciones heterogéneas, dos de los ejemplos son bidimensionales, y el tercero tridimensional; éste último utiliza una discretización no uniforme. Los resultados obtenidos con este código se comparan con los obtenidos con el paquete LVDA de MODFLOW, que es capaz de incorporar tensores genéricos representativos de los bloques, pero que necesita promediarlos para determinar las conductividades hidráulicas en la interfaz. Los resultados muestran que el código numérico desarrollado mejora los resultados obtenidos con el paquete LVDA de MODFLOW cuando las direcciones principales de los tensores no son paralelas a los ejes cartesianos.

Resum

Aquesta tesi presenta un nou algorisme, conjuntament amb un codi numèric per a la solució de l'equació de flux estacionari d'aigua en el subsòl en tres dimensiones utilitzant el mètode de diferències finites amb un esquema de dinou punts centrat en els blocs. La conductivitat hidràulica és representada com un tensor que cal definir en les interfases entre blocs; d'aquesta manera s'evita la necessitat de promediar tensors en blocs adjacents per a obtindre el seu valor en la interfase. Les possibilitats del codi numèric es demostren en tres formacions heterogènies, dos dels exemples són bidimensionals, i el tercer tridimensional; aquest últim utilitza una discretització no uniforme. Els resultats obtinguts amb aquest codi es comparen amb els obtinguts amb el mòdul LVDA de MODLFOW, que és capaç d'incorporar tensors genèrics representatius dels blocs, però que necessita promediar-los per a determinar les conductivitats hidràuliques a la interfase. Els resultats mostren que el codi numèric desenvolupat millora els resultats obtinguts amb el módul LVDA de MODFLOW quan les direccions principals dels tensors no són paral·leles als eixos cartesians.

Acknowledgements

I wish to thank my advisor Prof.J.Jaime Gómez-Hernández for providing me the opportunity, financial support and guidance to carry out this study. His help, stimulating suggestions and encouragement helped me in all the time of research.

This research was supported by European Commission (projects FUNMIG and PAMINA) and ENRESA, Empresa Nacional de Residuos Radioactivos S.A.,(project 0078000067).

I would like to say thanks to my colleagues in the hydrogeology group(Andrés, Eduardo, Javier, Jianlin, Gero, Salvador, Oscar). Especially, thanks to Gero for his valuable advice about LVDA package in MODFLOW, thanks also to Haiyan, either the interesting discussion about the research or the life in the abroad, everything was shared with us.

Finally, I would like to thank my parents and the rest of our families. Without their support, I would not have made it.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and Objectives

Numerical simulation has become a common approach to evaluate the groundwater resources in the last several decades. Normally, computer codes only take scalar hydraulic conductivities, or at most tensorial conductivities with their principal axes aligned with the Cartesian axes. However, this assumption fails, for instance, when modeling cross-bedded formations or mildly heterogeneous ones. To overcome this shortcoming it is necessary to use a full tensor (with non-zero off-diagonal components) description of hydraulic conductivity for flow simulation.

The commonly used MODFLOW code (Harbaugh et al., 2000) can handle generic tensors through the use of the LVDA package (Anderman et al., 2002). However, the approach is not fully three-dimensional (one of the principal directions must be the vertical one), thus not allowing for generic anisotropy, and uses as input the tensors at block centers, thus requiring the averaging of tensors at adjacent blocks in order to determine the interblock value needed by the finite-difference approximation.

The objective of this thesis is to develop a flow simulation algorithm for full hydraulic conductivity tensors using finite differences. Instead of requiring the input of hydraulic conductivity tensors at block centers, the code assumes that these tensors are provided directly at the interface. The thesis also discusses how this code can be used in combination with upscaling routines that could provide good estimates of these tensors at block interfaces.

1.2 Thesis Organization

The thesis is organized as follows, Chapter 1 justifies the motivation and objectives of the thesis. Then, Chapter 2 contains a paper that has been submitted to an international journal. Section 2.1 provides an introduction of the flow simulator and the importance of using full hydraulic conductivity tensors for accurate groundwater flow modeling. Section 2.2 describes the formulae derivation of the proposed algorithm. The detail of the implementation is given in section 2.3. In section 2.4, two 2D synthetic numerical experiments and a 3D example are used to test the proposed scheme. Some general conclusions are summarized in section 2.5.

Appendix A presents the numerical code of the flow modeling with full tensor by finite-difference used in the thesis. Appendix B shows an example parameter file.

submitted to Computers & Geosciences

2

Three-Dimensional Steady-state Groundwater Flow Modeling with Full Tensor Conductivities Using Finite Differences

Abstract

We present a new three-dimensional steady-state groundwater-flow forward-simulator with full conductivity tensors using a nineteen-points block-centered finite-difference method. Hydraulic conductivity tensors are defined at the block interfaces eliminating the need to average conductivity tensors at adjacent blocks to approximate their values at the interfaces. The capabilities of the code are demonstrated in three heterogeneous formulations, two of the examples are two-dimensional, and the third one is three-dimensional and uses a nonuniform discretization grid. A benchmark, in the context of conductivity upscaling, is carried out with the MODFLOW LVDA module, which uses hydraulic conductivity tensors at block centers and then approximates their values at the interfaces. The results show that our code outperforms the

MODFLOW LVDA module when the block conductivity principal directions are not parallel to the Cartesian axis.

2.1 Introduction

One of major problems faced by hydrologists is how to deal with the spatial variability of hydraulic conductivities. In the early stages of numerical groundwater flow modeling, it was common to assume very mild heterogeneities, limited to a layer-cake description of the aquifer and some zonation of conductivities within each layer. A standard seven-point block-centered finite-difference approach is typically employed to solve the partial differential equation in three dimensions, assuming an alignment of the block sides and principal directions (Harbaugh et al., 2000), and using the arithmetic mean (upper bound,Penman (1988)) or the harmonic mean (lower bound, Duvaut et al. (1976)) of neighboring block conductivities to approximate their values at block interfaces, needed by the finite-difference formulation. However, these approaches have two shortcomings: first, they assume that the block conductivity is, at most, a diagonal tensor, that is, it could be anisotropic to flow but always with its principal directions parallel to the Cartesian axes; and second, they generally assume that the harmonic mean is the value that best approximate the conductivities. Both assumptions are flawed when we consider that the numerical model is always a simplification of a very heterogeneous spatial distribution of hydraulic conductivities at a scale much smaller than that of the discretization. For one thing, the spatial heterogeneity within the numerical block could seldom be upscaled to a diagonal tensor, in general the tensor would be non-diagonal, particularly in environments in which cross-bedding is observed (Bierkens and Weerts, 1994), or where heterogeneity is high; for another, even if the exact approximate for the equivalent conductivity of two homogeneous blocks with flow across their interface is their harmonic mean, this is not the best approximation when we consider that each block value is already some upscaled value of the underlying conductivities, in such a case the best upscaled value for the interface could be a different average value, moreover, if the block conductivities are tensors, it is not clear how to average two tensors.

Cross-bedding (Bierkens and Weerts, 1994), blocks the size of which is about the correlation length of the underlying conductivity (Wen and Gómez-Hernández, 1996), misalignment between stratification and computational grid (Bear, 1972) are some of the causes that would require an analysis with full conductivity tensors, or, in other words, that would imply that the flow direction may not be parallel to the piezometric head gradient.

Several authors have approached the problem of groundwater flow simulation with non-null off-diagonal components in the conductivity tensor. Aavatsmark et al. (1996) derived a method that is proposed for control volume formulations on quadrilateral grids in two dimensions. Continuity conditions at the interfaces are enforced to construct the discretization method for non-orthogonal curvilinear grids. Edwards and Rogers (1998) developed a general theory for curvilinear coordinates but present a full set of coefficients for only the homogeneous case. Later, Anderman et al. (2002) extended this method for the heterogeneous case, and incorporated it in MODFLOW in two dimensions, namely in the package layer variable-direction horizontal anisotropy capacity (LVDA). An alternative is to use finite elements, or better control-volume finite-element methods (Rozon, 1989), however, the finite element formulation will always introduce unwanted discontinuities at control-volume boundaries (Aavatsmark et al., 1996).

In this context, we present a nineteen-point block-centered finite-difference code, programmed in C, which solves the steady-state groundwater flow equation with full tensor conductivities in three dimensions on a nonuniform grid. Because finite difference formulations use the conductivity values at block interfaces, the program takes as input values the interface conductivity tensors directly, i.e., it will not approximate them from block values. This approach is particularly suitable when the numerical model is a coarse model derived from a fine scale heterogeneous model through upscaling Zhou et al. (2010).

Our objectives in this paper are: (1) to present a new forward simulator with full conductivity tensors; (2) to compare its performance with the LVDA package in MOFLOW; (3) to describe how to treat Dirichlet and Neumann boundary conditions with irregularly shaped boundaries; (4) to demonstrate the capabilities of the code in three dimensions with a non-uniform grid.

The outline of this paper is as follows. We first introduce the flow governing equation and discuss the simulation algorithm. Next, the flowchart of the algorithm is illustrated with an emphasis on the numerical implementation. Then, the capabilities of the proposed approach are demonstrated using two varieties of heterogeneous formations, in the context of upscaling, making use if the upscaling code written by Zhou et al. (2010) and presented in an accompanying paper. Finally, extensive numerical tests for nonuniform blocks in three dimensions are shown to demonstrate the accuracy and efficiency of the proposed simulator.

2.2 Flow Simulator Algorithm

The steady-state groundwater flow equation of an incompressible or slightly compressible fluid in saturated porous media in a Cartesian coordinate system

can be expressed as:

$$\nabla \cdot (\mathbf{K}\nabla h) + q = 0 \tag{2.1}$$

where $h = h(\mathbf{x}, t)$ is the piezometric head $[L]$; $q = q(\mathbf{x}, t)$ is the volumetric source flow per unit volume $[T^{-1}]$; $\nabla\cdot = (\frac{\partial}{\partial x} + \frac{\partial}{\partial y} + \frac{\partial}{\partial z})$ is the divergence operator of a vector field; and $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})^T$ is the gradient operator of a scalar field.

In its most general formulation, the hydraulic-conductivity tensor $\mathbf{K} = \mathbf{K}(\mathbf{x})$ is a symmetric full tensor of rank three and has the form:

$$\mathbf{K} = \begin{bmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{xy} & K_{yy} & K_{yz} \\ K_{xz} & K_{yz} & K_{zz} \end{bmatrix}. \tag{2.2}$$

The partial differential equation governing steady-state groundwater flow in three dimensions can be rewritten as,

$$\begin{aligned} &\frac{\partial}{\partial x}\Big(K_{xx}\frac{\partial h}{\partial x} + K_{xy}\frac{\partial h}{\partial y} + K_{xz}\frac{\partial h}{\partial z}\Big) + \frac{\partial}{\partial y}\Big(K_{xy}\frac{\partial h}{\partial x} + K_{yy}\frac{\partial h}{\partial y} + K_{yz}\frac{\partial h}{\partial z}\Big) + \\ &\frac{\partial}{\partial z}\Big(K_{xz}\frac{\partial h}{\partial x} + K_{yz}\frac{\partial h}{\partial y} + K_{zz}\frac{\partial h}{\partial z}\Big) + q = 0. \end{aligned} \tag{2.3}$$

If this equation is discretized with a nineteen-point block-centered finite-difference scheme with nonuniform parallel-piped blocks, the following equation results(see Figure 2.1),

$$\begin{aligned} &\frac{1}{\Delta x|_{i,j,k}}\Bigg[\Big(K_{xx}\frac{\partial h}{\partial x} + K_{xy}\frac{\partial h}{\partial y} + K_{xz}\frac{\partial h}{\partial z}\Big)\Big|_{i+1/2,j,k} - \\ &\qquad \Big(K_{xx}\frac{\partial h}{\partial x} + K_{xy}\frac{\partial h}{\partial y} + K_{xz}\frac{\partial h}{\partial z}\Big)\Big|_{i-1/2,j,k}\Bigg] + \\ &\frac{1}{\Delta y|_{i,j,k}}\Bigg[\Big(K_{xy}\frac{\partial h}{\partial x} + K_{yy}\frac{\partial h}{\partial y} + K_{yz}\frac{\partial h}{\partial z}\Big)\Big|_{i,j+1/2,k} - \\ &\qquad (K_{xy}\frac{\partial h}{\partial x} + K_{yy}\frac{\partial h}{\partial y} + K_{yz}\frac{\partial h}{\partial z}\Big)\Big|_{i,j-1/2,k}\Bigg] + \\ &\frac{1}{\Delta z|_{i,j,k}}\Bigg[\Big(K_{xz}\frac{\partial h}{\partial x} + K_{yz}\frac{\partial h}{\partial y} + K_{zz}\frac{\partial h}{\partial z}\Big)\Big|_{i,j,k+1/2} - \\ &\qquad (K_{xz}\frac{\partial h}{\partial x} + K_{yz}\frac{\partial h}{\partial y} + K_{zz}\frac{\partial h}{\partial z}\Big)\Big|_{i,j,k-1/2}\Bigg] + \\ &\qquad\qquad q_{i,j,k} = 0. \end{aligned} \tag{2.4}$$

for a block (i, j, k) of size $\Delta x|_{i,j,k} \times \Delta y|_{i,j,k} \times \Delta z|_{i,j,k}$.

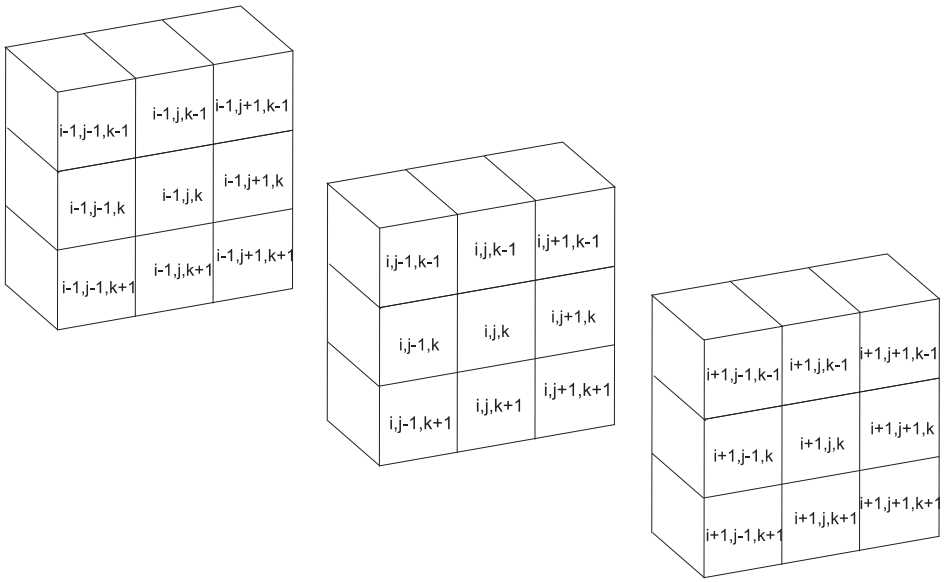

Figure 2.1. Schematic illustration of the three dimensional finite difference spatial discretization

The hydraulic gradients at the interfaces are approximated by central difference from the heads at the nineteen blocks surrounding (i, j, k), That is,

$$\begin{aligned}
\left.\frac{\partial h}{\partial x}\right|_{i+1/2,j,k} =& \frac{h_{i,j+1,k} - h_{i,j-1,k}}{\triangle x|_{i,j+1,k} + 2\triangle x|_{i,j,k} + \triangle x|_{i,j-1,k}} + \\
& \frac{h_{i+1,j+1,k} - h_{i+1,j-1,k}}{\triangle x|_{i+1,j+1,k} + 2\triangle x|_{i+1,j,k} + \triangle x|_{i+1,j-1,k}} \\
\left.\frac{\partial h}{\partial y}\right|_{i+1/2,j,k} =& \frac{2(h_{i+1,j,k} - h_{i,j,k})}{\triangle y|_{i+1,j,k} + \triangle y|_{i,j,k}} \\
\left.\frac{\partial h}{\partial z}\right|_{i+1/2,j,k} =& \frac{h_{i,j,k+1} - h_{i,j,k-1}}{\triangle z|_{i,j,k+1} + 2\triangle z|_{i,j,k} + \triangle z|_{i,j,k-1}} + \\
& \frac{h_{i+1,j,k+1} - h_{i+1,j,k-1}}{\triangle z|_{i+1,j,k+1} + 2\triangle z|_{i+1,j,k} + \triangle z|_{i+1,j,k-1}} \quad .
\end{aligned} \tag{2.5}$$

The partial derivatives of the hydraulic head in the other five interfaces can be given by a similar expression. Substitution equation (2.5) into (2.4), multiplying both sides by $\Delta x|_{i,j,k}\Delta y|_{i,j,k}\Delta z|_{i,j,k}$, and rearranging terms, the

following nineteen-point scheme results,

$$\begin{aligned}&Ah_{i,j+1,k} + Bh_{i,j,k} + Ch_{i+1,j+1,k} + Dh_{i-1,j+1,k} + Eh_{i+1,j,k} + Fh_{i-1,j,k} + \\ &Gh_{i,j+1,k+1} + Hh_{i,j+1,k-1} + Ih_{i,j,k+1} + Jh_{i,j,k-1} + Kh_{i,j-1,k} + Lh_{i+1,j-1,k} + \\ &Mh_{i-1,j-1,k} + Nh_{i,j-1,k+1} + Oh_{i,j-1,k-1} + Ph_{i+1,j,k+1} + Qh_{i+1,j,k-1} + \\ &Rh_{i-1,j,k+1} + Sh_{i-1,j,k-1} = -q_{i,j,k}\end{aligned} \tag{2.6}$$

where,

$$\begin{aligned}A =& \triangle y|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{2K_{xx}\big|_{i,j+1/2,k}}{\triangle x|_{i,j+1,k} + \triangle x|_{i,j,k}} \right] + \\ & \triangle x|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{K_{yx}\big|_{i+1/2,j,k} - K_{yx}\big|_{i-1/2,j,k}}{\triangle x|_{i,j+1,k} + 2\triangle x|_{i,j,k} + \triangle x|_{i,j-1,k}} \right] + \\ & \triangle x|_{i,j,k} \triangle y|_{i,j,k} \left[\frac{K_{zx}\big|_{i,j,k+1/2} - K_{zx}\big|_{i,j,k-1/2}}{\triangle x|_{i,j+1,k} + 2\triangle x|_{i,j,k} + \triangle x|_{i,j-1,k}} \right] \\ B =& \triangle y|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{-2K_{xx}\big|_{i,j+1/2,k}}{\triangle x|_{i,j+1,k} + \triangle x|_{i,j,k}} + \frac{-2K_{xx}\big|_{i,j-1/2,k}}{\triangle x|_{i,j,k} + \triangle x|_{i,j-1,k}} \right] + \\ & \triangle x|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{-2K_{yy}\big|_{i+1/2,j,k}}{\triangle y|_{i+1,j,k} + \triangle y|_{i,j,k}} + \frac{-2K_{yy}\big|_{i-1/2,j,k}}{\triangle y|_{i,j,k} + \triangle y|_{i-1,j,k}} \right] + \\ & \triangle x|_{i,j,k} \triangle y|_{i,j,k} \left[\frac{-2K_{zz}\big|_{i,j,k+1/2}}{\triangle z|_{i,j,k+1} + \triangle z|_{i,j,k}} + \frac{-2K_{zz}\big|_{i,j,k-1/2}}{\triangle z|_{i,j,k} + \triangle z|_{i,j,k-1}} \right] \\ C =& \triangle y|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{K_{xy}\big|_{i,j+1/2,k}}{\triangle y|_{i+1,j+1,k} + 2\triangle y|_{i-1,j+1,k} + \triangle y|_{i,j+1,k}} \right] + \\ & \triangle x|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{K_{yx}\big|_{i+1/2,j,k}}{\triangle x|_{i+1,j+1,k} + 2\triangle x|_{i+1,j-1,k} + \triangle x|_{i+1,j,k}} \right] \\ D =& \triangle y|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{-K_{xy}\big|_{i,j+1/2,k}}{\triangle y|_{i+1,j+1,k} + 2\triangle y|_{i-1,j+1,k} + \triangle y|_{i,j+1,k}} \right] + \\ & \triangle x|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{-K_{yx}\big|_{i-1/2,j,k}}{\triangle x|_{i-1,j+1,k} + 2\triangle x|_{i-1,j-1,k} + \triangle x|_{i-1,j,k}} \right]\end{aligned} \tag{2.7}$$

$$
\begin{aligned}
E =& \triangle y|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{K_{xy}\big|_{i,j+1/2,k} - K_{xy}\big|_{i,j-1/2,k}}{\triangle y|_{i+1,j,k} + 2\triangle y|_{i,j,k} + \triangle y|_{i-1,j,k}} \right] + \\
& \triangle x|_{i,j,k} \triangle z\_i, j, k \left[\frac{2K_{yy}\big|_{i+1/2,j,k}}{\triangle y|_{i+1,j,k} + \triangle y|_{i,j,k}} \right] + \\
& \triangle x|_{i,j,k} \triangle y|_{i,j,k} \left[\frac{K_{zy}\big|_{i,j,k+1/2} - K_{zy}\big|_{i,j,k-1/2}}{\triangle y|_{i+1,j,k} + 2\triangle y|_{i,j,k} + \triangle y_{i-1,j,k}} \right] \\
F =& \triangle y|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{K_{xy}\big|_{i,j-1/2,k} - K_{xy}\big|_{i,j+1/2,k}}{\triangle y|_{i+1,j,k} + 2\triangle y|_{i,j,k} + \triangle y|_{i-1,j,k}} \right] + \\
& \triangle x|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{2K_{yy}\big|_{i-1/2,j,k}}{\triangle y|_{i,j,k} + \triangle y|_{i-1,j,k}} \right] + \\
& \triangle x|_{i,j,k} \triangle y|_{i,j,k} \left[\frac{K_{zy}\big|_{i,j,k-1/2} - K_{zy}\big|_{i,j,k+1/2}}{\triangle y|_{i+1,j,k} + 2\triangle y|_{i,j,k} + \triangle y|_{i-1,j,k}} \right] \\
G =& \triangle y|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{K_{xz}\big|_{i,j+1/2,k}}{\triangle z|_{i,j+1,k+1} + 2\triangle z|_{i,j+1,k} + \triangle z|_{i,j+1,k-1}} \right] + \\
& \triangle x|_{i,j,k} \triangle y|_{i,j,k} \left[\frac{K_{zx}\big|_{i,j,k+1/2}}{\triangle x|_{i,j+1,k+1} + 2\triangle x|_{i,j,k+1} + \triangle x|_{i,j-1,k+1}} \right] \\
H =& \triangle y|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{-K_{xz}\big|_{i,j+1/2,k}}{\triangle z|_{i,j+1,k+1} + 2\triangle z|_{i,j+1,k} + \triangle z|_{i,j+1,k-1}} \right] + \\
& \triangle x|_{i,j,k} \triangle y|_{i,j,k} \left[\frac{-K_{zx}\big|_{i,j,k-1/2}}{\triangle x|_{i,j+1,k-1} + 2\triangle x|_{i,j,k-1} + \triangle x|_{i,j-1,k-1}} \right] \\
I =& \triangle y|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{K_{xz}\big|_{i,j+1/2,k} - K_{xz}\big|_{i,j-1/2,k}}{\triangle z|_{i,j,k+1} + 2\triangle z|_{i,j,k} + \triangle z|_{i,j,k-1}} \right] + \\
& \triangle x|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{K_{yz}\big|_{i+1/2,j,k} - K_{yz}\big|_{i-1/2,j,k}}{\triangle z|_{i,j,k+1} + 2\triangle z|_{i,j,k} + \triangle z|_{i,j,k-1}} \right] + \\
& \triangle x|_{i,j,k} \triangle y|_{i,j,k} \left[\frac{2K_{zz}\big|_{i,j,k+1/2}}{\triangle z|_{i,j,k+1} + \triangle z|_{i,j,k}} \right]
\end{aligned}
\tag{2.8}
$$

$$
\begin{aligned}
J =& \triangle y|_{i,j,k}\triangle z|_{i,j,k}\left[\frac{K_{xz}\big|_{i,j-1/2,k} - K_{xz}\big|_{i,j+1/2,k}}{\triangle z|_{i,j,k+1} + 2\triangle z|_{i,j,k} + \triangle z|_{i,j,k-1}}\right] + \\
& \triangle x|_{i,j,k}\triangle z|_{i,j,k}\left[\frac{K_{yz}\big|_{i-1/2,j,k} - K_{yz}\big|_{i+1/2,j,k}}{\triangle z|_{i,j,k+1} + 2\triangle z|_{i,j,k} + \triangle z|_{i,j,k-1}}\right] + \\
& \triangle x|_{i,j,k}\triangle y|_{i,j,k}\left[\frac{2K_{zz}\big|_{i,j,k-1/2}}{\triangle z|_{i,j,k-1} + \triangle z|_{i,j,k}}\right] \\
K =& \triangle y|_{i,j,k}\triangle z|_{i,j,k}\left[\frac{2K_{xx}\big|_{i,j-1/2,k}}{\triangle x|_{i,j,k} + \triangle x|_{i,j-1,k}}\right] + \\
& \triangle x|_{i,j,k}\triangle z|_{i,j,k}\left[\frac{(K_{yx}\big|_{i-1/2,j,k} - K_{yx}\big|_{i+1/2,j,k})}{\triangle x|_{i,j+1,k} + 2\triangle x|_{i,j,k} + \triangle x|_{i,j-1,k}}\right] + \\
& \triangle x|_{i,j,k}\triangle y|_{i,j,k}\left[\frac{(K_{zx}\big|_{i-1/2,j,k} - K_{zx}\big|_{i+1/2,j,k})}{\triangle x|_{i,j+1,k} + 2\triangle x|_{i,j,k} + \triangle x|_{i,j-1,k}}\right] \\
L =& \triangle y|_{i,j,k}\triangle z|_{i,j,k}\left[\frac{-K_{xy}\big|_{i,j-1/2,k}}{\triangle y|_{i+1,j-1,k} + 2\triangle y|_{i,j-1,k} + \triangle y|_{i-1,j-1,k}}\right] + \\
& \triangle x|_{i,j,k}\triangle z|_{i,j,k}\left[\frac{-K_{yx}\big|_{i+1/2,j,k}}{\triangle x|_{i+1,j+1,k} + 2\triangle x|_{i+1,j-1,k} + \triangle x|_{i+1,j,k}}\right] \\
M =& \triangle y|_{i,j,k}\triangle z|_{i,j,k}\left[\frac{K_{xy}\big|_{i,j-1/2,k}}{\triangle y|_{i+1,j-1,k} + 2\triangle y|_{i,j-1,k} + \triangle y|_{i-1,j-1,k}}\right] + \\
& \triangle x|_{i,j,k}\triangle z|_{i,j,k}\left[\frac{K_{yx}\big|_{i-1/2,j,k}}{\triangle x|_{i-1,j+1,k} + 2\triangle x|_{i-1,j-1,k} + \triangle x|_{i-1,j,k}}\right] \\
N =& \triangle y|_{i,j,k}\triangle z|_{i,j,k}\left[\frac{-K_{xz}\big|_{i,j-1/2,k}}{\triangle z|_{i,j-1,k+1} + 2\triangle z|_{i,j-1,k} + \triangle z|_{i,j-1,k-1}}\right] + \\
& \triangle x|_{i,j,k}\triangle y|_{i,j,k}\left[\frac{-K_{zx}\big|_{i,j,k+1/2}}{\triangle x|_{i,j+1,k+1} + 2\triangle x|_{i,j,k+1} + \triangle x|_{i,j-1,k+1}}\right] \\
O =& \triangle y|_{i,j,k}\triangle z|_{i,j,k}\left[\frac{K_{xz}\big|_{i,j-1/2,k}}{\triangle z|_{i,j-1,k+1} + 2\triangle z|_{i,j-1,k} + \triangle z|_{i,j-1,k-1}}\right] + \\
& \triangle x|_{i,j,k}\triangle y|_{i,j,k}\left[\frac{K_{zx}\big|_{i,j,k-1/2}}{\triangle x|_{i,j+1,k-1} + 2\triangle x|_{i,j,k-1} + \triangle x|_{i,j-1,k-1}}\right] \\
P =& \triangle y|_{i,j,k}\triangle z|_{i,j,k}\left[\frac{K_{yz}\big|_{i+1/2,j,k}}{\triangle z|_{i+1,j,k+1} + 2\triangle z|_{i+1,j,k} + \triangle z|_{i+1,j,k-1}}\right] + \\
& \triangle x|_{i,j,k}\triangle y|_{i,j,k}\left[\frac{K_{zy}\big|_{i,j,k+1/2}}{\triangle y|_{i+1,j,k+1} + 2\triangle y|_{i,j,k+1} + \triangle y|_{i-1,j,k+1}}\right]
\end{aligned}
\tag{2.9}
$$

$$
\begin{aligned}
Q =& \triangle y|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{-K_{yz}\big|_{i+1/2,j,k}}{\triangle z|_{i+1,j,k+1} + 2\triangle z|_{i+1,j,k} + \triangle z|_{i+1,j,k-1}} \right] + \\
& \triangle x|_{i,j,k} \triangle y|_{i,j,k} \left[\frac{-K_{zy}\big|_{i,j,k-1/2}}{\triangle y|_{i+1,j,k-1} + 2\triangle y|_{i,j,k-1} + \triangle y|_{i-1,j,k-1}} \right] \\
R =& \triangle y|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{-K_{yz}\big|_{i-1/2,j,k}}{\triangle z|_{i-1,j,k+1} + 2\triangle z|_{i-1,j,k} + \triangle z|_{i-1,j,k-1}} \right] + \\
& \triangle x|_{i,j,k} \triangle y|_{i,j,k} \left[\frac{-K_{zy}\big|_{i,j,k+1/2}}{\triangle y|_{i+1,j,k+1} + 2\triangle y|_{i,j,k+1} + \triangle y|_{i-1,j,k+1}} \right] \\
S =& \triangle y|_{i,j,k} \triangle z|_{i,j,k} \left[\frac{K_{yz}\big|_{i-1/2,j,k}}{\triangle z|_{i-1,j,k+1} + 2\triangle z|_{i-1,j,k} + \triangle z|_{i-1,j,k-1}} \right] + \\
& \triangle x|_{i,j,k} \triangle y|_{i,j,k} \left[\frac{K_{zy}\big|_{i,j,k-1/2}}{\triangle y|_{i+1,j,k-1} + 2\triangle y|_{i,j,k-1} + \triangle y|_{i-1,j,k-1}} \right].
\end{aligned}
\tag{2.10}
$$

Two types of boundary conditions are taken into consideration in the process of building the system of linear equations: (1) Dirichlet condition that is utilized to model the interaction with the outside of the model domain. It is modeled by prescribing the head on the boundary, (2) Neumann condition that is used to model fixed source terms, in which the groundwater flux is specified (e.g., wells). Furthermore, a no-flow boundary, a special case of Neumann condition, is also considered.

The implementation of the two types of boundary conditions is as follows: (i) for prescribed heads we take advantage that we use an iterative approach for the solution of the linear system of equations, the prescribed head values at the corresponding blocks are set to their values at the beginning of the iterations and never updated during successive iterations, (ii) for prescribed fluxes through the boundaries, their values are assimilated inside the volumetric source vector q, whereas to handle no flow boundary conditions we always surround the entire domain with a skin of one block of inactive cells, in this way, equation (2.6) can be applied for all the active blocks without having to consider especial cases for the blocks at the edges of the aquifer; then, for all inactive cells, the interface conductivity with adjacent blocks is set to zero, and, in addition, in order to compute the piezometric head gradient parallel to the interfaces a fictitious head value is assigned to these inactive cells equal to the average of the heads in the neighboring active blocks.

Also, when a well straddles several cells, the interface conductivities "inside" the well are set to a very large value and the pumping rate assigned to the bottommost cell.

With the above considerations for the boundary conditions, equation (2.6) is written for all active blocks within the aquifer, resulting in a system of linear equations that is solved using an iterative method based on a successive over relaxation (SOR) scheme.

Before the benchmarking of the proposed code and the LDVA package of MODFLOW (Harbaugh et al., 2000), it was verified in several synthetic exercises with scalar conductivities and various scenarios against MODFLOW, with results in perfect agreement.

2.3 Program Description

The program takes as input values the hydraulic conductivity tensors at the block interfaces, these values can be predetermined by the user. In our experiments we assume that they have been obtained by upscaling a fine scale conductivity field using the Laplacian with skin method (Gómez-Hernández, 1991) by means of the code described in the accompanying paper by Zhou et al. (2010).

The verification of the code will be achieved by comparing the specific discharges at the interfaces with those resulting from the solution of the groundwater flow equation at the fine scale. Short of analytical expressions against which to compare our code results, the use of a fine scale simulation is a good alternative. A flowchart of the code can be seen in Figure 2.2.

The different steps are the following:

1. Read initial interblock hydraulic conductivity tensors. These data can be derived by upscaling a fine scale simulation (Wen and Gómez-Hernández, 1996), generated using geostatistical methods (Deutsch and Journel, 1992), or simply by processing (averaging) the hydraulic conductivities at block centers, if they are available.

2. Read boundary condition data. As is the case for other forward simulators (e.g., MODFLOW), a boundary condition flag is stored in variable *ibound* with the following meaning:

$$\begin{aligned} &ibound(k,i,j) < 0; \quad \text{block } (k,i,j) \text{ has a prescribed head,} \\ &ibound(k,i,j) = 0; \quad \text{block } (k,i,j) \text{ is inactive,} \\ &ibound(k,i,j) > 0; \quad \text{block } (k,i,j) \text{ is active.} \end{aligned} \tag{2.11}$$

3. Read the initial guess heads to start the iterations. These initial heads must be equal to the prescribed head values at the blocks with prescribed head boundary conditions.

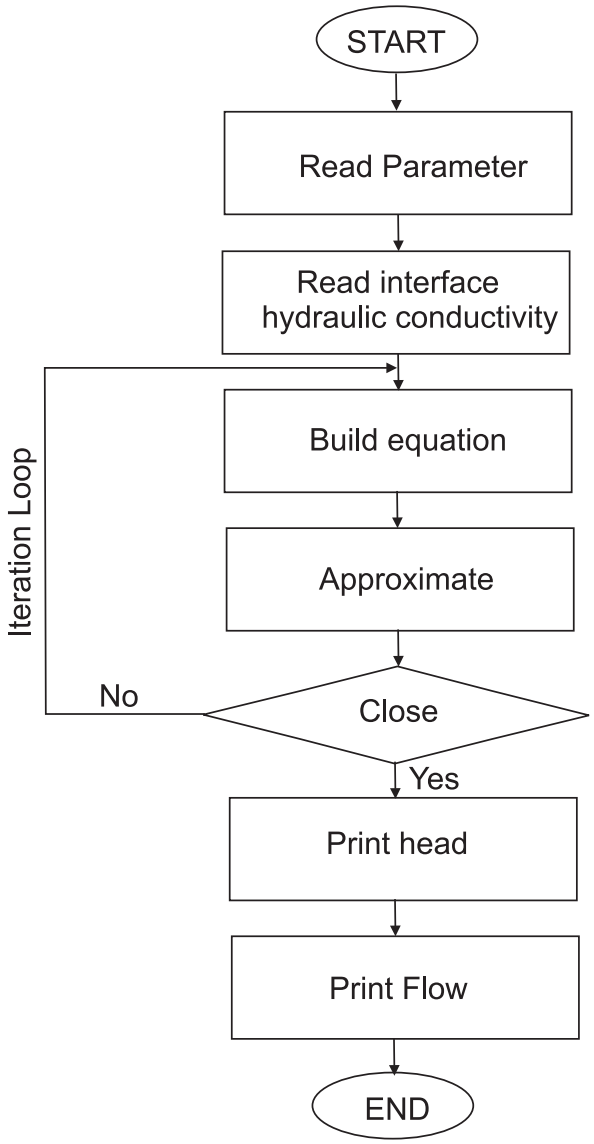

Figure 2.2. Flowchart of proposed simulator

4. Set the interface conductivities to zero at impermeable boundaries and compute the fictitious head values at inactive cells (recall that these values are needed to compute the gradients parallel to the block interface)

5. Configure the linear system of equations by determining the nineteen coefficients in equation (2.6) for each active cell.

6. Solve the flow equation with implicit technique (SOR). On each iteration, the heads at all active nodes are updated. Then, an intermediate value $\hat{h}_{k,i,j}$ is computed using the most recently updated values.

$$\begin{aligned}\hat{h}_{k,i,j} = \frac{1}{B}(&- Ah_{i,j+1,k} - Ch_{i+1,j+1,k} - Dh_{i-1,j+1,k} + Eh_{i+1,j,k} - Fh_{i-1,j,k} \\ &- Gh_{i,j+1,k+1} - Hh_{i,j+1,k-1} - Ih_{i,j,k+1} - Jh_{i,j,k-1} - Kh_{i,j-1,k} \\ &- Lh_{i+1,j-1,k} - Mh_{i-1,j-1,k} - Nh_{i,j-1,k+1} - Oh_{i,j-1,k-1} \\ &- Ph_{i+1,j,k+1} - Qh_{i+1,j,k-1} - Rh_{i-1,j,k+1} - Sh_{i-1,j,k-1} - q_{i,j,k}).\end{aligned} \tag{2.12}$$

   The updated head value for iteration $m+1$ is finally given by,

$$h_{k,i,j}^{m+1} = h_{k,i,j}^{m} + \omega(\hat{h}_{k,i,j} - h_{k,i,j}^{m}) \tag{2.13}$$

   where ω is the relaxation coefficient. When $\omega = 1$ the iterative scheme is the standard Gauss-Seidel algorithm. Iterations are stopped when the maximum absolute head change over all aquifer nodes is smaller than the predefined closure value.

   The above approach is not applied to prescribed head blocks, in which their value is not modified through the iterations. Once an iteration is completed, the fictitious heads at inactive cells are also updated by assigning to them the average value of the neighboring active blocks.

7. Once the head solution converges, Darcy's law is used to determine specific discharges $\mathbf{q}$ at block interfaces by,

$$\mathbf{q} = -\mathbf{K}\nabla h \tag{2.14}$$

2.4 Synthetic Examples

This section demonstrates the algorithm presented in the previous section for two 2D heterogeneous realizations with uniform discretization, and a 3D example with nonuniform blocks.

All examples deal with heterogeneous fields, for which no analytical solution is available. For this reason, and in order to verify the accuracy of the

solution, we always consider that the coarse model is the result of the upscaling of a fine scale model with a higher resolution characterized by scalar conductivities. On this fine-scale model, standard finite-difference numerical simulators can be used to compute heads and specific discharges that can serve as the yardstick with which to measure the accuracy of the coarse scale simulations.

2.4.1 Two dimensional examples

The hydraulic conductivity tensors at the interfaces are derived after upscaling two different fine scale fields that have been generated with different random function models. The fields at the fine scale are discretized with a mesh of 10 m by 10 m and extend on a square of 3000 m by 3000 m. We will refer to them as models A, B. The $\ln K$ field associated with model A was generated by Sequential Gaussian Simulation (GCOSIM3D) (Gómez-Hernández and Journel, 1993) with mean zero and unit variance, displaying a strong anisotropic spatial correlation in the 45° direction (see Figure 2.3). The correlation length in the largest continuity direction (x') is $\lambda_{x'} = 150$ m and in the smallest continuity direction (y') is $\lambda_{y'} = 30$ m. The variogram is exponential:

$$\gamma(r) = 1.0 \cdot \left\{ 1 - \exp\left[- \sqrt{\left(\frac{3r_{x'}}{150}\right)^2 + \left(\frac{3r_{y'}}{30}\right)^2} \right] \right\} \tag{2.15}$$

with

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.16}$$

The fine scale hydraulic conductivity field associated with model B represents a sand/shale aquifer. The sand logconductivity values are generated by GCOSIM3D with zero mean and an isotropic exponential variogram function given by $\gamma(r) = 1 - \exp(-3r/\lambda)$, where the correlation length λ is set to 30 m. All shales are assigned a constant log-conductivity of -9. The distribution of sand and shales is generated by Sequential Indicator Simulation (ISIM3D) (Gómez-Hernández and Srivastava, 1990) using an indicator random function. The binary random function is defined as,

$$I(x) = \begin{cases} 1, & \text{if } x \text{ in shale} \\ 0, & \text{if } x \text{ in sand} \end{cases} \tag{2.17}$$

with a mean value of $p = 0.15$, (15% of the aquifer is shale); variance of $p(1-p) = 0.1275$, and an exponential covariance function with correlation length in x direction $\lambda_x = 8$ m and negligible correlation in the y direction (Figure 2.4).

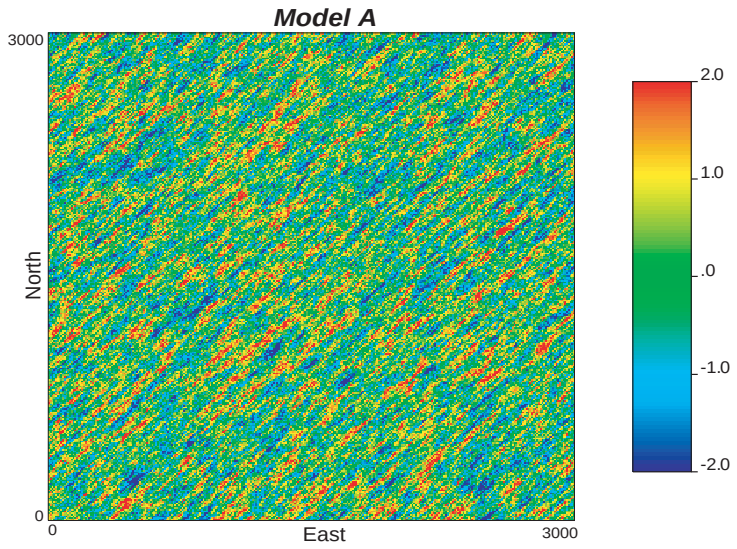

Figure 2.3. The fine scale hydraulic conductivity field used in model A. It was generated using Sequential Gaussian Simulation

$$\gamma_I(r) = 0.1275\big[1 - \exp(-3|r_x|/\lambda_x)\big] \tag{2.18}$$

Figure 2.5 shows the standardized variograms for both realizations together with the theoretical variograms.

Both hydraulic conductivities fields are upscaled onto a coarser model. The coarse model extends over the inner 2400 m by 2400 m of the fine scale simulations with a discretization of 100 m by 100 m. The need to use a skin in the upscaling process Gómez-Hernández (1991) requires that the fine scale model extends beyond the limits of the coarse model. For the details on the upscaling process and how the interblock conductivities are computed, the reader is referred to the work by Zhou et al. (2010). For both model A and B, the skin used in the upscaling process is half the block size, i.e., 50 m, or 5 fine scale cells.

2.4.2 Flow Simulation

A forced-gradient flow problem is posed for both models over the 2400 m by 2400 m inner area of the fine scale realizations. See Figure 2.6. Prescribed heads are assigned on the perimeter of the model so that the average head gradient is parallel to the 45° direction. No sinks or sources are included.

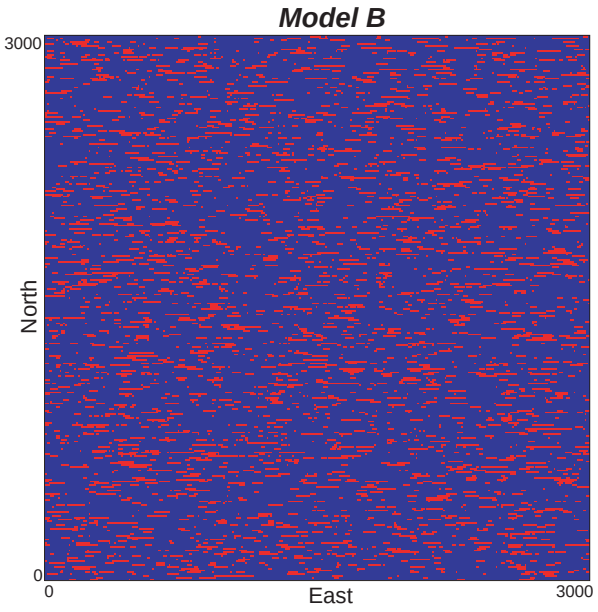

Figure 2.4. The fine scale hydraulic conductivity field used in model B. It was generated using Sequential Indicator Simulation (shales in red with $\ln K = -9$, sands in purple with mean $\ln K = 0$ and variance $\ln K = 1$)

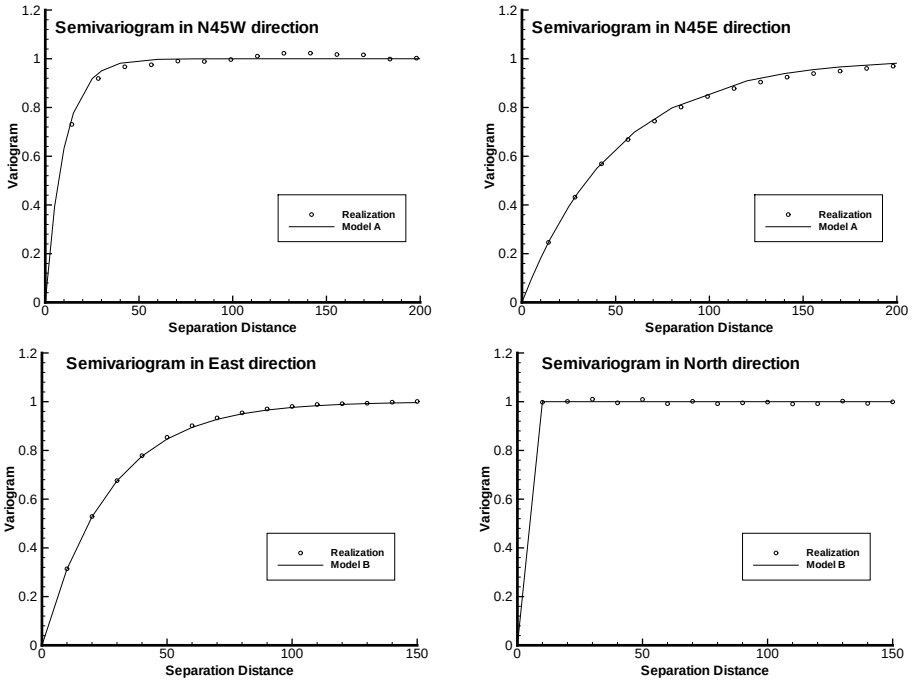

Figure 2.5. Model variograms and experimental variograms as inferred from the realizations for their respective directions of maximum and minimum continuity

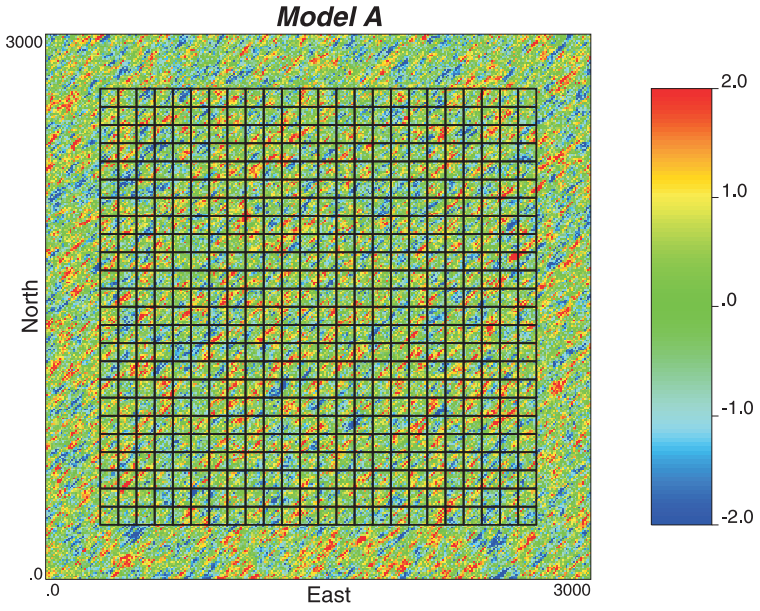

Figure 2.6. Model A fine scale hydraulic conductivity model overlaid with the discretization of the numerical model at the coarse scale

Flow is solved on the fine scale model over the 2400 m by 2400 m area, and the fluxes crossing the interfaces at the coarse scale discretization are computed. We will compare them to the flows from the coarse scale model.

We compare the effectiveness of our approach with that of MODFLOW. For this purpose we focus on the reproduction of the fluxes at the block interfaces. We compare the coarse scale flows, that is, those obtained after solving the flow equation with the upscaled conductivities, with the reference flows, that is, those obtained from the solution of the flow equation at the fine scale. The mismatch between these two values is measured with the Relative Bias defined as,

$$RB = \left(\frac{1}{N} \sum_{N} \frac{|q_f - q_c|}{q_f} \right) \cdot 100, \tag{2.19}$$

where N is the number of blocks/interblocks used to compute the relative bias; q_f are the specific discharges computed on the fine scale solution, and q_c are the specific discharges from the coarse scale simulation.

Notice that we have chosen to analyze the goodness of the models by comparing the reproduction of the specific discharges, since these are much more sensitive than the hydraulic heads. Indeed, Figure 2.7 shows the hydraulic heads from the fine scale solution, and from the coarse scale solutions, calcu-

lated with the MODFLOW LVDA module and with the proposed simulator in model A. The differences in hydraulic heads are negligible; however, the discrepancy on the specific discharges is larger as will be shown later (see Figure 2.8). Recall that proper reproduction of the specific discharges is important in solute transport predictions.

In order to avoid the problems associated with the upscaling of the boundary conditions described by Vermeulen et al. (2006), only the inner 20 by 20 blocks are used to compute the relative bias. Figures 2.8 and 2.9 show the cross-plots between the specific discharges computed on the fine scale (reference values) and the ones computed on the coarse scale using the LVDA package from MOFLOW and the proposed approach. In the MODFLOW case, the upscaling code by Zhou et al. (2010) is used to compute the block conductivity tensors (instead of the tensors at the interfaces) since these are the parameters needed by the package. The LVDA package uses these block tensor values to approximate the conductivity tensors at the interfaces. It is clear that our numerical model outperforms MODFLOW LVDA package in both cases: for model A with the spatially anisotropic realization and for model B with the realization of sands and shales. The main reason for these results is the direct use of the interblock conductivities in our code, thus avoiding any averaging of block values to get to the interblock tensors.

2.4.3 Application in Three Dimensions

In this subsection, we present an application of the code to a three dimensional example with a spatial anisotropy which is not parallel to any of the refernence axes. This case cannot be handled by the MODFLOW LVDA module, which only considers anisotropy within each layer. At the same time, we will check the code on a nonuniform discretization.

As in the 2D examples, we start with a fine scale realization generated over a grid of $120 \times 170 \times 70$ cells, each cell being a cube of 1 m by 1 m by 1 m (see Figure 2.10). The realization is generated by GCOSIM3D with a $\ln K$ distribution $\sim N(0, 1)$ and an isotropic exponential variogram with a correlation length of 20 m. The fine scale realization is upscaled onto a coarse model with 13 by 18 by 7 blocks of variable dimensions as shown in Figure 2.11 using the upscaling code by Zhou et al. (2010).

Note that the fact that the spatial variability of hydraulic conductivity is isotropic does not imply that the interblock conductivity tensors are diagonal, thus the needed of using full tensor hydraulic conductivities at the interfaces.

The aquifer is confined and subject to linearly varying prescribed heads in the outside boundary that induce an overall head gradient oriented from the upper left corner in the top layer to the lower right corner in the bottom layer (Figure 2.11).

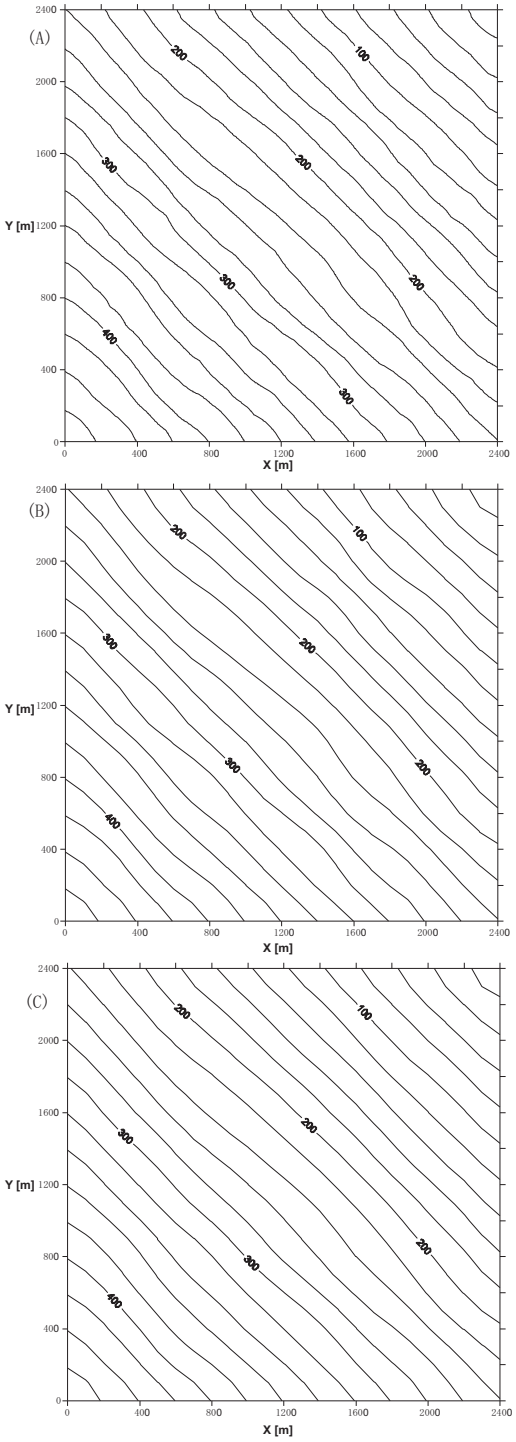

Figure 2.7. Contour lines of piezometric head for model A. (A) Contour lines obtained after within block averaging of the fine scale simulated heads. (B) Contour lines of the coarse heads obtained with the proposed simulator. (C), Contour lines of the coarse head obtained with MODFLOW using the LVDA package

.

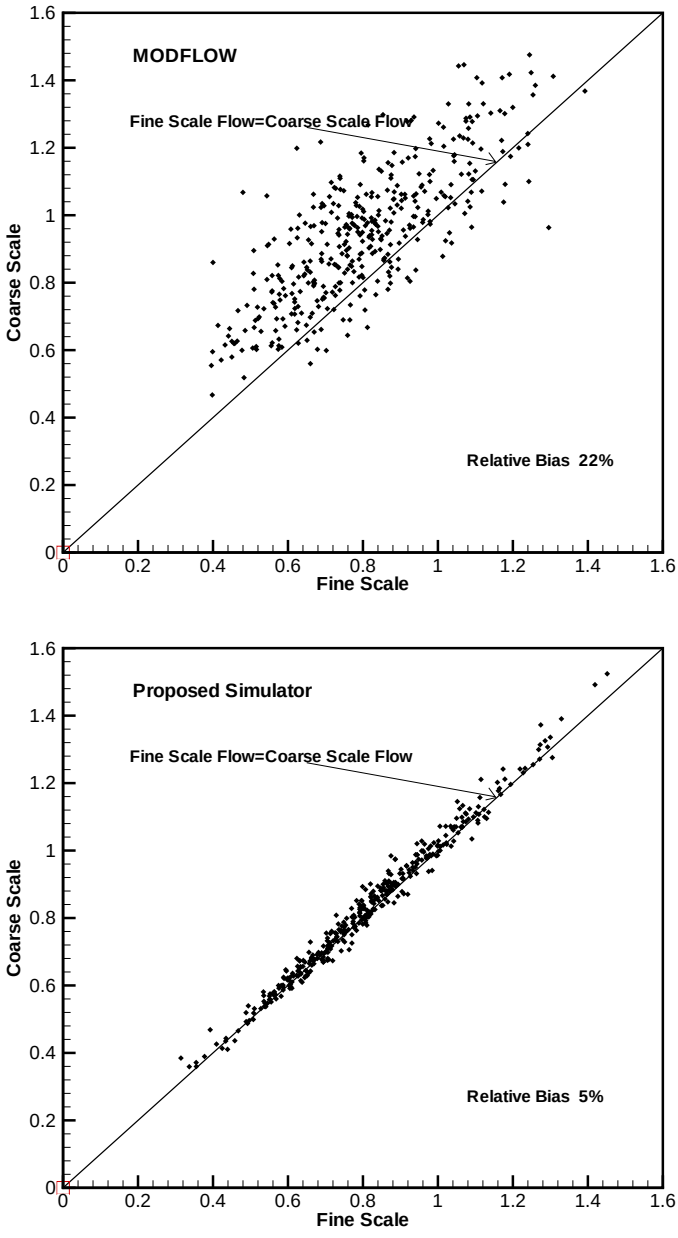

Figure 2.8. Flow comparisons in proposed simulator and MODFLOW using Model A.

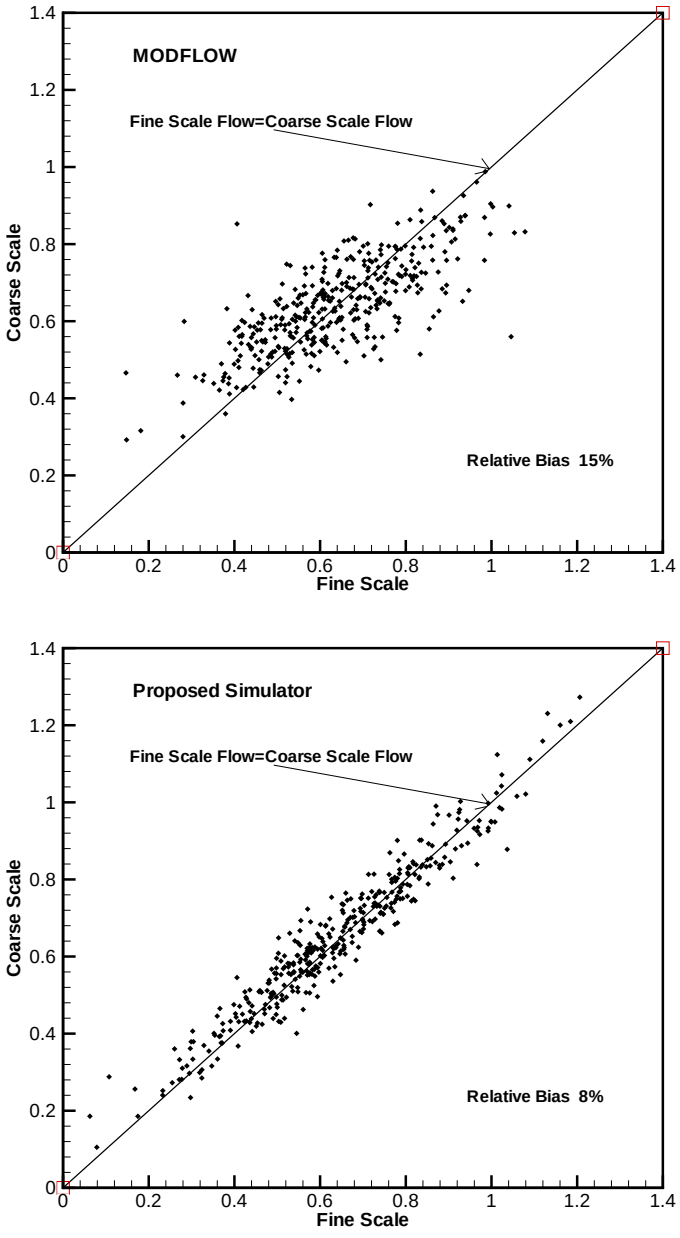

Figure 2.9. Flow comparisons in proposed simulator and MODFLOW using Model B.

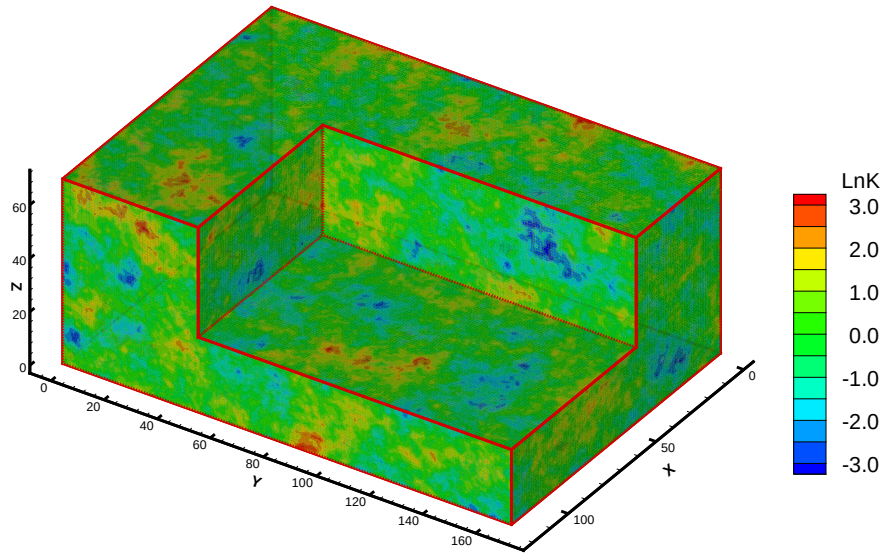

Figure 2.10. Reference ln$K$ field in three dimensions

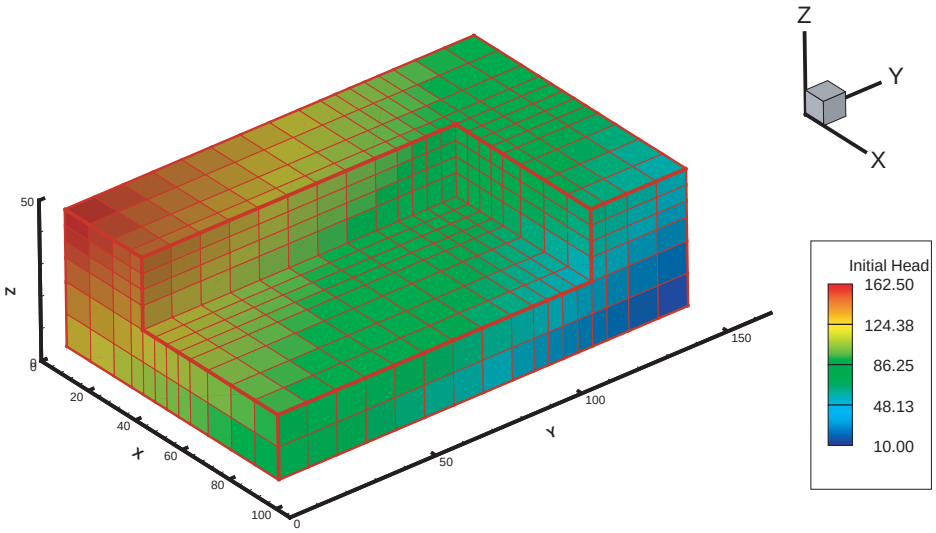

Figure 2.11. Nonuniform block discretization in three dimensions displaying the initial guess head values to start the iterations along with the prescribed heads at the boundaries

Figures 2.12 to 2.14 show the cross-plots for the specific discharges across the block faces in the coarse model computed on both the fine scale and the coarse scale models. We have discriminated the fluxes according to the face orientation. We can see that the relative bias is small in all three orientations and that the reproduction of the specific discharges on the block faces of the coarse model approximate very well their value as derived from the detailed fine scale simulation. We can conclude that the proposed flow simulator is robust and accurate for the simulation of steady-state groundwater flow in three dimensions using full tensor hydraulic conductivities at the interblock

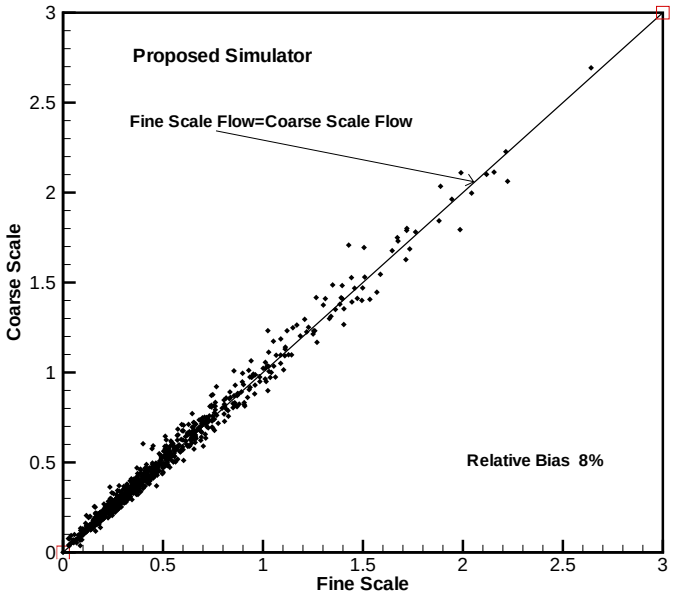

Figure 2.12. Flow comparison in X direction

2.5 Conclusion

Modeling groundwater flow in anisotropic highly heterogeneous media calls for models capable to handle a tensorial description of hydraulic conductivity without the need of forcing all blocks to have the same principal directions, or that the principal directions be aligned in a specific orientation. Careful modeling of the hydraulic conductivities will yield accurate computation of aquifer fluxes, which are especially important for solute transport predictions. We have presented an algorithm and code that is capable to handle this generic description of the hydraulic conductivity tensor field and we have demonstrated

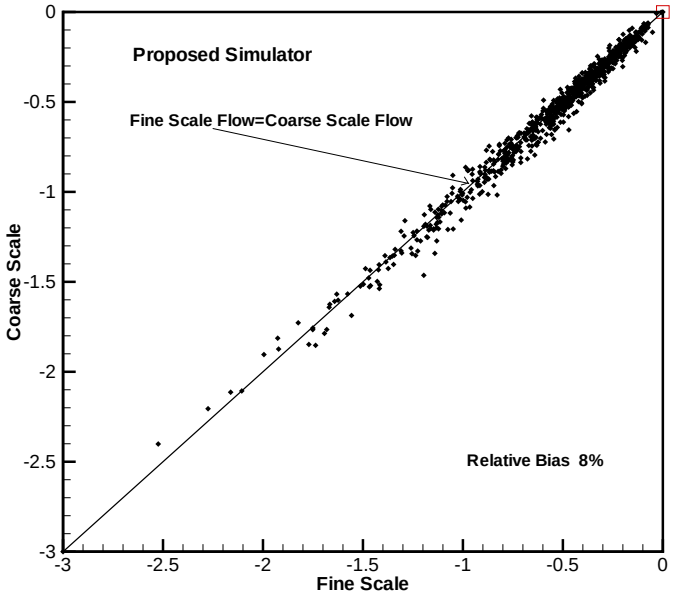

Figure 2.13. Flow comparison in Y direction

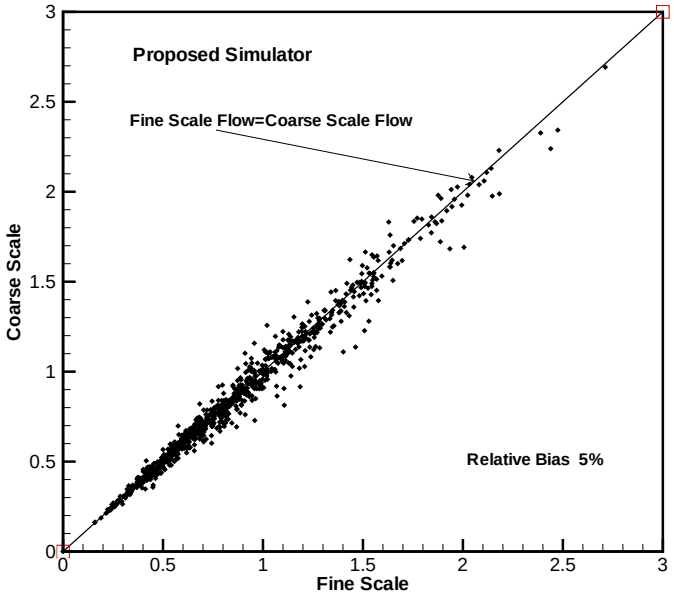

Figure 2.14. Flow comparison in Z direction

its accuracy both in 2D and 3D, and also using uniform and nonuniform discretizations. The main characteristic of this code is that it works directly with the interblock tensor conductivities needed in the finite-difference formulation of the groundwater flow equation, thus avoiding unnecessary averaging of conductivities from neighboring blocks. This feature is the responsible for the better performance of our algorithm than the LVDA package in MODFLOW.

Although the examples presented are associated with the solution of an upscaling problem, the code is generic and could be used in any other context. The next step in the development of the code will be to incorporate transient capabilities.

Bibliography

Aavatsmark, I., Barkve, T., øBøe, Mannseth, T., 1996. Discretization on non-orthogonal, quadrilateral grids for inhomogeneous, anisotropic media. Journal of computational physics 127 (1), 2–14.

Anderman, E. R., Kipp, K. L., Hill, M. C., Valstar, J., Neupauer, R. M., 2002. MODFLOW-2000, the US geological survey modular Ground-Water Model Documentation of the Model-Layer Variable-Direction horizontal anisotropy (LVDA) capability of the Hydrogeologic-Unit flow (HUF) package. US Geological Survey, Open file Report, 02–409.

Bear, J., 1972. Dynamics of fluids in porous media. American Elsevier Pub. Co., New York.

Bierkens, M. F. P., Weerts, H. J. T., 1994. Block hydraulic conductivity of cross-bedded fluvial sediments. Water Resources Research 30 (10), 2665–2678.

Deutsch, C. V., Journel, A. G., 1992. GSLIB, Geostatistical Software Library and User's Guide. Oxford University Press, New York.

Duvaut, G., Lions, J. L., John, C. W., 1976. Inequalities in mechanics and physics. Springer Verlag.

Edwards, M. G., Rogers, C. F., 1998. Finite volume discretization with imposed flux continuity for the general tensor pressure equation. Computational Geosciences 2 (4), 259–290.

Gómez-Hernández, J. J., 1991. A stochastic approach to the simulation of block conductivity values conditioned upon data measured at a smaller scale. Ph.D. thesis, Stanford University.

Gómez-Hernández, J. J., Journel, A. G., 1993. Joint sequential simulation of multi-Gaussian fields. Geostatistics Troia 92 (1), 85–94.

Gómez-Hernández, J. J., Srivastava, R. M., 1990. ISIM3D: an ANSI-C three dimensional multiple indicator conditional simulation program. Computer and Geosciences 16 (4), 395–440.

Harbaugh, A. W., Banta, E. R., Hill, M. C., McDonald, M. G., 2000. MODFLOW-2000, the U.S. Geological Survey modular ground-water model. U.S. Geological Survey, Branch of Information Services, Reston, VA, Denver, CO.

Penman, J., 1988. Dual and complementary variational techniques for the calculation of electromagnetic fields. Advances in electronics and electron physics. 70, 315–364.

Rozon, B., 1989. A generalized finite volume discretization method for reservoir simulation. In: SPE Symposium on Reservoir Simulation.

Vermeulen, P. T. M., Stroet, C. B. M. T., Heemink, A. W., 2006. Limitations to upscaling of groundwater flow models dominated by surface water interaction. Water Resources Research 42 (10), W10406.

Wen, X., Gómez-Hernández, J. J., 1996. Upscaling hydraulic conductivities: An overview. J. of Hydrology 183 (1-2), ix–xxxii.

Zhou, H. Y., Gómez-Hernández, J. J., Li, L. P., 2010. Three-dimensional hydraulic conductivity upscaling in groundwater modelling. Computer and Geosciences submitted.

A
Finite Difference Code

```
//
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//                                                                             %
// Copyright (C) 2009, Liangping Li,J. J. Gomez-Hernandez, and Haiyan          %
// Zhou.  Universidad Politecnica de Valencia,All rights reserved.             %
//                                                                             %
// This program is distributed in the hope that they will be useful,           %
// but WITHOUT ANY WARRANTY.  No author or distributor accepts                 %
// responsibility to anyone for the consequences of using them or for          %
// whether they serve any particular purpose or work at all, unless he         %
// says so in writing.  Everyone is granted permission to copy, modify         %
// and redistribute this program, but only under the condition                 %
// that this notice and the above copyright notice remain intact.              %
//                                                                             %
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//-------------------------------------------------------------------------


//
//   Steady state groundwater flow modleing with full tensor
//                    by finite-difference
//          ***************************************
//
//
// The program is executed with no command line arguments.  The user
// will be prompted for the name of a parameter file.  The parameter
// file is described in the documentation (see the example flowxyz.par)
//



#include<iostream>
#include<fstream>
#pragma warning(disable:4786)
#include<vector>
#include<cmath>
#include<string>
using namespace std;

void Read_para(int &ncol,int &nrow,int &nlay,float &f_dx,float &f_dy,float &f_dz,
         int &max_iter,float &accl,float &closure,string &kxx_file,string &kyy_file,
         string &kzz_file,string &ihead_file,string &ibound_file,string &head_file,
         string &qx_file,string &qy_file,string &qz_file,int &n_well,string &well_file,
         string &dx_file,string &dy_file,string &dz_file,int &flag_dx,int &flag_dy,
         int &flag_dz,int &debug);
```

```
void Read_data(int ncol,int nrow,int nlay,string &kxx_file,string &kyy_file,
        string &kzz_file,string &ihead_file,string &ibound_file,
        vector< vector< vector<float> > > &dx_3d,vector< vector< vector<float> > > &dy_3d,
        vector< vector< vector<float> > > &dz_3d,vector< vector< vector<float> > > &head,
        vector< vector< vector<int> > > &ibo,vector< vector< vector<float> > > &cc_xx,
        vector< vector< vector<float> > > &cc_xy,vector< vector< vector<float> > > &cc_xz,
        vector< vector< vector<float> > > &cr_yx,vector< vector< vector<float> > > &cr_yy,
        vector< vector< vector<float> > > &cr_yz,vector< vector< vector<float> > > &cv_zx,
        vector< vector< vector<float> > > &cv_zy,vector< vector< vector<float> > > &cv_zz,
        string &well_file,int n_well,vector<float>&well_x,vector<float>&well_y,
        vector<float>&well_z,vector<float>&well_q, string &dx_file,string &dy_file,
        string &dz_file,float f_dx,float f_dy,float f_dz,int flag_dx,int flag_dy,
        int flag_dz,int debug);

void Set_boundary(int ncol,int nrow,int nlay,string &kxx_file,string &kyy_file,
        string &kzz_file,vector< vector< vector<int> > > &ibo,
        vector< vector< vector<float> > > &cc_xx,vector< vector< vector<float> > > &cc_xy,
        vector< vector< vector<float> > > &cc_xz,vector< vector< vector<float> > > &cr_yx,
        vector< vector< vector<float> > > &cr_yy,vector< vector< vector<float> > > &cr_yz,
        vector< vector< vector<float> > > &cv_zx,vector< vector< vector<float> > > &cv_zy,
        vector< vector< vector<float> > > &cv_zz,int debug);

void write_debug1(int nlay,int nrow,int ncol,string &ibound_file,
        vector< vector< vector<int> > > &ibo);

void write_debug2(int nlay,int nrow,int ncol,string &ihead_file,
        vector< vector< vector<float> > > &head);

void write_debug3(int nlay,int nrow,int ncol,string &kxx_file,
        vector< vector< vector<float> > > &cc_xy);

void Compute_abcs(int nlay,int nrow,int ncol,vector< vector< vector<float> > >dx,
        vector< vector< vector<float> > >dy,
        vector< vector< vector<float> > >dz,vector< vector< vector<float> > > &cc_xx,
        vector< vector< vector<float> > > &cc_xy,vector< vector< vector<float> > > &cc_xz,
        vector< vector< vector<float> > > &cr_yx,vector< vector< vector<float> > > &cr_yy,
        vector< vector< vector<float> > > &cr_yz,vector< vector< vector<float> > > &cv_zx,
        vector< vector< vector<float> > > &cv_zy,vector< vector< vector<float> > > &cv_zz,
        vector< vector< vector<float> > > &A,vector< vector< vector<float> > > &B,
        vector< vector< vector<float> > > &C,vector< vector< vector<float> > > &D,
        vector< vector< vector<float> > > &E,vector< vector< vector<float> > > &F,
        vector< vector< vector<float> > > &G,vector< vector< vector<float> > > &H,
        vector< vector< vector<float> > > &I,vector< vector< vector<float> > > &J,
        vector< vector< vector<float> > > &K,vector< vector< vector<float> > > &L,
        vector< vector< vector<float> > > &M,vector< vector< vector<float> > > &N,
        vector< vector< vector<float> > > &O,vector< vector< vector<float> > > &P,
        vector< vector< vector<float> > > &Q,vector< vector< vector<float> > > &R,
        vector< vector< vector<float> > > &S,vector< vector< vector<float> > > &RHS,
        vector< vector< vector<int> > > ibo,int debug);

void Fill_outside_ring(int nlay,int nrow,int ncol,vector< vector< vector<float> > > &head);

bool One_iteration(int nlay,int nrow,int ncol,float max_head_change,float closure,float accl,
        vector< vector< vector<float> > > &head,vector< vector< vector<int> > > ibo,
        vector< vector< vector<float> > > &A,vector< vector< vector<float> > > &B,
        vector< vector< vector<float> > > &C,vector< vector< vector<float> > > &D,
        vector< vector< vector<float> > > &E,vector< vector< vector<float> > > &F,
        vector< vector< vector<float> > > &G,vector< vector< vector<float> > > &H,
        vector< vector< vector<float> > > &I,vector< vector< vector<float> > > &J,
        vector< vector< vector<float> > > &K,vector< vector< vector<float> > > &L,
    vector< vector< vector<float> > > &M,vector< vector< vector<float> > > &N,
        vector< vector< vector<float> > > &O,vector< vector< vector<float> > > &P,
        vector< vector< vector<float> > > &Q,vector< vector< vector<float> > > &R,
        vector< vector< vector<float> > > &S,vector< vector< vector<float> > > &RHS,
        int debug,bool CONVERGENCE,bool NO_CONVERGENCE);

bool Iterate(int max_iter,int nlay,int nrow,int ncol,int debug,
        float max_head_change,float closure,float accl,
        vector< vector< vector<float> > > &head,vector< vector< vector<float> > > &A,
        vector< vector< vector<float> > > &B,vector< vector< vector<float> > > &C,
        vector< vector< vector<float> > > &D,vector< vector< vector<float> > > &E,
        vector< vector< vector<float> > > &F, vector< vector< vector<float> > > &G,
        vector< vector< vector<float> > > &H,vector< vector< vector<float> > > &I,
        vector< vector< vector<float> > > &J, vector< vector< vector<float> > > &K,
        vector< vector< vector<float> > > &L, vector< vector< vector<float> > > &M,
        vector< vector< vector<float> > > &N,vector< vector< vector<float> > > &O,
        vector< vector< vector<float> > > &P,vector< vector< vector<float> > > &Q,
        vector< vector< vector<float> > > &R, vector< vector< vector<float> > > &S,
        vector< vector< vector<float> > > &RHS,
```

```
        vector< vector< vector<int> > > ibo,bool CONVERGENCE,bool NO_CONVERGENCE);

void Compute_flow(int nlay,int nrow,int ncol,vector< vector< vector<float> > >dx,
        vector< vector< vector<float> > >dy,
        vector< vector< vector<float> > >dz,vector< vector< vector<float> > > &head,
        vector< vector< vector<float> > > &cc_xx,vector< vector< vector<float> > > &cc_xy,
        vector< vector< vector<float> > > &cc_xz,vector< vector< vector<float> > > &cr_yx,
        vector< vector< vector<float> > > &cr_yy,vector< vector< vector<float> > > &cr_yz,
        vector< vector< vector<float> > > &cv_zx,vector< vector< vector<float> > > &cv_zy,
        vector< vector< vector<float> > > &cv_zz,vector< vector< vector<float> > > &qxx,
        vector< vector< vector<float> > > &qyy,
        vector< vector< vector<float> > > &qzz,int debug);

void Print_flow(int nlay,int nrow,int ncol,string qx_file,float average_qx,
        vector< vector< vector<float> > > &qxx);

void Print_head(int nlay,int nrow,int ncol,string head_file,
        vector< vector< vector<float> > > &head,vector< vector< vector<int> > > ibo);

void Check_flow(int nlay,int nrow,int ncol,vector< vector< vector<float> > > &qxx,
        vector< vector< vector<float> > > &qyy,
        vector< vector< vector<float> > > &qzz,vector< vector< vector<float> > > &RHS,
        vector< vector< vector<float> > > &head,vector< vector< vector<float> > >dx,
        vector< vector< vector<float> > >dy,vector< vector< vector<float> > >dz);

void modifyK_well(int nlay,int nrow,int ncol,
        string &kxx_file,string &kyy_file,string &kzz_file,
        vector< vector< vector<float> > > &cc_xx,vector< vector< vector<float> > > &cc_xy,
        vector< vector< vector<float> > > &cc_xz,vector< vector< vector<float> > > &cr_yx,
        vector< vector< vector<float> > > &cr_yy,vector< vector< vector<float> > > &cr_yz,
        vector< vector< vector<float> > > &cv_zx,vector< vector< vector<float> > > &cv_zy,
        vector< vector< vector<float> > > &cv_zz,vector<float>&well_x,vector<float>&well_y,
        vector<float>&well_z,int n_well,int debug);
void modify_RHS_well(int nlay,int nrow,int ncol,vector<float>&well_x,vector<float>&well_y,
        vector<float>&well_z,vector<float>&well_q,int n_well,
        vector< vector< vector<float> > > &RHS);


void main()
{
        int ncol,nrow,nlay,max_iter,n_well,flag_dx,flag_dy,flag_dz,debug;
        float f_dx,f_dy,f_dz,accl,closure,max_head_change,average_qx,average_qy,average_qz;
        max_head_change=0;average_qx=0;average_qy=0;average_qz=0;
        string kxx_file,kyy_file,kzz_file,ihead_file,ibound_file,
                    head_file,qx_file,qy_file,qz_file,well_file,dx_file,dy_file,dz_file;
        bool CONVERGENCE=true;
        bool NO_CONVERGENCE=false;

        Read_para(ncol,nrow,nlay,f_dx,f_dy,f_dz,max_iter,accl,closure,kxx_file,
           kyy_file,kzz_file,ihead_file,ibound_file,head_file,qx_file,qy_file,qz_file,
           n_well,well_file,dx_file,dy_file,dz_file,flag_dx,flag_dy,flag_dz,debug);

        vector< vector< vector<float> > > dx_3d(nlay+2, vector< vector<float> >
                (nrow+2, vector<float>(ncol+2,0)) );
        vector< vector< vector<float> > > dy_3d(nlay+2, vector< vector<float> >
                (nrow+2, vector<float>(ncol+2,0)) );
        vector< vector< vector<float> > > dz_3d(nlay+2, vector< vector<float> >
                (nrow+2, vector<float>(ncol+2,0)) );

        vector< vector< vector< int > > > ibo  (nlay+2, vector< vector< int > >
                (nrow+2, vector< int >(ncol+2,1)) );
        vector< vector< vector<float> > > cc_xx(nlay+1, vector< vector<float> >
                (nrow+1, vector<float>(ncol+1,0)) );
        vector< vector< vector<float> > > cc_xy(nlay+1, vector< vector<float> >
                (nrow+1, vector<float>(ncol+1,0)) );
        vector< vector< vector<float> > > cc_xz(nlay+1, vector< vector<float> >
                (nrow+1, vector<float>(ncol+1,0)) );
        vector< vector< vector<float> > > cr_yx(nlay+1, vector< vector<float> >
                (nrow+1, vector<float>(ncol+1,0)) );
        vector< vector< vector<float> > > cr_yy(nlay+1, vector< vector<float> >
                (nrow+1, vector<float>(ncol+1,0)) );
        vector< vector< vector<float> > > cr_yz(nlay+1, vector< vector<float> >
                (nrow+1, vector<float>(ncol+1,0)) );
        vector< vector< vector<float> > > cv_zx(nlay+1, vector< vector<float> >
                (nrow+1, vector<float>(ncol+1,0)) );
        vector< vector< vector<float> > > cv_zy(nlay+1, vector< vector<float> >
                (nrow+1, vector<float>(ncol+1,0)) );
        vector< vector< vector<float> > > cv_zz(nlay+1, vector< vector<float> >
                (nrow+1, vector<float>(ncol+1,0)) );
```

```
vector< vector< vector<float> > > A(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > B(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > C(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > D(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > E(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > F(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > G(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > H(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > I(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > J(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > K(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > L(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > M(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > N(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > O(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > P(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > Q(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > R(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > S(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > RHS(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );

vector< vector< vector<float> > > head(nlay+2, vector< vector<float> >
        (nrow+2, vector<float>(ncol+2,0)) );

vector< vector< vector<float> > > qxx(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > qyy(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > qzz(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );
vector< vector< vector<float> > > q_well(nlay+1, vector< vector<float> >
        (nrow+1, vector<float>(ncol+1,0)) );

vector<float>well_x(n_well+1,0);
vector<float>well_y(n_well+1,0);
vector<float>well_z(n_well+1,0);
vector<float>well_q(n_well+1,0);

Read_data(ncol,nrow,nlay,kxx_file,kyy_file,kzz_file,ihead_file,ibound_file,
        dx_3d,dy_3d,dz_3d,head,ibo,cc_xx,cc_xy,cc_xz,cr_yx,cr_yy,cr_yz,cv_zx,
        cv_zy,cv_zz,well_file,n_well,well_x,well_y,well_z,well_q,dx_file,dy_file,
        dz_file,f_dx,f_dy,f_dz,flag_dx,flag_dy,flag_dz,debug);

Set_boundary(ncol,nrow,nlay,kxx_file,kyy_file,kzz_file,ibo,cc_xx,cc_xy,
        cc_xz,cr_yx,cr_yy,cr_yz,cv_zx, cv_zy,cv_zz,debug);

if (n_well>0)
        modifyK_well(ncol,nrow,nlay,kxx_file,kyy_file,kzz_file,cc_xx,cc_xy,cc_xz,
        cr_yx,cr_yy,cr_yz,cv_zx, cv_zy,cv_zz,well_x,well_y,well_z,n_well,debug);

Compute_abcs(nlay, nrow, ncol, dx_3d, dy_3d, dz_3d,cc_xx, cc_xy,cc_xz,
        cr_yx,cr_yy,cr_yz,cv_zx,cv_zy,cv_zz,A,B,C,D,E,F,G,
        H,I,J,K,L,M,N,O,P,Q,R,S,RHS,ibo,debug);

if (n_well>0)
        modify_RHS_well(nlay, nrow, ncol,well_x,well_y,well_z,well_q,n_well,RHS);

if(Iterate(max_iter, nlay,nrow, ncol, debug, max_head_change, closure, accl,
        head,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,
        RHS,ibo,CONVERGENCE,NO_CONVERGENCE)==CONVERGENCE)
```

```
        {
                Print_head(nlay,nrow, ncol,head_file,head,ibo);
                cout<<"I am working. Please wait.....";
                Compute_flow(nlay,nrow,ncol,dx_3d,dy_3d,dz_3d,head,cc_xx,cc_xy,cc_xz,cr_yx,
                        cr_yy,cr_yz,cv_zx,cv_zy,cv_zz,qxx,qyy,qzz,debug);
                Print_flow(nlay,nrow,ncol, qx_file,average_qx,qxx);
                Print_flow(nlay,nrow,ncol, qy_file,average_qy,qyy);
                Print_flow(nlay,nrow,ncol, qz_file,average_qz,qzz);
        }
        else
                cout<<"Max Iteration achieved without convergence!!\n";

        if (debug>=1)
                Check_flow(nlay, nrow, ncol,qxx,qyy,qzz,RHS,head,dx_3d,dy_3d,dz_3d);

}

//-----------------------------------------------------------------------//
//       Read parameters for block size, convergence, file names         //
//-----------------------------------------------------------------------//
void Read_para(int &ncol,int &nrow,int &nlay,float &f_dx,float &f_dy,float &f_dz,
         int &max_iter,float &accl,float &closure,string &kxx_file,string &kyy_file,
         string &kzz_file,string &ihead_file,string &ibound_file,string &head_file,
         string &qx_file,string &qy_file,string &qz_file,int &n_well,string &well_file,
         string &dx_file,string &dy_file,string &dz_file,int &flag_dx,int &flag_dy,
         int &flag_dz,int &debug)
{
//------read size of block, convergence standard, and output filename-----------
        cout<<"***********************************"<<endl;
        cout<<"* Flow simulation with full tensor *"<<endl;
        cout<<"*        U.P.V ( Sep.2009)         *"<<endl;
        cout<<"***********************************"<<endl;
        cout<<"                                    "<<endl;
        //cout<<"Parameter file:";
//      string filename;
//      cin>>filename;
        ifstream paraFile;
        paraFile.open("flowxyz.par");

        string comments;
        int i;
        for (i=1;i<=3;i++)
                getline(paraFile,comments,'\n');

        paraFile>>ncol>>nrow>>nlay;
        getline(paraFile,comments,'\n');
        paraFile>>flag_dx>>flag_dy>>flag_dz;
        getline(paraFile,comments,'\n');
        paraFile>>f_dx>>f_dy>>f_dz;
        getline(paraFile,comments,'\n');
        paraFile>>dx_file;
        getline(paraFile,comments,'\n');
        paraFile>>dy_file;
        getline(paraFile,comments,'\n');
        paraFile>>dz_file;
        getline(paraFile,comments,'\n');
        paraFile>>accl>>closure>>max_iter;
        getline(paraFile,comments,'\n');
        paraFile>>kxx_file;
        getline(paraFile,comments,'\n');
        paraFile>>kyy_file;
        getline(paraFile,comments,'\n');
        paraFile>>kzz_file;
        getline(paraFile,comments,'\n');
        paraFile>>ihead_file;
        getline(paraFile,comments,'\n');
        paraFile>>ibound_file;
        getline(paraFile,comments,'\n');
        paraFile>>head_file;
        getline(paraFile,comments,'\n');
        paraFile>>qx_file;
        getline(paraFile,comments,'\n');
        paraFile>>qy_file;
        getline(paraFile,comments,'\n');
        paraFile>>qz_file;
        getline(paraFile,comments,'\n');
        paraFile>>debug;
        getline(paraFile,comments,'\n');
        //-----read well information
        getline(paraFile,comments,'\n');
```

```
        paraFile>>n_well;
        getline(paraFile,comments,'\n');
        paraFile>>well_file;

        paraFile.close();


        if (debug>=1)
        {
                ofstream debug_file;
                debug_file.open("debug.dbg");
                debug_file<<"ncol,nrow,nlay-->"<<ncol<<"  "<<nrow<<"  "<<nlay <<endl;
                debug_file<<"f_dx, f_dy, f_dz-->"<<f_dx<<" "<<f_dy<<" "<<f_dz<<endl;
                debug_file<<"accl, closure,max_iter->"<<accl<<" "<<closure<<" "<<
                                max_iter<<endl;
                debug_file<<"kxx file-->"<<kxx_file<<endl;
                debug_file<<"kyy file---->"<<kyy_file<<endl;
                debug_file<<"kzz file---->"<<kzz_file<<endl;
                debug_file<<"ihead file---->"<<ihead_file<<endl;
                debug_file<<"ibound file---->"<<ibound_file<<endl;
                debug_file<<"head file---->"<<head_file<<endl;
                debug_file<<"qx file---->"<<qx_file<<endl;
                debug_file<<"qy file---->"<<qy_file<<endl;
                debug_file<<"qz file---->"<<qz_file<<endl;
                debug_file<<"number of well---->"<<n_well<<endl;
                debug_file<<"well file---->"<<well_file<<endl;
                debug_file.close();
        }
}
//-----------------------------------------------------------------//
//   Read conductance,initial head and boundary condition          //
//-----------------------------------------------------------------//
void Read_data(int ncol,int nrow,int nlay,string &kxx_file,string &kyy_file,
         string &kzz_file,string &ihead_file,string &ibound_file,
         vector< vector< vector<float> > > &dx_3d,
         vector< vector< vector<float> > > &dy_3d,vector< vector< vector<float> > > &dz_3d,
         vector< vector< vector<float> > > &head,
         vector< vector< vector<int> > > &ibo,vector< vector< vector<float> > > &cc_xx,
         vector< vector< vector<float> > > &cc_xy,vector< vector< vector<float> > > &cc_xz,
         vector< vector< vector<float> > > &cr_yx,vector< vector< vector<float> > > &cr_yy,
         vector< vector< vector<float> > > &cr_yz,vector< vector< vector<float> > > &cv_zx,
         vector< vector< vector<float> > > &cv_zy,vector< vector< vector<float> > > &cv_zz,
         string &well_file,int n_well,vector<float>&well_x,vector<float>&well_y,
         vector<float>&well_z,vector<float>&well_q,string &dx_file,string &dy_file,
         string &dz_file,float f_dx,float f_dy,float f_dz,int flag_dx,
         int flag_dy,int flag_dz,int debug)
{
        int i,j,k;

//------------read the dx
        if (flag_dx==1){
        ifstream dxFile;
        dxFile.open(dx_file.c_str());
        for (i=1;i<=3;i++)
                getline(dxFile,comments,'\n');

        for (j=1;j<=ncol;j++)
                dxFile>>dx_3d[0][0][j];
        dx_3d[0][0][0]=dx_3d[0][0][1];dx_3d[0][0][ncol+1]=dx_3d[0][0][ncol];
        for (i=1;i<=nrow+1;i++)
                for (j=0;j<=ncol+1;j++)
                        dx_3d[0][i][j]=dx_3d[0][0][j];
        for (k=1;k<=nlay+1;k++)
                for (i=0;i<=nrow+1;i++)
                        for (j=0;j<=ncol+1;j++)
                                dx_3d[k][i][j]=dx_3d[0][i][j];
        }
        else
        {
        for (k=0;k<=nlay+1;k++)
                for (i=0;i<=nrow+1;i++)
                        for (j=0; j<=ncol+1;j++)
                                dx_3d[k][i][j]=f_dx;
        }

//------------read the dy
        if (flag_dy==1){
        ifstream dyFile;
        dyFile.open(dy_file.c_str());
        for (i=1;i<=3;i++)
```

```
                getline(dyFile,comments,'\n');

        for (i=nrow;i>=1;i--)
                dyFile>>dy_3d[0][i][0];
        dy_3d[0][0][0]=dy_3d[0][1][0];dy_3d[0][nrow+1][0]=dy_3d[0][nrow][0];
        for (i=0;i<=nrow+1;i++)
                for (j=0;j<=ncol+1;j++)
                        dy_3d[0][i][j]=dy_3d[0][i][0];
        for (k=1;k<=nlay+1;k++)
                for (i=0;i<=nrow+1;i++)
                        for (j=0; j<=ncol+1;j++)
                                dy_3d[k][i][j]=dy_3d[0][i][j];
        }
        else
        {
        for (k=0;k<=nlay+1;k++)
                for (i=0;i<=nrow+1;i++)
                        for (j=0; j<=ncol+1;j++)
                                dy_3d[k][i][j]=f_dy;
        }
//------------read the dz
        if (flag_dz==1){
        ifstream dzFile;
        dzFile.open(dz_file.c_str());
        for (i=1;i<=3;i++)
                getline(dzFile,comments,'\n');

        for (k=nlay;k>=1;k--)
                dzFile>>dz_3d[k][0][0];
        dy_3d[0][0][0]=dy_3d[1][0][0];dy_3d[nlay+1][0][0]=dy_3d[nlay][0][0];
        for (k=0;k<=nlay+1;k++)
                for (i=0;i<=nrow+1;i++)
                        for (j=0; j<=ncol+1;j++)
                                dz_3d[k][i][j]=dz_3d[k][0][0];
        }
        else
        {
        for (k=0;k<=nlay+1;k++)
                for (i=0;i<=nrow+1;i++)
                        for (j=0; j<=ncol+1;j++)
                                dz_3d[k][i][j]=f_dz;
        }
//---------read conductances
        ifstream condxFile;
        condxFile.open(kxx_file.c_str());
        for (i=1;i<=5;i++)
                getline(condxFile,comments,'\n');

        for (k=nlay;k>=1;k--)
                for (i=nrow;i>=1;i--)
                        for (j=1; j<=ncol-1;j++)
                                condxFile>>cc_xx[k][i][j]>>cc_xy[k][i][j]>>cc_xz[k][i][j];
        condxFile.close();

        ifstream condyFile;
        condyFile.open(kyy_file.c_str());
        for (i=1;i<=5;i++)
                getline(condyFile,comments,'\n');
        for (k=nlay;k>=1;k--)
                for (i=nrow-1;i>=1;i--)
                        for (j=1; j<=ncol;j++)
                                condyFile>>cr_yx[k][i][j]>>cr_yy[k][i][j]>>cr_yz[k][i][j];
        condyFile.close();

        ifstream condzFile;
        condzFile.open(kzz_file.c_str());
        for (i=1;i<=5;i++)
                getline(condzFile,comments,'\n');
        for (k=nlay-1;k>=1;k--)
                for (i=nrow;i>=1;i--)
                        for (j=1; j<=ncol;j++)
                                condzFile>>cv_zx[k][i][j]>>cv_zy[k][i][j]>>cv_zz[k][i][j];
        condzFile.close();

//-----------read intial head
        ifstream ihFile;
        ihFile.open(ihead_file.c_str());
        for (i=1;i<=3;i++)
                getline(ihFile,comments,'\n');
        for (k=nlay;k>=1;k--)
```

```
                for (i=nrow;i>=1;i--)
                        for (j=1; j<=ncol;j++)
                                ihFile>>head[k][i][j];
        ihFile.close();
//-----------read boundary condition
        ifstream ibFile;
        ibFile.open(ibound_file.c_str());
        for (i=1;i<=3;i++)
                getline(ibFile,comments,'\n');
        for (k=nlay;k>=1;k--)
                for (i=nrow;i>=1;i--)
                        for (j=1; j<=ncol;j++)
                                ibFile>>ibo[k][i][j];
        ibFile.close();
//-----------read well information
        ifstream wellFile;
        wellFile.open(well_file.c_str());
        getline(wellFile,comments,'\n');
        for (int iw=1;iw<=n_well;iw++){
                wellFile>>well_x[iw]>>well_y[iw]>>well_z[iw]>>well_q[iw];
                getline(wellFile,comments,'\n');
        }
//-----------DEBUG
        if (debug>=1)
        {
                write_debug1(nlay,nrow,ncol,ibound_file,ibo);
                write_debug2(nlay,nrow,ncol,ihead_file,head);
                write_debug3(nlay,nrow,ncol,kxx_file,cc_xx);
                write_debug3(nlay,nrow,ncol,kxx_file,cc_xy);
                write_debug3(nlay,nrow,ncol,kxx_file,cc_xz);
                write_debug3(nlay,nrow,ncol,kyy_file,cr_yx);
                write_debug3(nlay,nrow,ncol,kyy_file,cr_yy);
                write_debug3(nlay,nrow,ncol,kyy_file,cr_yz);
                write_debug3(nlay,nrow,ncol,kzz_file,cv_zx);
                write_debug3(nlay,nrow,ncol,kzz_file,cv_zy);
                write_debug3(nlay,nrow,ncol,kzz_file,cv_zz);
        }

}
//-------------------------------------------------------------------//
//                  modify k for no flow boundary condition          //
//-------------------------------------------------------------------//
void Set_boundary(int ncol,int nrow,int nlay,string &kxx_file,string &kyy_file,
         string &kzz_file,vector< vector< vector<int> > > &ibo,
         vector< vector< vector<float> > > &cc_xx,vector< vector< vector<float> > > &cc_xy,
         vector< vector< vector<float> > > &cc_xz,vector< vector< vector<float> > > &cr_yx,
         vector< vector< vector<float> > > &cr_yy,vector< vector< vector<float> > > &cr_yz,
         vector< vector< vector<float> > > &cv_zx,vector< vector< vector<float> > > &cv_zy,
         vector< vector< vector<float> > > &cv_zz,int debug)
{
        int i,j,k;
        for (k=1;k<=nlay;k++)
                for (i=1;i<=nrow;i++)
                        for (j=1; j<=ncol;j++)
                                if (ibo[k][i][j]==0)
                                {
                                        cc_xx[k][i][j]=0; cc_xx[k][i][j-1]=0;
                                        cc_xy[k][i][j]=0; cc_xy[k][i][j-1]=0;
                                        cc_xz[k][i][j]=0; cc_xz[k][i][j-1]=0;

                                        cr_yx[k][i][j]=0; cr_yx[k][i-1][j]=0;
                                        cr_yy[k][i][j]=0; cr_yy[k][i-1][j]=0;
                                        cr_yz[k][i][j]=0; cr_yz[k][i-1][j]=0;

                                        cv_zx[k][i][j]=0; cv_zx[k-1][i][j]=0;
                                        cv_zy[k][i][j]=0; cv_zy[k-1][i][j]=0;
                                        cv_zz[k][i][j]=0; cv_zz[k-1][i][j]=0;
                                }
        if (debug>=1)
        {
                write_debug3(nlay,nrow,ncol,kxx_file,cc_xx);
                write_debug3(nlay,nrow,ncol,kxx_file,cc_xy);
                write_debug3(nlay,nrow,ncol,kxx_file,cc_xz);
                write_debug3(nlay,nrow,ncol,kyy_file,cr_yx);
                write_debug3(nlay,nrow,ncol,kyy_file,cr_yy);
                write_debug3(nlay,nrow,ncol,kyy_file,cr_yz);
                write_debug3(nlay,nrow,ncol,kzz_file,cv_zx);
                write_debug3(nlay,nrow,ncol,kzz_file,cv_zy);
                write_debug3(nlay,nrow,ncol,kzz_file,cv_zz);
        }
```

```
}
//-----------------------------------------------------------------//
//                 write to debug (type int)                       //
//-----------------------------------------------------------------//
void write_debug1(int nlay,int nrow,int ncol,string &ibound_file,
         vector< vector< vector<int> > > &ibo)
{
        int i,j,k;
        ofstream debug_file;
        debug_file.open("debug.dbg",ios::out|ios::app);
        for (k=0;k<=nlay;k++)
        {
                debug_file<<ibound_file<<"layer"<<k<<endl;
                for (i=0;i<=nrow;i++)
                {
                        for (j=0;j<=ncol;j++)
                        {
                                debug_file.setf(ios_base::right,ios_base::adjustfield);
                                debug_file.precision(8);
                                debug_file.setf(ios_base::showpoint);
                                ios_base::fmtflags old=debug_file.setf
                                        (ios_base::scientific,ios_base::floatfield);
                                debug_file.width(4);
                                debug_file<<ibo[k][i][j];
                        }
                        debug_file<<endl;
                }
                debug_file<<endl;
        }
        debug_file.close();
}
//-----------------------------------------------------------------//
//                 write to debug (head)                           //
//-----------------------------------------------------------------//
void write_debug2(int nlay,int nrow,int ncol,string &ihead_file,
         vector< vector< vector<float> > > &head)
{
        int i,j,k;
        ofstream debug_file;
        debug_file.open("debug.dbg",ios::out|ios::app);
        for (k=0;k<=nlay+1;k++)
        {
                debug_file<<ihead_file<<"layer"<<k<<endl;
                for (i=0;i<=nrow+1;i++)
                {
                        for (j=0;j<=ncol+1;j++)
                        {
                                debug_file.setf(ios_base::right,ios_base::adjustfield);
                                debug_file.precision(8);
                                debug_file.setf(ios_base::showpoint);
                                ios_base::fmtflags old=debug_file.setf
                                        (ios_base::scientific,ios_base::floatfield);
                                debug_file.width(20);
                                debug_file<<head[k][i][j];
                        }
                        debug_file<<endl;
                }
                debug_file<<endl;
        }
        debug_file.close();
}
//-----------------------------------------------------------------//
//                 write to debug (cc,cr,cv)                       //
//-----------------------------------------------------------------//
void write_debug3(int nlay,int nrow,int ncol,string &kxx_file,
         vector< vector< vector<float> > > &cc_xy)
{
        int i,j,k;
        ofstream debug_file;
        debug_file.open("debug.dbg",ios::out|ios::app);
        for (k=0;k<=nlay;k++)
        {
                debug_file<<kxx_file<<"  "<<"cc_cr_cv"<<"  "<<"layer"<<k<<endl;
                for (i=0;i<=nrow;i++)
                {
                        for (j=0;j<=ncol;j++)
                        {
                                debug_file.setf(ios_base::right,ios_base::adjustfield);
```

```
                                debug_file.precision(8);
                                debug_file.setf(ios_base::showpoint);
                                ios_base::fmtflags old=debug_file.setf
                                        (ios_base::scientific,ios_base::floatfield);
                                debug_file.width(20);
                                debug_file<<cc_xy[k][i][j];
                        }
                        debug_file<<endl;
                }
                debug_file<<endl;
        }
        debug_file.close();
}

//-------------------------------------------------------------------//
//                 compute coefficient matrix A--S                   //
//-------------------------------------------------------------------//
void Compute_abcs(int nlay,int nrow,int ncol,vector< vector< vector<float> > >dx,
         vector< vector< vector<float> > >dy,
         vector< vector< vector<float> > >dz,vector< vector< vector<float> > > &cc_xx,
         vector< vector< vector<float> > > &cc_xy,vector< vector< vector<float> > > &cc_xz,
         vector< vector< vector<float> > > &cr_yx,vector< vector< vector<float> > > &cr_yy,
         vector< vector< vector<float> > > &cr_yz,vector< vector< vector<float> > > &cv_zx,
         vector< vector< vector<float> > > &cv_zy,vector< vector< vector<float> > > &cv_zz,
         vector< vector< vector<float> > > &A,vector< vector< vector<float> > > &B,
         vector< vector< vector<float> > > &C,vector< vector< vector<float> > > &D,
         vector< vector< vector<float> > > &E,vector< vector< vector<float> > > &F,
         vector< vector< vector<float> > > &G,vector< vector< vector<float> > > &H,
         vector< vector< vector<float> > > &I,vector< vector< vector<float> > > &J,
         vector< vector< vector<float> > > &K,vector< vector< vector<float> > > &L,
         vector< vector< vector<float> > > &M,vector< vector< vector<float> > > &N,
         vector< vector< vector<float> > > &O,vector< vector< vector<float> > > &P,
         vector< vector< vector<float> > > &Q,vector< vector< vector<float> > > &R,
         vector< vector< vector<float> > > &S,vector< vector< vector<float> > > &RHS,
         vector< vector< vector<int> > > ibo,int debug)
{
        int i,j,k;
        float c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15,
                  c16,c17,c18,c19,c20,c21,c22,c23,c24,c25,c26,c27,c28,c29,c30;
        for (k=1;k<=nlay;k++)
        for (i=1;i<=nrow;i++)
        for (j=1;j<=ncol;j++)
        {
          c1=2*cc_xx[k][i][j]/(dx[k][i][j]+dx[k][i][j+1]);
          c2=0.5*cc_xy[k][i][j]/(0.5*dy[k][i+1][j+1]+0.5*dy[k][i-1][j+1]+dy[k][i][j+1]);
          c3=0.5*cc_xy[k][i][j]/(0.5*dy[k][i+1][j]+0.5*dy[k][i-1][j]+dy[k][i][j]);
          c4=0.5*cc_xz[k][i][j]/(0.5*dz[k+1][i][j+1]+0.5*dz[k-1][i][j+1]+dz[k][i][j+1]);
          c5=0.5*cc_xz[k][i][j]/(0.5*dz[k+1][i][j]+0.5*dz[k-1][i][j]+dz[k][i][j]);
          c6=-2*cc_xx[k][i][j-1]/(    dx[k][i][j]+dx[k][i][j-1]);
          c7=-0.5*cc_xy[k][i][j-1]/(0.5*dy[k][i+1][j]+0.5*dy[k][i-1][j]+dy[k][i][j]);
          c8=-0.5*cc_xy[k][i][j-1]/(0.5*dy[k][i+1][j-1]+0.5*dy[k][i-1][j-1]+dy[k][i][j-1]);
          c9=-0.5*cc_xz[k][i][j-1]/(0.5*dz[k+1][i][j]+0.5*dz[k-1][i][j]+dy[k][i][j]);
          c10=-0.5*cc_xz[k][i][j-1]/(0.5*dz[k+1][i][j-1]+0.5*dz[k-1][i][j-1]+dz[k][i][j-1]);
          c11=0.5*cr_yx[k][i][j]/(0.5*dx[k][i][j+1]+0.5*dx[k][i][j-1]+dx[k][i][j]);
          c12=0.5*cr_yx[k][i][j]/(0.5*dx[k][i+1][j+1]+0.5*dx[k][i+1][j-1]+dx[k][i+1][j]);
          c13=2* cr_yy[k][i][j]/(dy[k][i][j]+dy[k][i+1][j]);
          c14=0.5*cr_yz[k][i][j]/(0.5*dz[k+1][i][j]+0.5*dz[k-1][i][j]+dz[k][i][j]);
          c15=0.5*cr_yz[k][i][j]/(0.5*dz[k+1][i+1][j]+0.5*dz[k-1][i+1][j]+dz[k][i+1][j]);
          c16=-0.5*cr_yx[k][i-1][j]/(0.5*dx[k][i][j+1]+0.5*dx[k][i][j-1]+dx[k][i][j]);
          c17=-0.5*cr_yx[k][i-1][j]/(0.5*dx[k][i-1][j+1]+0.5*dx[k][i-1][j-1]+dx[k][i-1][j]);
          c18=-2*cr_yy[k][i-1][j]/(dy[k][i][j]+dy[k][i-1][j]);
          c19=-0.5*cr_yz[k][i-1][j]/(0.5*dz[k+1][i][j]+0.5*dz[k-1][i][j]+dz[k][i][j]);
          c20=-0.5*cr_yz[k][i-1][j]/(0.5*dz[k+1][i-1][j]+0.5*dz[k-1][i-1][j]+dz[k][i-1][j]);
          c21=0.5*cv_zx[k][i][j]/(0.5*dx[k][i][j+1]+0.5*dx[k][i][j-1]+dx[k][i][j]);
          c22=0.5*cv_zx[k][i][j]/(0.5*dx[k+1][i][j+1]+0.5*dx[k+1][i][j-1]+dx[k+1][i][j]);
          c23=0.5*cv_zy[k][i][j]/(0.5*dy[k][i+1][j]+0.5*dy[k][i-1][j]+dy[k][i][j]);
          c24=0.5*cv_zy[k][i][j]/(0.5*dy[k+1][i+1][j]+0.5*dy[k+1][i-1][j]+dy[k+1][i][j]);
          c25=2*cv_zz[k][i][j]/(dz[k+1][i][j]+dz[k][i][j]);
          c26=-0.5*cv_zx[k-1][i][j]/(0.5*dx[k][i][j+1]+0.5*dx[k][i][j-1]+dx[k][i][j]);
          c27=-0.5*cv_zx[k-1][i][j]/(0.5*dx[k-1][i][j+1]+0.5*dx[k-1][i][j-1]+dx[k-1][i][j]);
          c28=-0.5*cv_zy[k-1][i][j]/(0.5*dy[k][i+1][j]+0.5*dy[k][i-1][j]+dy[k][i][j]);
          c29=-0.5*cv_zy[k-1][i][j]/(0.5*dy[k-1][i+1][j]+0.5*dy[k-1][i-1][j]+dy[k-1][i][j]);
          c30=-2*cv_zz[k-1][i][j]/(dz[k-1][i][j]+dz[k][i][j]);


          A[k][i][j]=dy[k][i][j]*dz[k][i][j]*c1+dx[k][i][j]*dz[k][i][j]*(c11+c16)+
                                  dx[k][i][j]*dy[k][i][j]*(c21+c26);
          B[k][i][j]=dy[k][i][j]*dz[k][i][j]*(-c1+c6)+dx[k][i][j]*dz[k][i][j]*(-c13+c18)+
                                  dx[k][i][j]*dy[k][i][j]*(-c25+c30);
          C[k][i][j]=dy[k][i][j]*dz[k][i][j]*c2+dx[k][i][j]*dz[k][i][j]*c12;
```

```
          D[k][i][j]=dy[k][i][j]*dz[k][i][j]*(-c2)+dx[k][i][j]*dz[k][i][j]*c17;
          E[k][i][j]=dy[k][i][j]*dz[k][i][j]*(c3+c7)+dx[k][i][j]*dz[k][i][j]*c13+
                                dx[k][i][j]*dy[k][i][j]*(c23+c28);
          F[k][i][j]=dy[k][i][j]*dz[k][i][j]*(-c3-c7)+dx[k][i][j]*dz[k][i][j]*(-c18)+
                                dx[k][i][j]*dy[k][i][j]*(-c23-c28);
          G[k][i][j]=dy[k][i][j]*dz[k][i][j]*c4+dx[k][i][j]*dy[k][i][j]*c22;
          H[k][i][j]=dy[k][i][j]*dz[k][i][j]*(-c4)+dx[k][i][j]*dy[k][i][j]*c27;
          I[k][i][j]=dy[k][i][j]*dz[k][i][j]*(c5+c9)+dx[k][i][j]*dz[k][i][j]*(c14+c19)+
                                dx[k][i][j]*dy[k][i][j]*c25;
          J[k][i][j]=dy[k][i][j]*dz[k][i][j]*(-c5-c9)+dx[k][i][j]*dz[k][i][j]*(-c14-c19)+
                                dx[k][i][j]*dy[k][i][j]*(-c30);
          K[k][i][j]=dy[k][i][j]*dz[k][i][j]*(-c6)+dx[k][i][j]*dz[k][i][j]*(-c11-c16)+
                                dy[k][i][j]*dx[k][i][j]*(-c21-c26);
          L[k][i][j]=dy[k][i][j]*dz[k][i][j]*(c8)+dx[k][i][j]*dz[k][i][j]*(-c12);
          M[k][i][j]=dy[k][i][j]*dz[k][i][j]*(-c8)+dx[k][i][j]*dz[k][i][j]*(-c17);
          N[k][i][j]=dy[k][i][j]*dz[k][i][j]*(c10)+dy[k][i][j]*dx[k][i][j]*(-c22);
          O[k][i][j]=dy[k][i][j]*dz[k][i][j]*(-c10)+dy[k][i][j]*dx[k][i][j]*(-c27);
          P[k][i][j]=dx[k][i][j]*dz[k][i][j]*(c15)+dy[k][i][j]*dx[k][i][j]*(c24);
          Q[k][i][j]=dx[k][i][j]*dz[k][i][j]*(-c15)+dy[k][i][j]*dx[k][i][j]*(c29);
          R[k][i][j]=dx[k][i][j]*dz[k][i][j]*(c20)+dy[k][i][j]*dx[k][i][j]*(-c24);
          S[k][i][j]=dx[k][i][j]*dz[k][i][j]*(-c20)+dy[k][i][j]*dx[k][i][j]*(-c29);
        }

        if(debug>=1)
        {
          ofstream debug_file;
          debug_file.open("debug.dbg",ios::out|ios::app);
          debug_file<<"A"<<"  "<<"B"<<"  "<<"C"<<"  "<<"D"<<"  "<<"E"<<"  "<<"F"<<
                    "  "<<"G"<<"  "<<"H"<<"  "<<"I"<<"  "<<"J"<<"  "<<"K"<<"  "
                    <<"L"<<"  "<<"M"<<"  "<<"N"<<"  "<<"O"<<"  "<<"P"<<"  "<<"Q"<<
                    "  "<<"R"<<"  "<<"S"<<"  "<<"RHS"<<endl;
          for (k=1;k<=nlay;k++){
           debug_file<<"layer"<<"  "<<k<<endl;
                for (i=1;i<=nrow;i++){
                  for (j=1;j<=ncol;j++){
                        debug_file.setf(ios_base::right,ios_base::adjustfield);
                        debug_file.precision(8);
                        debug_file.setf(ios_base::showpoint);
                        ios_base::fmtflags old=debug_file.setf
                                (ios_base::scientific,ios_base::floatfield);
                        debug_file.width(20);
                        debug_file<<A[k][i][j]<<"   "<<B[k][i][j]<<"   "
                        <<C[k][i][j]<<"   "<<D[k][i][j]<<"   "
                        <<E[k][i][j]<<"   "<<F[k][i][j]<<"   "
                        <<G[k][i][j]<<"   "<<H[k][i][j]<<"  "
                        <<I[k][i][j]<<"   "<<J[k][i][j]<<"   "
                        <<K[k][i][j]<<"   "<<L[k][i][j]<<"   "
                        <<M[k][i][j]<<"   "<<N[k][i][j]<<"   "
                        <<O[k][i][j]<<"   "<<P[k][i][j]<<"   "
                        <<Q[k][i][j]<<"   "<<R[k][i][j]<<"   "
                        <<S[k][i][j]<<"    "<<RHS[k][i][j]<<endl;
                                }debug_file<<endl;
                         }debug_file<<endl;
          }debug_file.close();
        }
}
//-------------------------------------------------------------------------//
//                        SOR iteration                                    //
//-------------------------------------------------------------------------//
bool Iterate(int max_iter,int nlay,int nrow,int ncol,int debug,
          float max_head_change,float closure,float accl,
          vector< vector< vector<float> > > &head,vector< vector< vector<float> > > &A,
          vector< vector< vector<float> > > &B,vector< vector< vector<float> > > &C,
          vector< vector< vector<float> > > &D,vector< vector< vector<float> > > &E,
          vector< vector< vector<float> > > &F, vector< vector< vector<float> > > &G,
          vector< vector< vector<float> > > &H,vector< vector< vector<float> > > &I,
          vector< vector< vector<float> > > &J, vector< vector< vector<float> > > &K,
          vector< vector< vector<float> > > &L, vector< vector< vector<float> > > &M,
          vector< vector< vector<float> > > &N,vector< vector< vector<float> > > &O,
          vector< vector< vector<float> > > &P,vector< vector< vector<float> > > &Q,
          vector< vector< vector<float> > > &R, vector< vector< vector<float> > > &S,
          vector< vector< vector<float> > > &RHS,
          vector< vector< vector<int> > > ibo,bool CONVERGENCE,bool NO_CONVERGENCE)
{
        int m,ifk,i,j,k;
        ifk=0;
        for(m=1;m<=max_iter;m++)
        {
          Fill_outside_ring(nlay, nrow, ncol,head);
           if(debug>=1 && m==1 && ifk==0)
```

```
                {
                  ofstream debug_file;
                debug_file.open("debug.dbg",ios::out|ios::app);
                    debug_file<<"extended head"<<endl;
                    for (k=0;k<=nlay+1;k++){
                                debug_file<<"layer"<<"  "<<k<<endl;
                                for (i=0;i<=nrow+1;i++){
                                        for (j=0;j<=ncol+1;j++){
                                        debug_file.setf
                                                (ios_base::right,ios_base::adjustfield);
                                        debug_file.precision(8);
                                        debug_file.setf(ios_base::showpoint);
                                        ios_base::fmtflags old=debug_file.setf
                                                (ios_base::scientific,ios_base::floatfield);
                                        debug_file.width(20);
                                        debug_file<<head[k][i][j];
                                        }
                                        debug_file<<endl;
                                }
                                debug_file<<endl;
                        }

                        debug_file.close();
                }//end of if

                if(One_iteration(nlay,nrow,ncol,max_head_change,closure,accl,head,
                        ibo,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,RHS,
                        debug,CONVERGENCE,NO_CONVERGENCE)==CONVERGENCE) break;
        }
        if(m>max_iter)
                return NO_CONVERGENCE;
        else
                return CONVERGENCE;
}
//-------------------------------------------------------------------------//
//                        extend the head for no flow boundary condition     //
//-------------------------------------------------------------------------//
void Fill_outside_ring(int nlay,int nrow,int ncol,vector< vector< vector<float> > > &head)
{
        int i,j,k;
        for(k=1;k<=nlay;k++)
        {
                head[k][0][0]=head[k][1][1];
                head[k][0][ncol+1]=head[k][1][ncol];
                head[k][nrow+1][0]=head[k][nrow][1];
                head[k][nrow+1][ncol+1]=head[k][nrow][ncol];

                for(i=1;i<=nrow;i++)
                {
                        head[k][i][0]=head[k][i][1];
                        head[k][i][ncol+1]=head[k][i][ncol];
                }

                for(j=1;j<=ncol;j++)
                {
                        head[k][0][j]=head[k][1][j];
                        head[k][nrow+1][j]=head[k][nrow][j];
                }
        }

        for(i=0;i<=nrow+1;i++)
                for(j=0;j<=ncol+1;j++)
                {
                        head[0][i][j]=head[1][i][j];
                        head[nlay+1][i][j]=head[nlay][i][j];
                }
}
//-------------------------------------------------------------------------//
//                       One iteration                                     //
//-------------------------------------------------------------------------//
bool One_iteration(int nlay,int nrow,int ncol,float max_head_change,float closure,float accl,
          vector< vector< vector<float> > > &head,vector< vector< vector<int> > > ibo,
          vector< vector< vector<float> > > &A,vector< vector< vector<float> > > &B,
          vector< vector< vector<float> > > &C,vector< vector< vector<float> > > &D,
          vector< vector< vector<float> > > &E,vector< vector< vector<float> > > &F,
          vector< vector< vector<float> > > &G,vector< vector< vector<float> > > &H,
          vector< vector< vector<float> > > &I,vector< vector< vector<float> > > &J,
          vector< vector< vector<float> > > &K,vector< vector< vector<float> > > &L,
          vector< vector< vector<float> > > &M,vector< vector< vector<float> > > &N,
          vector< vector< vector<float> > > &O,vector< vector< vector<float> > > &P,
```

```
          vector< vector< vector<float> > > &Q,vector< vector< vector<float> > > &R,
          vector< vector< vector<float> > > &S,vector< vector< vector<float> > > &RHS,
          int debug,bool CONVERGENCE,bool NO_CONVERGENCE)
{
        int i,j,k;
        int ii,kk,jj;
        float intermediate_h,increment;
        for(k=1;k<=nlay;k++)
        for(i=1;i<=nrow;i++)
        for (j=1;j<=ncol;j++)
        if(ibo[k][i][j]==1){
        //cope with inactive cells
        for (kk=k-1;kk<=k+1;kk++)
        for (ii=i-1;ii<=i+1;ii++)
        for (jj=j-1;jj<=j+1;jj++){
        if ((kk==k-1 && ii==i-1 && jj==j-1) || (kk==k+1 && ii==i-1 && jj==j-1) ||
            (kk==k-1 && ii==i+1 && jj==j-1) || (kk==k+1 && ii==i+1 && jj==j-1) ||
                (kk==k-1 && ii==i+1 && jj==j+1) || (kk==k+1 && ii==i+1 && jj==j+1) ||
                (kk==k-1 && ii==i-1 && jj==j+1) || (kk==k+1 && ii==i-1 && jj==j+1) ||
                (kk==k   && ii==i   && jj==j  ))
                // these cells cannot be used to finite-difference.
                break;
        int count=0;
        float sum_head=0;
        if (ibo[kk][ii][jj]==0)  //the neighbour 6 cells are considered in 3D.
        {
        if (ibo[kk+1][ii][jj]  !=0 )
                {count=count+1;sum_head=sum_head+head[kk+1][ii  ][jj  ];}
        if (ibo[kk-1][ii][jj]  !=0 )
                {count=count+1;sum_head=sum_head+head[kk-1][ii  ][jj  ];}
        if (ibo[kk][ii-1][jj]  !=0 )
                {count=count+1;sum_head=sum_head+head[kk  ][ii-1][jj  ];}
        if (ibo[kk][ii+1][jj]  !=0 )
                {count=count+1;sum_head=sum_head+head[kk  ][ii+1][jj  ];}
        if (ibo[kk][ii][jj+1]  !=0 )
                {count=count+1;sum_head=sum_head+head[kk  ][ii  ][jj+1];}
        if (ibo[kk][ii][jj-1]  !=0 )
                {count=count+1;sum_head=sum_head+head[kk  ][ii  ][jj-1];}
        head[kk][ii][jj]=sum_head/count;}
        }
        //cope with inactive cell
        intermediate_h=(-(A[k][i][j]*head[k][i][j+1]+C[k][i][j]
                *head[k][i+1][j+1]+D[k][i][j]*head[k][i-1][j+1]
                +E[k][i][j]*head[k][i+1][j]+F[k][i][j]
                *head[k][i-1][j]+G[k][i][j]*head[k+1][i][j+1]
                +H[k][i][j]*head[k-1][i][j+1]+I[k][i][j]
                *head[k+1][i][j]+J[k][i][j]*head[k-1][i][j]
                +K[k][i][j]*head[k][i][j-1]+L[k][i][j]
                *head[k][i+1][j-1]+M[k][i][j]*head[k][i-1][j-1]
                +N[k][i][j]*head[k+1][i][j-1]+O[k][i][j]
                *head[k-1][i][j-1]+P[k][i][j]*head[k+1][i+1][j]
                +Q[k][i][j]*head[k-1][i+1][j]+R[k][i][j]
                *head[k+1][i-1][j]+S[k][i][j]*head[k-1][i-1][j])+RHS[k][i][j])
                /B[k][i][j];
        increment=intermediate_h-head[k][i][j];
        if(fabs(increment)>=max_head_change)
                max_head_change=fabs(increment);
                head[k][i][j]=head[k][i][j]+accl*increment;
                }
        if(max_head_change <=  closure)
                return (CONVERGENCE);
        else
                return(NO_CONVERGENCE);
}
//-------------------------------------------------------------------------//
//                      Compute interface flow                             //
//-------------------------------------------------------------------------//
void Compute_flow(int nlay,int nrow,int ncol,vector< vector< vector<float> > >dx,
          vector< vector< vector<float> > >dy,
          vector< vector< vector<float> > >dz,vector< vector< vector<float> > > &head,
          vector< vector< vector<float> > > &cc_xx,vector< vector< vector<float> > > &cc_xy,
          vector< vector< vector<float> > > &cc_xz,vector< vector< vector<float> > > &cr_yx,
          vector< vector< vector<float> > > &cr_yy,vector< vector< vector<float> > > &cr_yz,
          vector< vector< vector<float> > > &cv_zx,vector< vector< vector<float> > > &cv_zy,
          vector< vector< vector<float> > > &cv_zz,vector< vector< vector<float> > > &qxx,
          vector< vector< vector<float> > > &qyy,
          vector< vector< vector<float> > > &qzz,int debug)
{
        int i,j,k;
        Fill_outside_ring(nlay, nrow, ncol,head);
```

```
        if(debug>=1)
                {
                ofstream debug_file;
                debug_file.open("debug.dbg",ios::out|ios::app);
                debug_file<<"finally extended head"<<endl;
                for (k=0;k<=nlay+1;k++)
                        {
                        debug_file<<"layer"<<"  "<<k<<endl;
                        for (i=0;i<=nrow+1;i++)
                                {
                                for (j=0;j<=ncol+1;j++)
                                        {
                                        debug_file.setf
                                                (ios_base::right,ios_base::adjustfield);
                                        debug_file.precision(8);
                                        debug_file.setf(ios_base::showpoint);
                                        ios_base::fmtflags old=debug_file.setf
                                                (ios_base::scientific,ios_base::floatfield);
                                        debug_file.width(20);
                                        debug_file<<head[k][i][j];
                                        }
                                        debug_file<<endl;
                                }
                                debug_file<<endl;
                        }
                        debug_file.close();
                }
        float c1,c2,c3,c4,c5,c11,c12,c13,c14,c15,c21,c22,c23,c24,c25;
        for(k=1;k<=nlay;k++)
        for(i=1;i<=nrow;i++)
        for(j=1;j<=ncol;j++){
             c1=2*  cc_xx[k][i][j]/(    dx[k][i][j]+dx[k][i][j+1]);
             c2=0.5*cc_xy[k][i][j]/(0.5*dy[k][i+1][j+1]+0.5*dy[k][i-1][j+1]+dy[k][i][j+1]);
             c3=0.5*cc_xy[k][i][j]/(0.5*dy[k][i+1][j]+0.5*dy[k][i-1][j]+dy[k][i][j]);
             c4=0.5*cc_xz[k][i][j]/(0.5*dz[k+1][i][j+1]+0.5*dz[k-1][i][j+1]+dz[k][i][j+1]);
             c5=0.5*cc_xz[k][i][j]/(0.5*dz[k+1][i][j]+0.5*dz[k-1][i][j]+dz[k][i][j]);
             c11=0.5*cr_yx[k][i][j]/(0.5*dx[k][i][j+1]+0.5*dx[k][i][j-1]+dx[k][i][j]);
             c12=0.5*cr_yx[k][i][j]/(0.5*dx[k][i+1][j+1]+0.5*dx[k][i+1][j-1]+dx[k][i+1][j]);
             c13=2*cr_yy[k][i][j]/(dy[k][i][j]+dy[k][i+1][j]);
             c14=0.5*cr_yz[k][i][j]/(0.5*dz[k+1][i][j]+0.5*dz[k-1][i][j]+dz[k][i][j]);
             c15=0.5*cr_yz[k][i][j]/(0.5*dz[k+1][i+1][j]+0.5*dz[k-1][i+1][j]+dz[k][i+1][j]);
             c21=0.5*cv_zx[k][i][j]/(0.5*dx[k][i][j+1]+0.5*dx[k][i][j-1]+dx[k][i][j]);
             c22=0.5*cv_zx[k][i][j]/(0.5*dx[k+1][i][j+1]+0.5*dx[k+1][i][j-1]+dx[k+1][i][j]);
             c23=0.5*cv_zy[k][i][j]/(0.5*dy[k][i+1][j]+0.5*dy[k][i-1][j]+dy[k][i][j]);
             c24=0.5*cv_zy[k][i][j]/(0.5*dy[k+1][i+1][j]+0.5*dy[k+1][i-1][j]+dy[k+1][i][j]);
             c25=2*cv_zz[k][i][j]/(dz[k+1][i][j]+dz[k][i][j]);

             qxx[k][i][j]=-c1*(head[k][i][j+1]-head[k][i][j])-c2*(head[k][i+1][j+1]-
             head[k][i-1][j+1])-c3*(head[k][i+1][j]-head[k][i-1][j])
             -c4*(head[k+1][i][j+1]-head[k-1][i][j+1])-c5*(head[k+1][i][j]-head[k-1][i][j]);
             qyy[k][i][j]=-c11*(head[k][i][j+1]-head[k][i][j-1])-c12*(head[k][i+1][j+1]
             -head[k][i+1][j-1])-c13*(head[k][i+1][j]-head[k][i][j])-c14*(head[k+1][i][j]
             -head[k-1][i][j])-c15*(head[k+1][i+1][j]-head[k-1][i+1][j]);
             qzz[k][i][j]=-c21*(head[k][i][j+1]-head[k][i][j-1])-c22*(head[k+1][i][j+1]
             -head[k+1][i][j-1])-c23*(head[k][i+1][j]-head[k][i-1][j])
             -c24*(head[k+1][i+1][j]-head[k+1][i-1][j])-c25*(head[k+1][i][j]-head[k][i][j]);
        }
}
//-------------------------------------------------------------------------//
//                       output flow ..........                            //
//-------------------------------------------------------------------------//
void Print_flow(int nlay,int nrow,int ncol,string qx_file,
          float average_qx,vector< vector< vector<float> > > &qxx)
{
        int i,j,k;
        ofstream outFlow;
        outFlow.open(qx_file.c_str());

        outFlow<<"flux"<<endl;
        outFlow<<"1"<<endl;
        outFlow<<"Value"<<endl;
        for(k=nlay;k>=1;k--)
        {
                for(i=nrow;i>=1;i--)
                {
                        for (j=1;j<=ncol;j++)
                        {
                                outFlow<<qxx[k][i][j]<<endl;
                        //      average_qx+=qxx[k][i][j];
                        }
```

```
                }
        }
        outFlow.close();
}
}
//-------------------------------------------------------------------------//
//                      output head.....                                   //
//-------------------------------------------------------------------------//
void Print_head(int nlay,int nrow,int ncol,string head_file,
          vector< vector< vector<float> > > &head,vector< vector< vector<int> > > ibo)
{
        int i,j,k;
        ofstream outHead;
        outHead.open(head_file.c_str());
        outHead<<"Simulated head"<<endl;
        outHead<<"1"<<endl;
        outHead<<"value"<<endl;
        for(k=nlay;k>=1;k--)
        {
                for(i=nrow;i>=1;i--)
                {
                        for (j=1;j<=ncol;j++)
                        {
                                if(ibo[k][i][j]==0)
                                {
                                        head[k][i][j]=-9999;
                                        //the head of inactive cells are defined as -9999.
                                }
                                outHead<<head[k][i][j]<<endl;
                        }
                }
        }
        outHead.close();
}

//-------------------------------------------------------------------------//
//                      check flows  .....                                 //
//-------------------------------------------------------------------------//
void Check_flow(int nlay,int nrow,int ncol,
          vector< vector< vector<float> > > &qxx,vector< vector< vector<float> > > &qyy,
          vector< vector< vector<float> > > &qzz,vector< vector< vector<float> > > &RHS,
          vector< vector< vector<float> > > &head,vector< vector< vector<float> > > >dx,
          vector< vector< vector<float> > >dy,vector< vector< vector<float> > >dz)
{
        int i,j,k;
        float qtot=0;
        ofstream debug_file;
        debug_file.open("debug.dbg",ios::out|ios::app);
        debug_file<<"row"<<"  "<<"col"<<"  "
                <<"head[k][i][j]   qxx[k][i][j]   qxx[k][i][j-1]    qyy[k][i][j]
                qyy[k][i-1][j]    qzz[k][i][j]   qzz[k-1][i][j]  well_q[k][i][j]  qtot"<<endl;
        for (k=1;k<=nlay;k++){
                debug_file<<"layer"<<k<<endl;
                for (i=1;i<=nrow;i++){
                        for (j=1;j<=ncol;j++){
                        qtot=qxx[k][i][j]*dy[k][i][j]*dz[k][i][j]-
                        qxx[k][i][j-1]*dy[k][i][j-1]*dz[k][i][j-1]+
                        qyy[k][i][j]*dx[k][i][j]*dz[k][i][j]-
                        qyy[k][i-1][j]*dx[k][i-1][j]*dz[k][i-1][j]
                        +qzz[k][i][j]*dy[k][i][j]*dx[k][i][j]-
                        qzz[k-1][i][j]*dy[k-1][i][j]*dx[k-1][i][j];
                        debug_file.width(4);
                        debug_file<< i;
                        debug_file.width(4);
                        debug_file<< j;
                        debug_file.width(16);
                        debug_file.precision(8);
                        debug_file.setf(ios_base::right,ios_base::adjustfield);
                        debug_file.setf(ios_base::showpoint);
                        debug_file<<"  "<<head[k][i][j]<<"   "<<qxx[k][i][j]
                                <<"   "<<qxx[k][i-1][j]<<"   "<<qyy[k][i][j]<<"   "
                        <<qyy[k][i][j-1]<<"   "<<qzz[k][i][j]<<"   "<<
                        qzz[k-1][i][j]<<"   "<<RHS[k][i][j]<<"    "<<qtot<<endl;
                        }
                        debug_file<<endl;
                }
                debug_file<<endl;
        }
        debug_file.close();
}
```

```
//-------------------------------------------------------------------------//
//                modify the conductance for constant flow well            //
//-------------------------------------------------------------------------//
void modifyK_well(int nlay,int nrow,
          int ncol,string &kxx_file,string &kyy_file,string &kzz_file,
          vector< vector< vector<float> > > &cc_xx,vector< vector< vector<float> > > &cc_xy,
          vector< vector< vector<float> > > &cc_xz,vector< vector< vector<float> > > &cr_yx,
          vector< vector< vector<float> > > &cr_yy,vector< vector< vector<float> > > &cr_yz,
          vector< vector< vector<float> > > &cv_zx,vector< vector< vector<float> > > &cv_zy,
          vector< vector< vector<float> > > &cv_zz,vector<float>&well_x,vector<float>&well_y,
          vector<float>&well_z,int n_well,int debug)
{

        int i,j,k;
        for (int iw=1;iw<=n_well;iw++){
                for (k=1;k<=nlay;k++)
                        for (i=1;i<=nrow;i++)
                                for (j=1; j<=ncol;j++)
                                        if (well_x[iw]==i && well_y[iw]==j &&well_z[iw]==k)
                                        {
                                                for (int kk=1;kk<=k;kk++){
                                                        cv_zx[kk][i][j]=99999;
                                                        cv_zy[kk][i][j]=99999;
                                                        cv_zz[kk][i][j]=99999;
                                                }
                                        }
        }


}
//-------------------------------------------------------------------------//
//                modify RHS for constant flow rate well                   //
//-------------------------------------------------------------------------//
void modify_RHS_well(int nlay,int nrow,int ncol,
           vector<float>&well_x,vector<float>&well_y,
           vector<float>&well_z,vector<float>&well_q,
           int n_well,vector< vector< vector<float> > > &RHS)
{int i,j,k;
        for (int iw=1;iw<=n_well;iw++)
                for (k=1;k<=nlay;k++)
                        for (i=1;i<=nrow;i++)
                                for (j=1; j<=ncol-1;j++)
                                {
                                        if (well_x[iw]==i && well_y[iw]==j &&well_z[iw]==k)
                                                RHS[k][i][j]=RHS[k][i][j] - well_q[iw];
                                }
}
```

B

Example Parameter File

```
*******************************************************
*                  Parameter File                     *
*       Technical University of Valencia,Spain        *
*              Liangping Li, Sep. 2009                *
*    All input and output files have a GeoEAS format.  *
*******************************************************
13 18 7            ** ncol, nrow, nlay
1 1 1              ** flag of dx,dy,dz: 0(const);1(file)
10 10 10           ** constant dx, dy, dz if(0)
interval_x.dat     ** Filename of dx if(1)
interval_y.dat     ** Filename of dY if(1)
interval_z.dat     ** Filename of dZ if(1)
1  1e-7 10000      ** accel, closure(e-9), maxiter
kxx.txt            ** Filename of conductances between columns
kyy.txt            ** Filename of conductances between rows
kzz.txt            ** Filename of conductances between layers
ihead.dat          ** Filename of intial guess head
ibound.dat         ** Filename of boundary condition
Head.dat           ** Filename of simulated head
qx.dat             ** Filename of calculated flow in x direction
qy.dat             ** Filename of calculated flow in y direction
qz.dat             ** Filename of calculated flow in z direction
-------------well
```

```
0                   ** Number of well
well.txt            ** Filename of well information
-------------Debug
0                   ** debug yes(>=1),no(0)
```