Three-dimensional Hydraulic Conductivity Upscaling in Groundwater Modelling

Master Thesis submitted by
Haiyan Zhou

Advisor
J. Jaime Gómez-Hernández

September 2009

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Three-dimensional Hydraulic Conductivity Upscaling in Groundwater Modelling

Master Thesis submitted by
Haiyan Zhou

Advisor
J. Jaime Gómez-Hernández

Master Program
Ingeniería Hidráulica y Medio Ambiente

Department
Ingeniería Hidráulica y Medio Ambiente

September 2009

UNIVERSIDAD POLITECNICA DE VALENCIA

grupo de HIDROGEOLOGIA

Abstract

Groundwater numerical simulation is a tool nowadays routinely used in water resources evaluation. The accuracy of groundwater flow and transport simulations relies very much on the ability to properly characterize the spatial variability of hydraulic conductivity. One of the problems that this characterization faces is the disparity between the sample support and the support used in the discretization of the numerical model. While it is possible to generate realizations of conductivity at the measurement support scale, it is too demanding to perform numerical simulations with such a level of discretization. The need to change of support calls for upscaling techniques. We refer to the scale at which hydraulic conductivity can be characterized as the fine scale, and the scale of the numerical discretization as the coarse scale.

This thesis proposes a three-dimensional hydraulic conductivity upscaling algorithm geared to its use with a finite difference code. Because finite difference codes use the interblock conductivity to compute the groundwater flow between blocks, the algorithm aims at computing the hydraulic conductivity representative of the volume between block centers as direct input to the groundwater flow solver, thus avoiding unnecessary averaging rules between neighboring block conductivities. This is particularly important since at the coarse scale hydraulic conductivities will, in general, have to be represented by full tensors, and the averaging of tensors is not a trivial task. The anisotropic spatial correlation of the hydraulic conductivities at the fine scale, even when these conductivities are considered isotropic at this scale, will induce flow anisotropy at coarser scales.

Determining the interblock upscaled conductivity tensor is done by isolating the fine scale hydraulic conductivities that make up the interblock of interest plus a sufficiently large skin surrounding it, and then solving the groundwater flow equation using several boundary conditions. The symmetric 3D tensor that is capable to best reproduce the average fluxes through the interblock, given the average hydraulic head gradient, for the different boundary conditions is computed by a simple optimization and retained as the interblock conductivity tensor at the coarse scale.

The algorithm has been verified in three synthetic experiments. Hydraulic conductivity at the small scale is considered isotropic to flow in all cases, but displaying different spatial heterogeneity: isotropic spatial correlation, anisotropic correlation, and a sand/shale distribution. In all three cases the upscaled models reproduce very well the average flows between blocks as computed at the fine scale. The speed of the algorithm depends very much on the size of the skin selected to perform the small scale simulations to determine each of the interblock conductivity tensor. The larger the skin, the better the final reproduction of the average flows; however, we found that a skin about half the size of the upscaling block gives good results in the three examples.

Resumen

Los modelos numéricos para la simulación hidrogeológica son herramientas de uso común hoy en día en la evaluación de recursos hidráulicos. La precisión de las simulaciones del flujo de agua subterránea y del transporte de masa dependen, en gran medida, en la caracterización de la variabilidad espacial de la conductividad hidráulica. Uno de los problemas que tiene esta caracterización tiene que ver con la disparidad de escalas entre las muestras y la discretización del modelo numérico. Aunque es posible generar realizaciones de la conductividad hidráulica a la escala de las muestras, es demasiado exigente la simulación numérica a esa escala. Es necesario por tanto desarrollar técnicas de escalado. A partir de aquí nos referiremos a la escala a la que puede caracterizarse la variabilidad espacial de la conductividad como la escala fina, y la escala de la discretización numérica como la escala gruesa.

Esta tesis propone un algoritmo de escalado tridimensional orientado a su uso con códigos de diferencias finitas. Puesto que los programas de diferencias finitas utilizan la conductividad entre bloques para calcular los flujos y establecer el balance de masas, el algoritmo propuesto calcula directamente las conductividades del volumen entre los centros de los bloques para su uso por el programa de diferencias finitas sin necesidad de aplicar ningún tipo de promedio de conductividades entre bloques. Este punto es particularmente importante puesto que a la escala gruesa las conductividades hidráulicas tienen que representarse como tensores, y promediar tensores no es trivial. Una correlación espacial anisotrópica a la escala fina, incluso si las conductividades a esta escala son isótropas, inducirán anisotropía en el flujo a escala gruesa.

El cálculo del tensor de conductividad hidráulica entre bloques se hace aislando las conductividades a escala fina que conforman el volumen entre bloques, más un volumen adicional formado por una piel alrededor de este bloque. En este volumen se resuelve la ecuación de flujo con varias condiciones de contorno. El tensor simétrico tridimensional que mejor reproduce los flujos promedio que atraviesan el volumen entre bloques para los correspondientes gradientes medios se calcula con una simple optimización y se asigna al volumen entre bloques.

El algoritmo se ha verificado en tres casos sintéticos. En los tres casos la conductividad se considera isótropa a la escala fina, pero en cada caso tienen diferentes patrones de variabilidad espacial: isotrópica, anisotrópica y binaria. En los tres casos, el modelo escalado reproduce muy bien los flujos medios entre bloques derivados de las simulaciones a escala fina. La velocidad del algoritmo depende mucho del tamaño de la piel elegida para realizar las simulaciones a escala pequeña para el cálculo de los tensores de conductividad de bloque. Cuanto mayor es el tamaño de la piel utilizada, mejor se reproducen los flujos promedio; sin embargo hemos encontrado que usando una piel de tamaño igual a la mitad del bloque da buenos resultados.

Resum

Els models numèrics per a la simulació hidrogeológica són eines d'ús comú avui dia en l'avaluació de recursos hidràulics. La precisió de les simulacions del flux d'aigua subterrània i del transport de massa depenen, en gran mesura, en la caracterització de la variabilitat espacial de la conductivitat hidràulica. Un dels problemes que té aquesta caracterització té a veure amb la disparitat d'escales entre les mostres i la discretització del model numèric. Encara que és possible generar realitzacions de la conductivitat hidràulica a l'escala de les mostres, és massa exigent la simulació numèrica a aquesta escala. És necessari per tant desenvolupar tècniques d'escalat. A partir d'aquí ens referirem a l'escala a la qual pot caracteritzar-se la variabilitat espacial de la conductivitat com l'escala fina, i l'escala de la discretització numèrica com l'escala grossa.

Aquesta tesi proposa un algorisme d'escalat tridimensional orientat al seu ús amb codis de diferències finites. Ja que els programes de diferències finites utilitzen la conductivitat entre blocs per a calcular els fluxos i establir el balanç de masses, l'algorisme proposat calcula directament les conductivitats del volum entre els centres dels blocs per al seu ús pel programa de diferències finites sense necessitat d'aplicar cap tipus de mitjana de conductivitats entre blocs. Aquest punt és particularment important ja que a l'escala grossa les conductivitats hidràuliques han de representar-se com tensors, i fer la mitjana de tensors no és trivial. Una correlació espacial anisòtropa a l'escala fina, fins i tot si les conductivitats a aquesta escala són isòtropes, induiran anisotropia en el flux a escala grossa.

El càlcul del tensor de conductivitat hidràulica entre blocs es fa aïllant les conductivitats a escala fina que conformen el volum entre blocs, més un volum addicional format per una pell al voltant d'aquest bloc. En aquest volum es resol l'equació de flux amb diverses condicions de contorn. El tensor simètric tridimensional que millor reproduïx els fluxos mitjà que travessen el volum entre blocs per als corresponents gradients mitjos es calcula amb una simple optimització i s'assigna al volum entre blocs.

L'algorisme s'ha verificat en tres casos sintètics. En els tres casos la conductivitat es considera isòtropa a l'escala fina, però en cada cas tenen diferents patrons de variabilitat espacial: isòtropa, anisòtropa i binària. En els tres casos, el model escalat reproduïx molt bé els fluxos mitjos entre blocs derivats

de les simulacions a escala fina. La velocitat de l'algorisme depèn molt de la grandària de la pell triada per a realitzar les simulacions a escala fina per al càlcul dels tensors de conductivitat de bloc. Com més gran és la grandària de la pell utilitzada, millor es reproduïxen els fluxos mitjà; no obstant això hem trobat que usant una pell de grandària igual a la meitat del bloc dóna bons resultats.

Acknowledgements

I want to express thanks to my advisor Prof. Gómez-Hernández for providing me with the opportunity to study in the group of hydrogeology in Universidad Politecnica de Valencia. His pertinent advice and guide, patient explanation, and enthusiastic help is appreciated very much. He never hesitated to assist me to accustom myself to the courses and life in Valencia. I cannot complete the Master thesis so soon without his support.

A financial support from China through China Scholarship Council assures my study in Spain and I will cherish the experience here for ever. Additional economic support is provided by the Spanish Nuclear Waste Management Company, ENRESA, which is also acknowledged.

I would like to thank other professors and my colleagues in the group of hydrogeology, Ándres, Eduardo, Javier, Gero, Jianlin, Salvador, Fran, Oscar and Amanda. They helped me a lot no matter in study or in life, especially during the most difficult time when I arrived at Valencia. I also want to thank all the professors responsible for the master courses who spent much time explaining the content to me in English.

At last, I want to thank my family for their unconditional love and care and they are always my spirit support. Special thanks goes to Liangping who faces the difficulties and shares the happiness with me all the time.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and Background

Hydrogeology has experienced a long evolution since Darcy (1856) conducted experiments to establish the macroscopic relationship between water flux through sands and piezometric head gradient. Nowadays, the use of computer codes for the simulation of groundwater flow and the fate of contaminant in the subsurface is common.

The key parameter in all hydrogeological analyses is the one derived by Darcy: hydraulic conductivity. A proportionality constant between flux and gradient that can vary several orders of magnitude within any given aquifer in a difficult to predict pattern. Proper characterization of the spatial variability of hydraulic conductivity is of paramount importance for the accuracy of numerical model predictions. Such a characterization is done based on measurements, which could be taken over supports ranging from centimeters to a hundreds of meters. Although, there are geostatistical algorithms capable to generate realizations of conductivity at the support scale of the measurements, it would be unfeasible to attempt to run the flow and transport computer codes at this scale, particularly if some type of sensitivity or Monte Carlo analysis is to be performed, which would require multiple runs of the code. Therefore there is a need to transform the finely characterized hydraulic conductivity realization into a coarser discretization through some type of upscaling procedure. This upscaling implies replacing a block made up of heterogeneous conductivities by a homogeneous one that should yield the same total flux through the block as computed in the fine scale simulation. The final block conductivity at the coarse scale will, in general, be anisotropic to flow as a result of the anisotropic spatial continuity of the fine scale conductivities; therefore, block conductivities will be tensors.

1.2 Thesis Outline

An upscaling algorithm for three-dimensional groundwater simulation is proposed in Chapter 2, in which the detailed principle, flowchart explanation and verification with isotropic and anisotropic conductivity fields are presented. Chapter 2 has been submitted to an international journal for publication. A further examination for the performance of proposed algorithm in a binary media aquifer is presented in Chapter 3. Finally, the conclusions are summarized in Chapter 4. The numerical upscaling code is included in Appendix A and explanation of parameter file together with the format of input and output files are given in Appendix B.

submitted to Computers & Geosciences

2 Three-dimensional Hydraulic Conductivity Upscaling in Groundwater Modelling

Abstract

The main point of this paper is to propose a non-local three-dimensional hydraulic conductivity full tensor upscaling algorithm and code. Flow rate and hydraulic head gradient are the variables used to relate the outputs from the fine scale model to the outputs from the coarse scale model. The flows and gradients computed at the coarse scale blocks should match the average values of the corresponding quantities observed at the fine scale. The algorithm is geared towards its use in conjunction with finite-difference codes for the solution of the groundwater flow equation capable of handling full tensor hydraulic conductivities. The finite-difference formulation of the groundwater flow equation requires specifying the hydraulic conductivity at the block interface in order to compute the discharge crossing the interface. Most finite-difference codes take as input conductivities at the blocks and then perform some type of averaging to come up with the interblock conductivity. Typically the harmonic mean of the conductivities of neighboring blocks is used. We propose computing directly the interblock conductivity during the upscaling process thus avoiding the need to average already averaged (upscaled) values of adjacent blocks. This approach also circumvents the problem associated with

averaging conductivity tensors when their principal directions are not aligned with the Cartesian axes. The proposed algorithm is successfully demonstrated in two synthetic examples with spatially isotropic and anisotropic conductivities fields at the fine scale. The computer code is provided with explanation of the input parameters and output results.

2.1 Introduction

Numerical simulation of groundwater flow and solute transport is nowadays widely employed in predicting available groundwater resources and the fate of pollutant plumes. Hydraulic conductivity is, without any doubt, the dominant parameter in the numerical simulations. However, hydraulic conductivities are measured in the field on a support scale which is much smaller than the discretization scale of the numerical mode. Dagan (1986) discusses three support scales: the laboratory, the local, and the regional scale. Transforming hydraulic conductivities from a fine scale onto a coarser scale is termed upscaling and has been the subject of research for many years (not only with regard to hydraulic conductivity but also with regard to other parameters and in other disciplines). The need for upscaling stems from the fact that it is possible to characterize the spatial variability of hydraulic conductivity at very small scales, yet, it is prohibitive to perform numerical simulations at such scale, particularly in the context of sensitivity or Monte-Carlo analyses, which require multiple runs of the computer codes. Many reviews about upscaling are available which present general overviews of upscaling approaches from different perspectives (e.g., Wen and Gómez-Hernández, 1996; Renard and Marsily, 1997; Farmer, 2002; Sanchez-Vila et al., 2006).

Several terms are used which are related to the concept of upscaling conductivities: effective hydraulic conductivity, equivalent conductivity, interpreted conductivity, upscaled conductivity, block conductivity, homogenized conductivity, ..., of which effective conductivity and equivalent conductivity are the most common. Effective conductivity is defined, from a stochastic point of view, through the equation, $E\{q\} = -K_{eff}E\{\nabla h\}$ (Matheron, 1967), where q is specific discharge, $E\{\cdot\}$ indicates expected value computed through an ensemble of realizations at a given location, and ∇h is the hydraulic head gradient. On the other hand, equivalent conductivity represents an average value of hydraulic conductivity within a given block in a given realization. Equivalent conductivities are computed to satisfy certain criteria, such as flow conservation, head reproduction or energy dissipation (Renard and Marsily, 1997; Vermeulen et al., 2006). Under certain conditions, effective conductivity and equivalent conductivity coincide; this happens when the random function describing hydraulic conductivity is stationary, ergodic, satisfy some symmetry properties, and the block size for which the equivalent conductivity is sought is much larger than the scale of correlation of the underlying random function (Matheron, 1967).

We are interested in computing equivalent conductivities to be used as replacement of a heterogeneous block of fine scale conductivities in a coarser groundwater flow model. Therefore, when we refer to block conductivities we are interested in equivalent conductivities, not in effective ones, and, as it will be described below, the block conductivity value should reproduce average flows for given average piezometric head gradients.

We know that for one-dimensional flow, the harmonic mean is the exact equivalent conductivity (Freeze and Cherry, 1979). In two-dimensional flow for media with isotropic spatial correlation and lognormal probability distribution, the geometric mean provides good equivalent conductivities (Matheron, 1967; Gómez-Hernández and Wen, 1994). In perfectly layered media, the equivalent conductivity is tensorial with the principal components equal to the arithmetic and harmonic means and oriented parallel and perpendicular to the layers, respectively. In the most general cases we have to resort to numerical approximations to come up with equivalent conductivities since analytical solutions are rare. Some analytical work has been carried out (Gutjahr et al., 1978; Wit, 1995); however, the assumptions necessary to obtain the results make these analytical solutions of limited practical application. Moreover, all these analytical solutions do not address the problem of identifying complex spatial variability patterns that would yield fully tensorial equivalent conductivities, such as is the case of aquifers with cross-bedded sediments (Bierkens and Weerts, 1994).

Some heuristic rules have been proposed for three-dimensional upscaling; for instance, Journel et al. (1986) proposed power average laws to compute block conductivities when upscaling a three-dimensional sand-shale formation: $K_V = (\frac{1}{V}\int_V k^p dV)^{1/p}$, with varying p as a function of the sand/shale proportion. To our knowledge, Holden and Lia (1992) were the first ones to perform a 3D analysis using full tensors to represent hydraulic conductivity at the coarse scale.

The objective of this paper is to present a very accurate three-dimensional full-tensor hydraulic conductivity upscaling algorithm and code aimed to deriving interblock hydraulic conductivity tensors for direct use in a finite-difference groundwater flow simulation program. Next, the algorithm is described, then two synthetic cases are presented, and the paper ends with some conclusions.

2.2 Algorithm and Implementation

2.2.1 Upscaling Methodology

Following Rubin and Gómez-Hernández (1990) we define block conductivity ($\vec{K}_V$) as the quantity that relates the average specific discharge within a given block to the average head gradient as follows:

$$\frac{1}{V}\int_V \vec{q}dV = -\vec{K}_V(\frac{1}{V}\int_V \vec{\nabla h}\, dV) \tag{2.1}$$

where $\vec{q}$ is the specific discharge, $\vec{\nabla h}$ is the specific piezometric head gradient, both are defined at the fine scale, and the averaging takes place over a block of volume V. The definition can be interpreted as Darcy's law at the coarse scale $\bar{q} = -\vec{K}_V\bar{\nabla h}$, where the bar denotes volumetric average. At the same time, $\vec{q}$ and $\vec{\nabla h}$ are also related by Darcy's law but at fine scale $\vec{q} = -\vec{k}\vec{\nabla h}$. Hereon, we will use the terms "block" and "cell" to denote the basic elements at the coarse and fine scales, respectively.

Except on counted occasions, $\vec{K}_V$ is a three-dimensional full tensor:

$$\vec{K}_V = \begin{pmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{yx} & K_{yy} & K_{yz} \\ K_{zx} & K_{zy} & K_{zz} \end{pmatrix}.$$

The tensor must be symmetric and positive definite (Bear, 1972; Farmer, 2002), that is,

$$\begin{aligned} K_{xy} = K_{yx}, K_{xz} = K_{zx}, K_{yz} &= K_{zy} \\ K_{xx} > 0, K_{yy} > 0, K_{zz} &> 0 \\ K_{xx}K_{yy} &> K_{xy}^2 \\ K_{xx}K_{yy}K_{zz} + 2K_{xy}K_{xz}K_{yz} - K_{xx}K_{yz}^2 - K_{yy}K_{xz}^2 - K_{zz}K_{xy}^2 &> 0. \end{aligned} \tag{2.2}$$

Most computer codes —particularly finite-difference codes— only admit diagonal tensors, something that requires that all blocks have the same principal directions, and that they are all aligned with the Cartesian axes. However, many natural geologic formations, e.g., cross-bedded sediments (Bierkens and Weerts, 1994), would require the use of full tensors. Even, under simple patterns of spatial heterogeneity, a full tensor description yields better results than a diagonal tensor, as will be shown later.

It is worthwhile to note that $\vec{K}_V$ is a function, not only of the cell values of $\vec{q}$ and $\vec{\nabla h}$ in equation (2.1), but also on the boundary conditions used to compute them. This dependence on the boundary conditions is what makes the resulting upscaled block values non-local. We can find in the literature different choices for the boundary conditions used to compute the block values, from permeameter-type boundary condition (two sides with constant head while the others are no-flow boundaries) to linearly varying prescribed heads around the block, to periodic boundary conditions (Pickup et al., 1994; Renard and Marsily, 1997; Wen et al., 2003). The prescribed head boundary condition on all sides of the domain is employed in the proposed procedure.

The best boundary conditions that could be used to determine the block conductivities are prescribed heads equal to the ones observed around the block under the specific (global) flow conditions applied to the entire aquifer. But obtaining these head values would require the solution of the flow problem at the fine scale, beating our objective of alleviating the computing effort for solving the flow equation. As an alternative to the solution of the flow equation at the fine scale over the full domain, Gómez-Hernández (1991) proposed to use a sufficiently large "border ring" or "skin" around the block so as to reduce the impact of the artificial boundary conditions used in the upscaling procedure. By isolating the block of volume V plus a skin around it and then applying prescribed heads as boundary conditions to compute the fine scale solution on the block-plus-skin domain, we let the cells that are neighboring the block control, to some extent, the flow pattern within the block; at the limit, for a skin infinitely large, the flow pattern within the block would be exactly the same as if the flow equation had been solved over the entire aquifer.

Another important feature of our algorithm is that it is aimed at computing directly the interblock conductivity values used by finite-difference codes to solve the groundwater flow equation. For completeness, we offer two alternatives as shown in Figure 2.1: computing the block conductivity tensor, or computing directly the interface conductivity. We argue that for the purpose of reproducing the flow patterns observed at the fine scale it is better to compute directly the interblock conductivities (Li et al., 2010). In the figure we can appreciate the block that is being upscaled plus a skin, a single-cell wide, surrounding it, for both cases. Since the skin is needed to determine the block values for the blocks at the edges of the aquifer, the figure also shows that there is a need to consider a skin around the coarse model, at least of the same width as the skin needed for the upscaling process.

In summary, block upscaling for a given volume within a larger domain that has been discretized into small cells is performed by isolating the cells within the block plus a sufficiently large number of skin cells surrounding it. Then, the flow equation is solved for this set of cells applying linearly varying prescribed heads along the boundaries. Several boundary conditions are used inducing average piezometric head gradients at varying orientations. For each boundary condition, block specific discharges and block head gradients are computed as explained later; then, the symmetric, positive definite tensor that

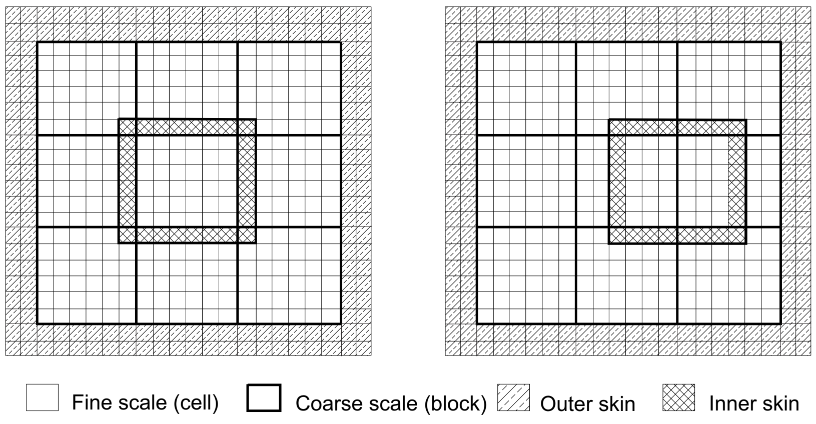

Figure 2.1. Schematic diagram to show the cells that will be considered to compute a block conductivity (left) or an interblock conductivity (right). In both cases, a skin of a single-cell wide around the block is used. Notice also the skin considered around the coarse aquifer model, this skin is necessary for properly computing the block values at the aquifer edges.

best reproduces the relationship between average flows and average gradients according to equation (2.1) is computed and used as the block conductivity.

2.2.2 Implementation Details

The algorithm can be summarized in the following steps:

1. Input parameters and fine scale field to be upscaled. As starting point we need the fine scale hydraulic conductivity realization (i.e, it could have been obtained by geostatistical simulation). At this scale, the aquifer is discretized in equal-sized parallelepiped cells. Then, we need the description of the coarse scale discretization; it can be non-uniform (see Figure 2.2 for a 2D example). Additional input parameters are the size of the skin to be used for upscaling given as number of cells (the same skin size will be used for all blocks), the number of flow problems that will be solved to upscale each block, for each of these problems a different set of linearly varying prescribed heads will be used, these prescribed heads are defined by a vector corresponding to the average head gradient induced in the block. A minimum of two flow problems have to be solved in 2D, and three in 3D, in order to be able to identify all components of the block conductivity tensor, but it is advisable to use a larger number. It is important to notice that because a skin is used to upscale all blocks, even the blocks which are at the aquifer boundaries, it is necessary that

the volume occupied by the fine scale realization be larger than the volume of the coarse scale model by, at least, a number of cells equal to the skin size in all directions (Figure 2.1 shows a coarse scale model with three by three blocks overlaying a fine scale discretization which is two cells wider in all directions than the coarse model)

Figure 2.3 shows four boundary conditions used for the upscaling of a 2D block with indication of the average piezometric head vectors defining them.

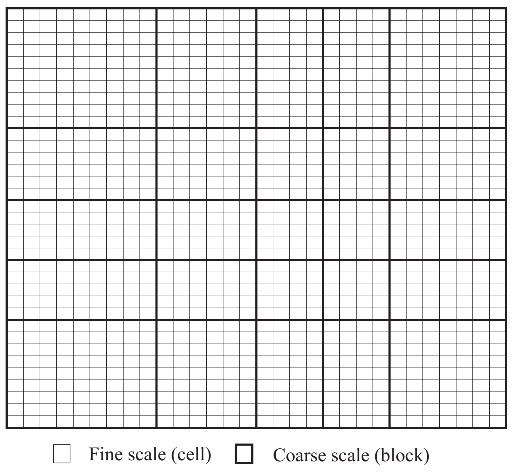

Figure 2.2. Schematic of non-uniform block discretization

2. Decide how to read the fine scale field depending on whether block centered or interface centered conductivity tensors are to be computed. The size of the skin will also have an influence on which values are read from the fine scale field for the computation of the block value. Notice that when computing interface centered conductivity tensors, the number of block values that have to be computed are twice the number of block centered conductivities in 2D and thrice the number in 3D.

3. For each block/interblock and for each mean head gradient direction, establish the boundary conditions and solve the groundwater flow problem assuming incompressible, single phase, steady-state groundwater flow. The preconditioned conjugate-gradient scheme, amended according to Hill (1990), is used to solve the flow equation. The number of flow directions should be sufficient to allow determining the components of the conductivity tensor. In 2D, for each set of boundary conditions we can obtain two equations relating the components of the average specific discharge through the block to the average head gradient within the block,

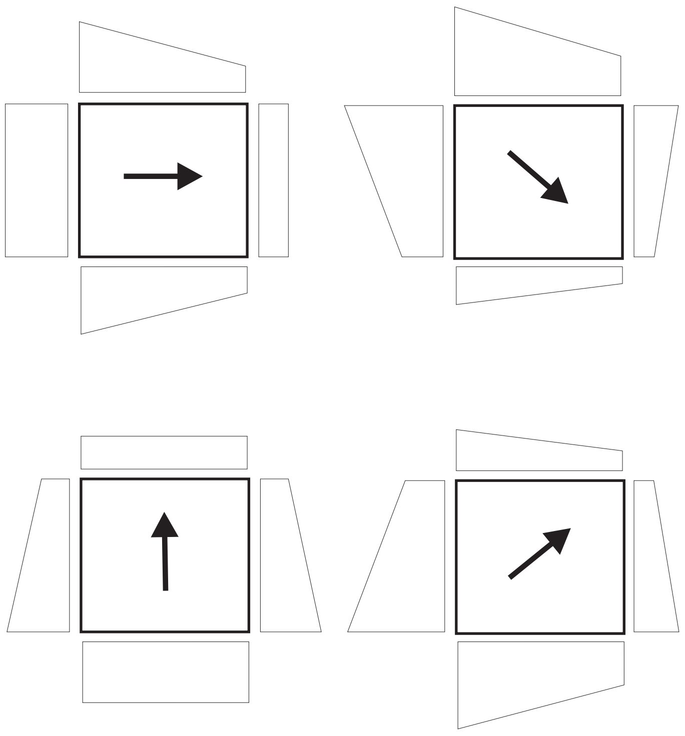

Figure 2.3. Four typical boundary conditions used to estimate a block conductivity in 2D

since we have three unknown components in the conductivity tensor, there is a need to run at least two sets of boundary conditions. We recommend using, at least, four such boundary conditions.

In 3D, for each boundary condition we can establish three equations:

$$\begin{pmatrix} \bar{q}_x \\ \bar{q}_y \\ \bar{q}_z \end{pmatrix} = - \begin{pmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{xy} & K_{yy} & K_{yz} \\ K_{xz} & K_{yz} & K_{zz} \end{pmatrix} \begin{pmatrix} \nabla \bar{h}_x \\ \nabla \bar{h}_y \\ \nabla \bar{h}_z \end{pmatrix}, \tag{2.3}$$

where K_{xx}, K_{xy}, K_{xz}, K_{yy}, K_{yz}, K_{zz} are the unknown components of the block conductivity tensor $\vec{K}_V$. The remaining coefficients in this equation have different meaning depending on whether we are computing a block or an interblock tensor. For the case in which we are interested in the block tensor, $\bar{q}_x$, $\bar{q}_y$, $\bar{q}_z$ and $\nabla \bar{h}_x$, $\nabla \bar{h}_y$, $\nabla \bar{h}_z$ are the averages of specific discharge $\vec{q}$ and gradient $\nabla \vec{h}$, respectively, within the block, which can be calculated as follows:

$$\begin{aligned} \bar{q}_x &= \frac{1}{n_c} \sum_{i=1}^{n_c} q_{xi}, & \nabla \bar{h}_x &= \frac{1}{n_c} \sum_{i=1}^{n_c} \nabla h_{xi}, \\ \bar{q}_y &= \frac{1}{n_c} \sum_{i=1}^{n_c} q_{yi}, & \nabla \bar{h}_y &= \frac{1}{n_c} \sum_{i=1}^{n_c} \nabla h_{yi}, \\ \bar{q}_z &= \frac{1}{n_c} \sum_{i=1}^{n_c} q_{zi}, & \nabla \bar{h}_z &= \frac{1}{n_c} \sum_{i=1}^{n_c} \nabla h_{zi}, \end{aligned} \tag{2.4}$$

where the summations extend over all n_c cells in the block and corresponds to the fluxes or gradients between each cell and the adjacent cell in the corresponding direction.

However, when we seek the interblock conductivity (see right side of Fig. 2.1) since our goal will be to reproduce the fluxes across the interface, we will use a different definition of the terms in Eq. 2.1, more precisely, for an interblock centered on an interface orthogonal to the x direction, we will take $\bar{q}_x$ as the specific discharge across the interface, and $\nabla \bar{h}_x$ as the gradient computed using the average of the piezometric heads at both sides of the interface, while for the y and z directions we will divide the interblock with a hypothetical interface in two equal halves, and we will compute the specific discharge across this interface, and the average gradient as the gradient computed using the average of the heads at both sides of the interface.

For instance, in 2D, for an interblock centered on an interface orthogonal to the x direction, after solving the flow equation for a given set of boundary conditions we will compute the average fluxes and gradients as follows (see Fig. 2.4)

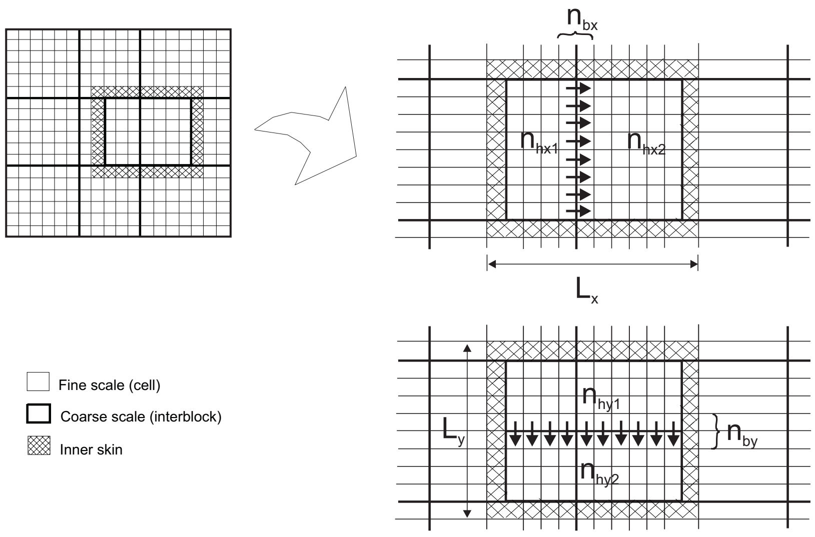

Figure 2.4. Computing the interblock fluxes.

$$
\begin{aligned}
\bar{q_x} &= \frac{1}{n_{bx}} \sum_{i=1}^{n_{bx}} q_{xi} \\
\nabla \bar{h_x} &= \frac{1}{2L_x} \left(\frac{1}{n_{hx2}} \sum_{i2=1}^{n_{hx2}} h_{i2} - \frac{1}{n_{hx1}} \sum_{i1=1}^{n_{hx1}} h_{i1} \right) \\
\bar{q_y} &= \frac{1}{n_{by}} \sum_{j=1}^{n_{by}} q_{yj} \\
\nabla \bar{h_y} &= \frac{1}{2L_y} \left(\frac{1}{n_{hy2}} \sum_{j2=1}^{n_{hy2}} h_{j2} - \frac{1}{n_{hy1}} \sum_{j2=1}^{n_{hy1}} h_{j1} \right)
\end{aligned}
\tag{2.5}
$$

where n_{bx} is the number of cells along the interface, q_{xi} are the specific discharges across the interface cells, n_{hx2} are the number of cells within the interblock to the right of the interface, and n_{hx1} are the number of cells to the left of the interface, h_{i2} and h_{i1} are the associated heads to the right and left of the interface respectively, L_x is the size of the enlarged interblock in the x direction; n_{by} are the number of cells along a fictitious interface running through the center of the interblock orthogonal to the y direction, n_{hy2} are the number of cells below the interface (we consider that the positive y direction runs downwards), n_{hy1} are the number of cells above the interface, h_{j2} and h_{j1} are the associated heads below and above the interface respectively, and L_y is the size of the enlarged interblock in the y direction.

A similar pair of equations are obtained for a block centered on an interface orthogonal to the y direction.

The extension to 3D is straightforward.

4. Once the flow equation has been solved for the set of boundary conditions chosen (as already mentioned we recommend 4 sets in 2D and 8 sets in 3D as a minimum) we can compute all elements in vectors $\vec{q}$ and $\nabla\vec{h}$ for each of the boundary conditions. When we assemble all of these equations we arrive to an overdetermined system of linear equations from which to derive the components of $\vec{K}_V$.

   The system of equations is shown next for the 3D case. Only the six equations, corresponding to two of the boundary conditions used to solve the flow equation are explicitly shown:

$$
-\underbrace{\begin{pmatrix}
\nabla\bar{h}_{x1} & \nabla\bar{h}_{y1} & \nabla\bar{h}_{z1} & 0 & 0 & 0 \\
0 & \nabla\bar{h}_{x1} & 0 & \nabla\bar{h}_{y1} & \nabla\bar{h}_{z1} & 0 \\
0 & 0 & \nabla\bar{h}_{x1} & 0 & \nabla\bar{h}_{y1} & \nabla\bar{h}_{z1} \\
\nabla\bar{h}_{x2} & \nabla\bar{h}_{y2} & \nabla\bar{h}_{z2} & 0 & 0 & 0 \\
0 & \nabla\bar{h}_{x2} & 0 & \nabla\bar{h}_{y2} & \nabla\bar{h}_{z2} & 0 \\
0 & 0 & \nabla\bar{h}_{x2} & 0 & \nabla\bar{h}_{y2} & \nabla\bar{h}_{z2} \\
\dots & & & & &
\end{pmatrix}}_{(3\times n_{BC})\times 6}
\underbrace{\begin{pmatrix} K_{xx} \\ K_{xy} \\ K_{xz} \\ K_{yy} \\ K_{yz} \\ K_{zz} \end{pmatrix}}_{6\times 1}
=
\underbrace{\begin{pmatrix} \bar{q}_{x1} \\ \bar{q}_{y1} \\ \bar{q}_{z1} \\ \bar{q}_{x2} \\ \bar{q}_{y2} \\ \bar{q}_{z2} \\ \dots \end{pmatrix}}_{(3\times n_{BC})\times 1}
\quad (2.6)
$$

   where n_{BC} is the number of boundary conditions used, and the subscripts "1, 2..." refer to the specific boundary condition.

   Standard least squares procedure is employed to solve this overdetermined system of equations (Press et al., 1988).

2.3 Synthetic Experiments

2.3.1 Experiment Description

To demonstrate the accuracy of the program, three synthetic experiments are carried out. In the first experiment hydraulic conductivity displays an isotropic spatial continuity and in the second and third ones hydraulic conductivity is anisotropic. In all cases, the fields were generated using a multiGaussian random field generator. The experiments proceed as follows:

1. Generate a stochastic conductivity field by Sequential Gaussian Simulation (using GCOSIM3D (Gómez-Hernández and Srivastava, 1990), or SGSIM from GSLIB (Deutsch and Journel, 1998)) over a domain discretized into 120 columns, 170 rows and 70 layers with cell size $\triangle x = \triangle y = \triangle z = 1m$. Only the inner 100 columns by 150 rows by 50 layers will be upscaled, considering different skin sizes. The conductivities in both examples follow a log-normal distribution, *i.e.,* $\ln K \sim N(0, 1)$ with an exponential variogram model. Fig. 2.5 shows the isotropic conductivity field used in the first experiment, in which the range is set to 20 m in all directions. In the second experiment the ellipsoid of anisotropy imposes ranges in the three principal directions of 80 m, 20 m and 5 m. The orientation of the ellipsoid is given following the GSLIB convention (Deutsch and Journel, 1998, p. 26): align the ellipsoid with its largest axis parallel to the Y-axis, its medium axis parallel to the X-axis and its shortest axis parallel to the Z-axis, then rotate the ellipsoid in the YZ plane 45° counter-clockwise. Fig. 2.6 displays the second field. The third case has the same anisotropic correlation as the second one, same zero mean, but a variance of 5.

2. Solve the groundwater flow equation at the fine scale. The aquifers in both synthetic examples are assumed to be confined and have prescribed head boundary conditions in the six sides of the domain. The prescribed heads are set so that they induce an overall head gradient from the lower left corner of the top layer to the upper right corner of the bottom layer. MODFLOW2000 (Harbaugh et al., 2000) is used to solve the flow equation. The upscaled model discretization of 10 columns by 15 rows by 5 layers is overlain on this solution and the specific discharges crossing the coarse scale block interfaces are computed. These specific discharges will serve as the reference quantities to be matched by the upscaled model.

3. Perform upscaling. The inner domain of 100 columns by 150 rows by 50 layers of the fine scale conductivity field is upscaled into a coarse scale

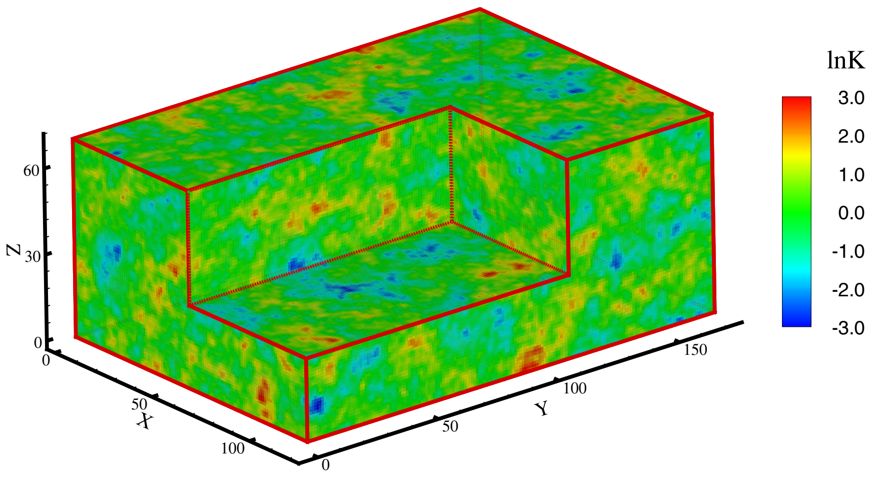

Figure 2.5. The reference isotropic conductivity field

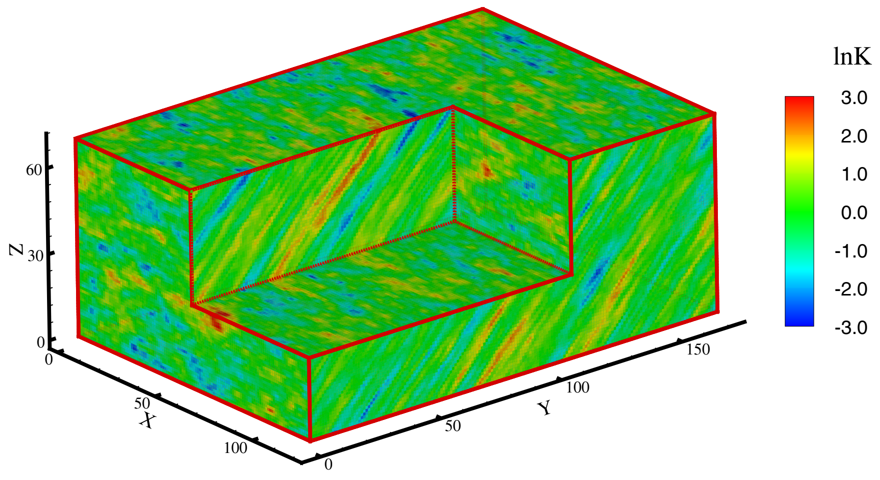

Figure 2.6. The reference anisotropic conductivity field

model of 10 columns by 15 rows by 5 layers, each block of size 10 m $\times$ 10 m $\times$ 10 m.

The upscaled interblock conductivities will be used directly in a finite difference code to solve the flow equation at the coarse scale, thus avoiding any unnecessary averaging of block conductivities to determine the interblock values. In addition, we perform a sensitivity study to the skin size by comparing the results of upscaling with four different skin widths: 0, 2, 5 and 10 fine-scale cells.

4. Solve the groundwater flow equation at the coarse scale with the interblock conductivities computed from step 3. Determine the corresponding specific discharges at the corase scale, which will be compared with the reference values obtained in step 2. For the solution of the flow equation with generic conductivity tensors and direct input of interblock conductivities we wrote a specific numerical code. The interested read should refer to Li et al. (2010) for the details in the flow simulator.

5. Perform comparisons. We will compare the interblock fluxes at the coarse scale with those obtained in the reference simulation. For this comparison, besides graphical representations we will quantify the departure between reference and upscaled values using "Root Mean Square Error" (*RMSE* in short)

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\bar{q}_i^f - \bar{q}_i^c)^2}, \tag{2.7}$$

where $\bar{q}_i^c$ and $\bar{q}_i^f$ are the ith interblock specific discharge at the coarse and fine scales, respectively, and N is the total number of interblocks computed.

2.3.2 Verification of the Flow Rate Reproduction

Isotropic Conductivity Field

The comparison of the specific discharges obtained from the coarse scale and the fine scale models is presented in Fig. 2.7. It can be found readily from Fig. 2.7 that the specific discharge is well reproduced in the upscaled model even when it is computed using a block value that has been computed with no skin or with a small skin. At the same time, with the increase of the size of the skin, the specific discharges at the coarse scale approximate better those of the fine scale. The *RMSE*s of specific discharge resulting from the simulations for different skins are listed in Table 2.1. *RMSE*s of $\bar{q}_x$, $\bar{q}_y$, $\bar{q}_z$ decrease as the skin grows, confirming the fact that computation of block conductivity should be non-local, i.e., it should account for the conductivities surrounding the block being upscaled. It is worth noting that the *RMSE*s for a skin of size 10 cells did not exhibit any notable decrease compared with that for a skin of 5 cells. From a computational point of view, the computer time needed for the upscaling grows exponentially with the size of the skin, thus it is important to balance the upscaling quality and the computational cost.

In this example, a skin of 5-cell width is sufficient to produce a good upscaling. We should notice that the size of the local model that is analyzed to perform the upscaling for this skin size is 20 m $\times$ 20 m$\times$ 20 m, which coincides with the correlation length of the fine scale field. As it is well known, whenever the blocks are larger than the correlation scales, the flow pattern is mostly determined by the conductivities within the blocks and not so much by the conductivities outside and boundary conditions (Gómez-Hernández, 1991). From our experience we have found that with a skin size equal to half the block size is generally enough to get good flow upscaling results.

Table 2.1. Root Mean Square Error *(RMSE)* of specific discharge in the isotropic case

Skin size	0 cells	2 cells	5 cells	10 cells
$\bar{q}_x$	0.145	0.111	0.082	0.074
$\bar{q}_y$	0.112	0.075	0.052	0.046
$\bar{q}_z$	0.119	0.084	0.062	0.056

Anisotropic Conductivity Field

Fig. 2.8 is used to illustrate the performance of the upscaling computer code in an anisotropic conductivity field. Compared with the isotropic example, the upscaled conductivity in the anisotropic field is more sensitive to the size of the skin and the flux reproduction is noticeably improved when a larger skin is employed. *RMSE*s for $\bar{q}_x$, $\bar{q}_y$, $\bar{q}_z$ are presented in Table 2.2. We found that the *RMSE*s in the anisotropic field are comparable with those in isotropic field (Table 2.1), indicating that the proposed upscaling technique works well in both isotropic and anisotropic cases.

As in the isotropic case, a skin size of 5 cells (half the block size) is enough to obtain good upscaling results. In this particular case, the local model size (20 m $\times$ 20 m$\times$ 20 m) is smaller than the correlation length in the direction of maximum continuity and larger than the correlation length in the minimum continuity directions. (Recall the correlation length in the three principal

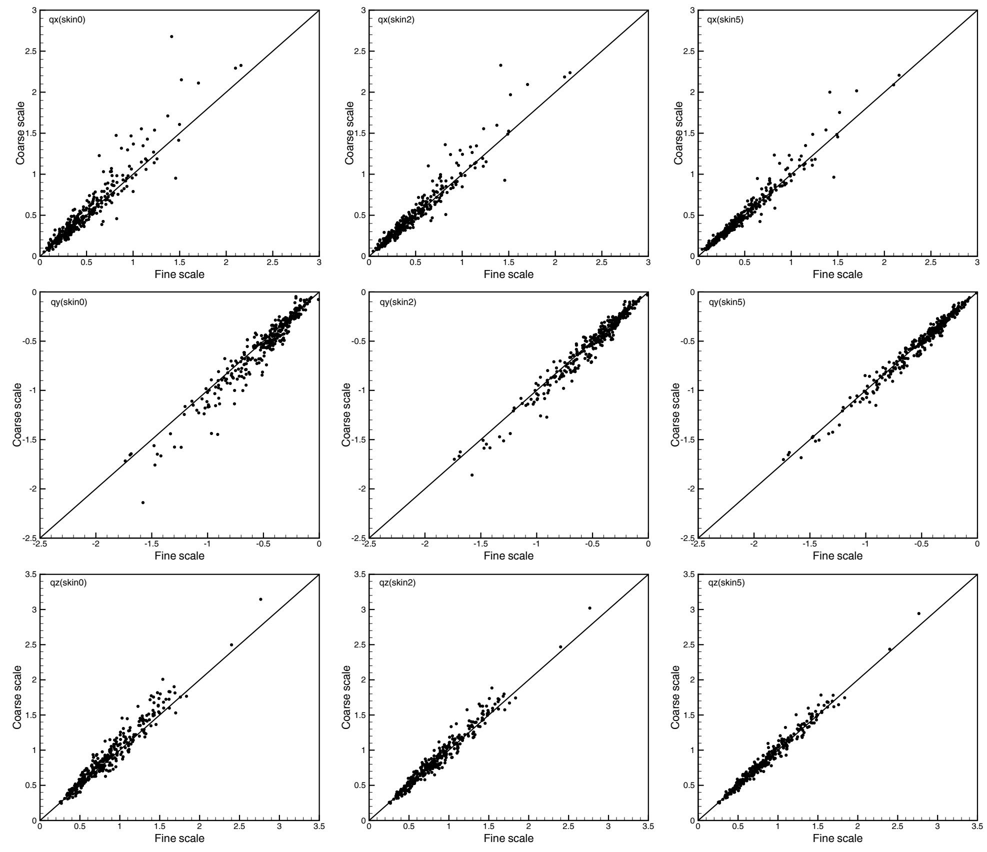

Figure 2.7. Specific discharges obtained with upscaled blocks using different skin sizes for the isotropic field. Comparison with the results from fine scale simulations. The graphs show, from top to bottom $\bar{q}_x$, $\bar{q}_y$ and $\bar{q}_z$, and from left to right skin sizes of 0, 2 and 5 fine scale cells.

directions are 80 m, 20 m and 5 m, and they are not all aligned with the Cartesian axes.)

The previous two synthetic experiments are characterized by a moderate variance, i.e., $\sigma^2_{lnK} = 1$. We have also tested the code with an anisotropic field of variance 5 (similar to the one found, for instance, at the MADE site (Rehfeldt et al., 1992)). The specific discharge reproduction and *RMSE*s are shown in Fig. 2.9 and Table 2.2, respectively. The larger the skin, the better the reproduction. In this particular case, going up to a skin of 10 cells improves with respect to the results using a skin of 5 cells.

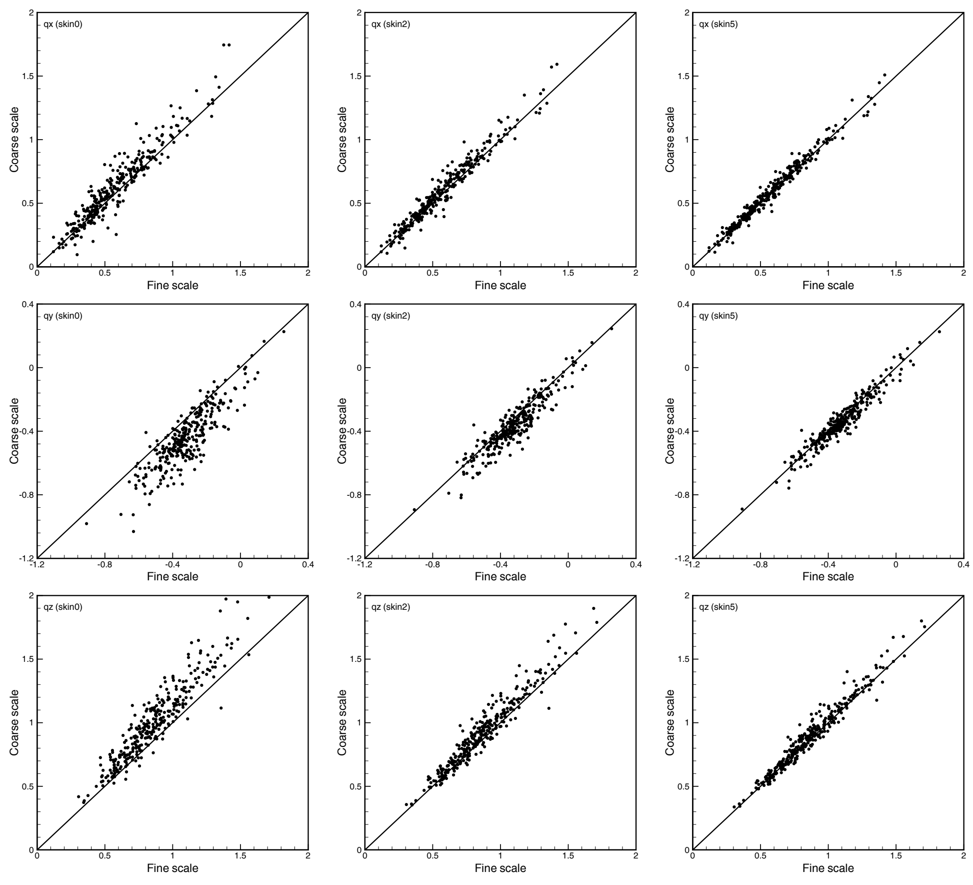

Figure 2.8. Specific discharges obtained with upscaled blocks using different skin sizes for the anisotropic field. Comparison with the results from fine scale simulations. The graphs show, from top to bottom $\bar{q}_x$, $\bar{q}_y$ and $\bar{q}_z$, and from left to right skin sizes of 0, 2 and 5 fine scale cells.

Table 2.2. Root Mean Square Error *(RMSE)* of the coarse scale specific discharge in the anisotropic for different skins and different variances

	skin size	0 cells	2 cells	5 cells	10 cells
	$\bar{q}_x$	0.100	0.062	0.041	0.036
$\sigma^2_{lnK} = 1$	$\bar{q}_y$	0.154	0.073	0.046	0.041
	$\bar{q}_z$	0.180	0.090	0.060	0.052
	$\bar{q}_x$	0.944	0.495	0.325	0.262
$\sigma^2_{lnK} = 5$	$\bar{q}_y$	0.693	0.310	0.236	0.229
	$\bar{q}_z$	0.947	0.370	0.237	0.225

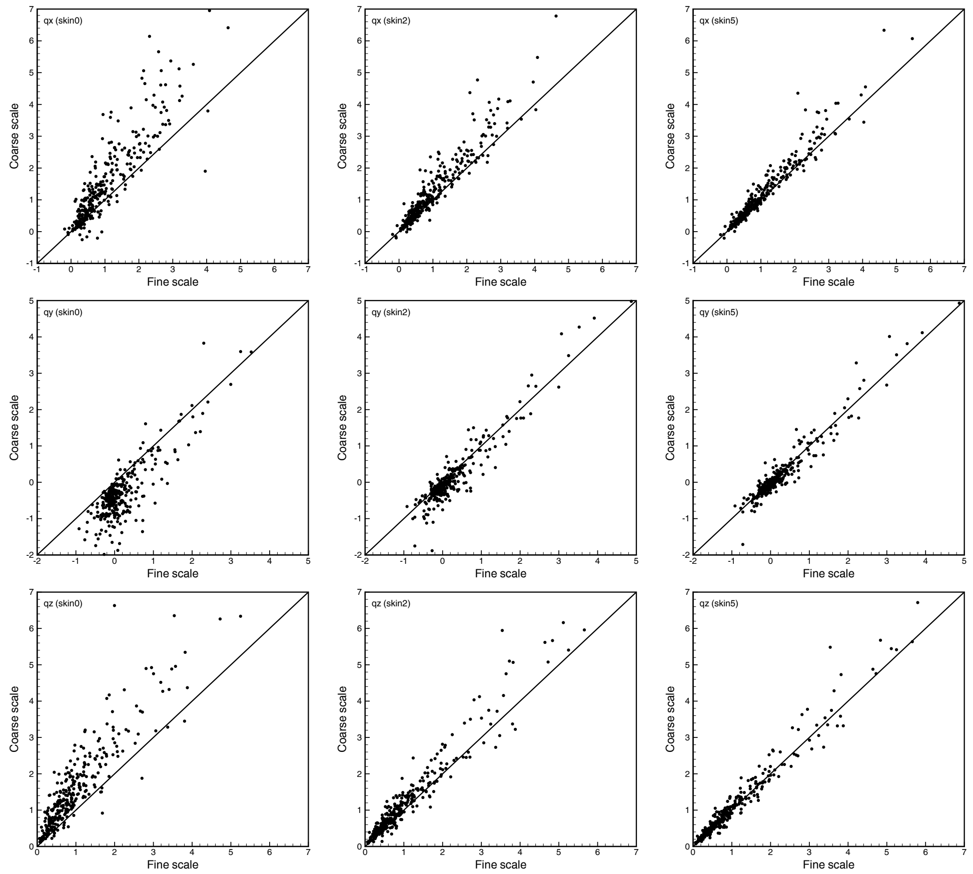

Figure 2.9. Reproduction of specific discharge, from top to bottom we show $\bar{q}_x, \bar{q}_y$ and $\bar{q}_z$, from left to right we show skin size 0, 2 and 5. The heterogeneity of the field is large, $\sigma^2_{lnK} = 5$.

2.4 Conclusions

A three-dimensional hydraulic conductivity full tensor upscaling algorithm and code was proposed in this paper. This work is an extension of the work by Gómez-Hernández (1991) in two dimensions. One of the critical features of this program resided in the capability of obtaining not only block conductivities at the center of blocks but also interblock conductivity at the interfaces. By computing directly interblock conductivities, we avoid the averaging of neighboring block values to come up with the interface conductivity needed by finite-difference codes. The verification of the algorithm requires the development of a groundwater flow solver that can handle interblock tensor conductivities. This code is also available and is described in an accompanying paper (Li et al., 2010). The importance of accounting for the cells surrounding the block being upscaled has been demonstrated, proving the non-local nature of upscaled conductivities. We found that a skin size which is half the block size is enough to produce good results in the two examples with moderate variance discussed in the paper and it should be increased if the approach is applied to highly heterogeneous conductivity field.

Three synthetic hydraulic conductivity fields were generated with isotropic and anisotropic spatial correlations, and different variances. In all cases, the proposed algorithm produced remarkable results, considering that the upscaling process amounted to a loss of resolution of three orders of magnitude, i.e., each group of 10 by 10 by 10 cells was replaced by a singe homogeneous block.

The proposed upscaling algorithm is based on the numerical solution of a local groundwater flow model over an area that includes the block being upscaled plus a surrounding skin. There are as many local groundwater flow solutions as interfaces in the coarse model. This way of upscaling implies a computational burden much larger than empirical results in which each block is replaced by some type of average of the cells within the block, but, undoubtedly produces more accurate results. Care has to be taken in using a skin size large enough to capture the flow response of the block, but as small as possible to reduce the computational cost.

Future tests will be performed in other types of synthetic fields, such as bimodal fields, and non multiGaussian fields with large continuities on the extreme values.

3
Further Performance Test in Sand/Shale Media

To analyze further the performance of the proposed algorithm, a bimodal formation consisting of sand and shale is considered.

3.1 Conductivity Field Description

The aquifer in this case is composed of 80% sand and 20% shale and the discretization of the domain is the same as in the two synthetic experiments in Chapter 2 (120 m $\times$ 170 m $\times$ 70 m in x, y and z directions, an outer skin of 20 m is considered). The sand hydraulic conductivity follows a lognormal distribution with zero mean and unit variance (in log space) and is characterized by an isotropic covariance with range of 20 m. The sand/shale distribution is characterized by an indicator function with an anisotropic covariance with ranges of 35 m and 15 m in the horizontal and 1 m in the vertical, where the direction of greatest continuity is N45°E. The shale are assigned a constant log-conductivity of -8. The sand/shale spatial distribution is generated by sequential indicator simulation using (ISIM3D) (Gómez-Hernández and Srivastava, 1990) and the sand fraction is populated with conductivity values using sequential Gaussian simulation (GCOSIM3D) (Gómez-Hernández and Journel, 1993) and sequential indicator simulation. Figure 3.1 shows the final realization. Equation 3.1 defines the indicator variable for sand/shale, and equation 3.3 shows the indicator variogram, the reproduction of which is given in Fig. 3.2.

$$I(x) = \begin{cases} 1, & \text{if x is shale} \\ 0, & \text{if x is sand} \end{cases} \tag{3.1}$$

$$\gamma(h') = 0.16[1 - \exp(-3h')] \tag{3.2}$$

where,

$$h' = \begin{pmatrix} h_{x'} \\ h_{y'} \\ h_{z'} \end{pmatrix} = \begin{pmatrix} \frac{1}{35} & 0 & 0 \\ 0 & \frac{1}{15} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} h_x \\ h_y \\ h_z \end{pmatrix} \tag{3.3}$$

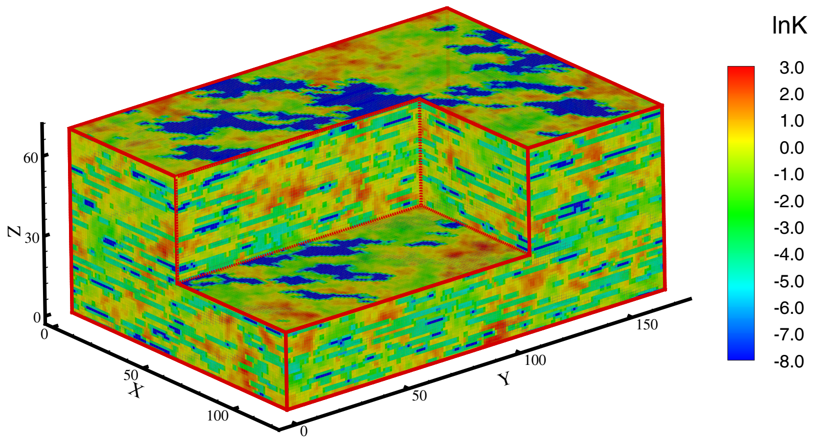

Figure 3.1. The hypothetic bimodal conductivity field

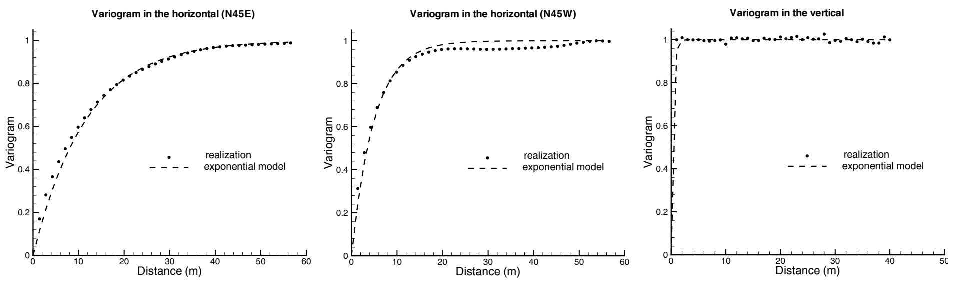

Figure 3.2. Variograms of the indicator variable used to generate the shale distribution in the aquifer for the three principal directions of continuity

Prescribed head boundaries are applied again similar to the two synthetic examples in Section 2.3.1. In this case, the head boundaries induce an average hydraulic gradient, from the upper left corner of the bottom layer to the lower right corner of the top layer (Fig. 3.3).

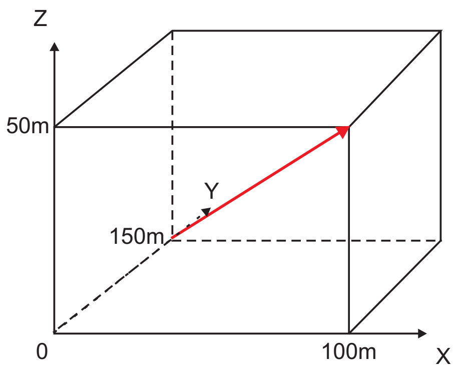

Figure 3.3. Schematic of average flow gradient induced by the boundary conditions

3.2 Results and Discussion

Reproduction of specific discharges across the block interface along the x, y and z directions and Root Mean Square Errors ($RMSE$s) are presented in Fig. 3.4 and Table 3.1, respectively. One can find that the fluxes are well reproduced especially when the a skin of at least 5 cells width is considered. Note that the convergence of the numerical scheme during flow simulation is not trivial in such aquifers where the hydraulic conductivities vary between extreme values as happens in this example, i.e., mean of lnK are -8 in the shales and 0 in the sands. Thus, one problem while the proposed algorithm is applied in the bimodal case involves the intense computation burden required to solve the flow equations within each block at the fine scale. We found a balance between computation burden and flux reproduction accuracy (Fig. 3.4 and Table 3.1), with a skin 5 cell wide. A skin half of a block in width is sufficient to present an acceptable upscaling result in this example. This conclusion is consistent with that in the isotropic and anisotropic examples in Chapter 2.

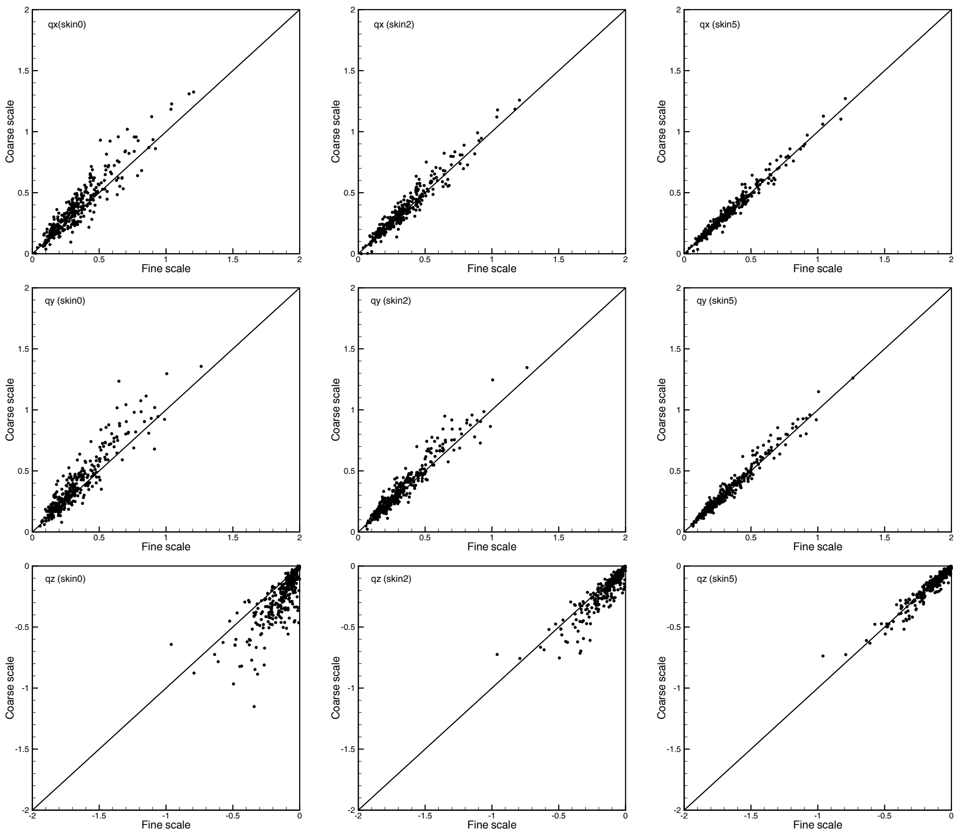

Figure 3.4. Specific discharge reproduction in the sand-shale aquifer with different skin sizes (from left to right: skin width of 0, 2 and 5 cells, from top to bottom, $\bar{q}_x$, $\bar{q}_y$ and $\bar{q}_z$)

Table 3.1. Root Mean Square Error *(RMSE)* of the coarse scale specific discharge in the bimodal case

	0 Skin	2 Skins	5 Skins	10 Skins
$\bar{q}_x$	0.095	0.054	0.034	0.028
$\bar{q}_y$	0.107	0.066	0.041	0.031
$\bar{q}_z$	0.172	0.084	0.040	0.033

4
Summary and Conclusions

A three-dimensional hydraulic conductivity upscaling algorithm and code are presented, which aims at obtaining equivalent conductivity full tensors at the coarse scale. Indeed, the procedure is an extension of the work by Gómez-Hernández (1991) in two dimensions.

Features of the program can be summarized into four main aspects. First of all, the output block conductivity is a full tensor, which is capable of handling the situations whenever the principal directions are not parallel with the Cartesian axes. Secondly, block conductivities at the center of block as well as interblock conductivities at the interface can be computed with the proposed algorithm. Finite difference flow simulator requires the conductivities at the interface, which usually are internally computed by averaging the values of adjacent blocks, this averaging introduces certain error particularly when having to average full conductivity tensors. By computing the interblock conductivities directly we avoid the averaging process between blocks in the flow simulator and thereby eliminate the errors involved in averaging. However, this approach requires development of a flow simulator which can handle directly tensor the conductivities defined at the interface. Refer to Li et al. (2010) for the details of that simulator. Thirdly, the adjacent cells neighboring the target block being upscaled are considered so as to extend the region over which the flow equations at the fine scale are solved. These surrounding cells are used to proximate the influence of the global boundary conditions over the flow domain. At the same time, the potential correlation of the spatial variability is combined with this skin scheme. Finally, the coarsened block can be nonuniform which allows for finer resolution in regions of interest and coarser ones in the remainder of the aquifer.

Three synthetic experiments are investigated to examine the performance of the proposed algorithm, that is, conductivity fields characterized by different spatial variability (isotropic, anisotropic and bimodal formations consisting of sand and shale). Specific discharges across the block interface in all three examples are well reproduced especially after the neighboring cells around the target block being upscaled are taken into account.

Bibliography

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.

Dagan, G., 1986. Statistical theory of groundwater flow and transport: Pore to laboratory, laboratory to formation, and formation to regional scale. Water Resour. Res. 22 (9), 120S–134S.

Darcy, H., 1856. Les Fontaines Publiques de la Ville de Dijon. Victor Dalmont.

Deutsch, C. V., Journel, A. G., 1998. GSLIB, Geostatistical Software Library and User's Guide, 2nd Edition. Oxford University Press, New York.

Farmer, C. L., 2002. Upscaling: a review. International Journal for Numerical Methods in Fluids 40 (1-2), 63–78.

Freeze, R. A., Cherry, J. A., 1979. Groundwater. Prentice-Hall.

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. In: Geostatistics Troia. No. 1. Kluwer, pp. 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.

Gómez-Hernández, J. J., Wen, X., 1994. Probabilistic assessment of travel times in groundwater modeling. J. of Stochastic Hydrology and Hydraulics 8 (1), 19–56.

Gutjahr, A. L., Gelhar, L. W., Bakr, A. A., MacMillan, J. R., 1978. Stochastic analysis of spatial variability in subsurface flows. 2. evaluation and application. Water Resour. Res. 14 (5), 953–959.

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

Hill, M. C., 1990. Preconditioned conjugate-gradient 2 (PCG2), a computer program for solving ground-water flow equations. US Geological Survey Water Resources Investigations Report 90-4048, 43.

Holden, L., Lia, O., May 1992. A tensor estimator for the homogenization of absolute permeability. Transport in Porous Media 8 (1), 37–46.

Journel, A. G., Deutsch, C. V., Desbarats, A. J., 1986. Power averaging for block effective permeability. Spe 15128.

Li, L. P., Gómez-Hernández, J. J., Zhou, H. Y., 2010. Steady-state groundwater flow modeling with full tensor conductivities using finite differences. Computer and Geosciences Submitted.

Matheron, G. F., 1967. éléments pour une Théorie des Milieux Poreux. Mason Et Cie.

Pickup, G., Ringrose, P., Jensen, J., Sorbie, K., Feb. 1994. Permeability tensors for sedimentary structures. Mathematical Geology 26 (2), 227–250.

Press, W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T., 1988. Numerical recipes in C. Cambridge University Press, Cambridge.

Rehfeldt, K. R., Boggs, J. M., Gelhar, L. W., 1992. Field study of dispersion in a heterogeneous aquifer 3. geostatistical analysis of hydraulic conductivity. Water Resour. Res. 28 (12), 3309–3324.

Renard, P., Marsily, G. D., 1997. Calculating equivalent permeability: A review. Advances in Water Resources 20 (5-6), 253–278.

Rubin, Y., Gómez-Hernández, J. J., 1990. A stochastic approach to the problem of upscaling of conductivity in disordered media, theory and unconditional numerical simulations. Water Resour. Res. 26 (4), 691–701.

Sanchez-Vila, X., Guadagnini, A., Carrera, J., 2006. Representative hydraulic conductivities in saturated groundwater flow. Reviews of Geophysics 44 (3).

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

Wen, X., Durlofsky, L., Edwards, M., Jul. 2003. Use of border regions for improved permeability upscaling. Mathematical Geology 35 (5), 521–547.

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

Wit, A. D., 1995. Correlation structure dependence of the effective permeability of heterogeneous porous media. Physics of Fluids 7 (11), 2553–2562.

A
Numerical Upscaling Code

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//                                                                      %
// Copyright(C)2009, Haiyan Zhou, J.J.Gomez-Hernandez,and Liangping Li. %
//        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.       %
//                                                                      %
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//-----------------------------------------------------------------------

//              Three-dimensional hydraulic conductivity upscaling
//                      in groundwater modelling

/*****************************************************************************************
    This program can be used to upacale the k to both block centered and inter-block
    conductivity full tensor.Two input files are needed to run this program: a parameter
    file and k at the fine scale. The user will be prompted for the name of the input.
    All input and output files follow GSLIB convention.
*****************************************************************************************/

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


void Read_para_k(int &ncmf,int &nrmf,int &nlmf, float &wcmf,float &wrmf,float &wlmf,int &ncpmfx,
                  int &ncpmfy, int &ncpmfz, int &ncpmix,int &ncpmiy,int &ncpmiz,int &ncmg,int &nrmg,
                  int &nlmg, vector <float> & wcmg, vector<float> &wrmg,vector<float> &wlmg,int &ib,
                  int &itt,int &ngra, int &px, vector<vector<int> > &gra, vector <vector<short> > &icon,
                  vector <vector<vector<double> > > &tmf,short &debug);

void caltra01(int nuca,int ib,int nl,int nr, int nc,int pnl,int pnr,int pnc,int nlmg,int nrmg,
                int ncmg,int ncpmfx,int ncpmfy,int ncpmfz, int ncpmix,int ncpmiy,int ncpmiz, float wlmf,
                float wrmf, float wcmf, vector <vector<vector<double> > >tmf,int ngra,int itt,
                vector <float> wcmg,vector<float> wrmg, vector<float> wlmg, vector<vector<int> > gra,
                vector <vector<short> > icon,int nlmf, short debug,int px);

void caltra02(int nl,int nr,int nc,int itt,vector<vector<int> > gra,vector <vector<vector<double> > >tmi,
                float dz,float dy,float dx,int ngra,vector <vector<short> > icon,int ncpmix,int ncpmiy,
                int ncpmiz, vector <double> &tmb,short debug,int ix,int jy,int kz,int nrmg,int ncmg,
                int pnl,int pnr, int pnc,int nlmf,int px,int nunk);

void genblo01(int &nl,int &nr, int &nc,int &pnl,int &pnr,int &pnc,int nlmg,int nrmg, int ncmg,
                int ncpmfx, int ncpmfy, int ncpmfz,int ncpmix, int ncpmiy,int ncpmiz,float wlmf,
                float wrmf,float wcmf, vector <vector<vector<double> > >tmf, vector<vector<int> > gra,
                vector <float> wcmg, vector<float> wrmg,vector<float> wlmg, int ngra,int itt,
                vector <vector<short> > icon, int nlmf,short debug,int px);

void genblo02(int &nl,int &nr, int &nc,int &pnl,int &pnr,int &pnc,int nlmg,int nrmg, int ncmg,
                int ncpmfx, int ncpmfy, int ncpmfz,int ncpmix, int ncpmiy,int ncpmiz,float wlmf,
                float wrmf,float wcmf,vector <vector<vector<double> > >tmf, vector<vector<int> > gra,
                vector <float> wcmg,vector<float> wrmg,vector<float> wlmg, int ngra,int itt,
                vector <vector<short> > icon,int nlmf,short debug,int px);

void genblo03(int &nl,int &nr, int &nc,int &pnl,int &pnr,int &pnc,int nlmg,int nrmg, int ncmg,
                int ncpmfx, int ncpmfy, int ncpmfz,int ncpmix, int ncpmiy,int ncpmiz,float wlmf,
                float wrmf,float wcmf,vector <vector<vector<double> > >tmf, vector<vector<int> > gra,
                vector <float> wcmg,vector<float> wrmg,vector<float> wlmg, int ngra,int itt,
                vector <vector<short> > icon,int nlmf,short debug,int px);

void genblo04(int &nl,int &nr, int &nc,int &pnl,int &pnr,int &pnc,int nlmg,int nrmg, int ncmg,
                int ncpmfx, int ncpmfy, int ncpmfz,int ncpmix, int ncpmiy,int ncpmiz,float wlmf,
                float wrmf,float wcmf,vector <vector<vector<double> > >tmf, vector<vector<int> > gra,
                vector <float> wcmg,vector<float> wrmg,vector<float> wlmg, int ngra,int itt,
                vector <vector<short> > icon,int nlmf,short debug,int px);

void tengen01(int nl,int nr,int nc,int itt,vector<vector<int> > gra,vector <vector<vector<double> > >tmi,
                float dz,float dy,float dx,int ngra,vector <vector<short> > icon,int ncpmix,int ncpmiy,
                int ncpmiz, vector <double> &tmb,short debug,int ix,int jy,int kz,int nrmg,int ncmg,
                int pnl,int pnr, int pnc, int nlmf,int nunk);

void Cond(int nl,int nr,int nc,int iflag,vector <vector<vector<double> > >tmi,
            vector <vector<vector<double> > > &cc, vector <vector<vector<double> > > &cr,
            vector <vector<vector<double> > > &cv,vector <double> &ccc, vector <double> &ccr,
            vector <double> &ccv, float dz, float dy, float dx,int nlmf);

void PCG2(int nl,int nr,int nc,vector <short> ibound,vector <double> ccc,vector <double> ccr,
            vector <double> ccv, vector <double> hcof,vector <double> rhs, vector <double> &hnew,
            vector <vector<vector<double> > > &hh);

void Flumed01(int nr,int nc,vector <vector<vector<double> > > cc,vector <vector<vector<double> > > cr,
                float dx,float dy,int ncpx,int ncpy,double &qx,double &qy,double &xjx,double &yjy,
                vector <vector<vector<double> > > hh);

void Flumed02(int nr,int nc,vector <vector<vector<double> > > cc,vector <vector<vector<double> > > cr,
                float dx,float dy, int pnr,int pnc,int ncpx,int ncpy,double &qx,double &qy,double &xjx,
                double &yjy, vector <vector<vector<double> > > hh);

void Flumed03(int nl,int nr,int nc,vector <vector<vector<double> > > cc,
                vector <vector<vector<double> > > cr,vector <vector<vector<double> > > cv,float dx,
                float dy, float dz,double &qx,double &qy,double &qz, double &xjx,double &yjy,double &zjz,
                vector <vector<vector<double> > > hh,int ncpx,int ncpy,int ncpz);

void Flumed04(int nl,int nr,int nc,vector <vector<vector<double> > > cc,
                vector <vector<vector<double> > > cr,vector <vector<vector<double> > > cv,float dx,
                float dy, float dz,double &qx,double &qy,double &qz, double &xjx,double &yjy,double &zjz,
                vector <vector<vector<double> > > hh,int ncpx,int ncpy, int ncpz, int pnl,int pnr,int pnc);

void Print(int nlmf,int nrmf,int ncmf,int ncpmfx,int ncpmfy,int ncpmfz,
                vector <vector<vector<double> > > tmf);

void Print(int kg,int ig,int jg, int nrmg, int ncmg, int nl, int nr, int nc,
                vector <vector<vector<double> > > tmi);

void Printc(int nl,int nr,int nc,int kg, int ig,int jg,int nrmg,int ncmg,
                vector <vector<vector<double> > > cc);

void Printh(int kount,int nl,int nr,int nc,int kz, int ix,int jy, vector <vector<vector<double> > > hh,
                int nrmg,int ncmg);

void Printflux(int kount,int nl,int nr,int nc,int kz, int ix,int jy,double qx,double qy, double qz,
                 double xjx,double yjy,double zjz,int nrmg,int ncmg);

void Overdet(vector <vector <double> >coeff,vector <double> rhs2,vector <double> &tmb,int nunk,int neq,
                int mne);
```

```
void ludcmp(vector <vector<double> > &ata,int n,vector <double> &indx, double &sum);
void lubksb(vector <vector<double> > &ata,int n,vector <double> &indx,vector <double> &tmb,double &sum);

void medgeo(vector <vector<vector<double> > >tmi,int nl,int nr,int nc,int ncpx,int ncpy,int ncpz,
            vector <double> &tmb);

void cuber(vector <vector<vector<double> > >tmi,int nl,int nr,int nc,int ncpx,int ncpy,int ncpz,
            vector <double> &tmb);

void analy(vector <vector<vector<double> > >tmi,int nl,int nr,int nc,int ncpx,int ncpy,int ncpz,
            vector <double> &tmb);

double harm(double x,double y);
double geom(double x,double y);


void main()
{
       int ncmf,nrmf,nlmf;                                        //discretization at the fine scale
       int ncmg,nrmg,nlmg;                                        //discretization at the coarse scale
       int ncpmfx,ncpmfy,ncpmfz;                                  //size of outer skin
       int ncpmix,ncpmiy,ncpmiz;                                  //size of inner skin
       int ib,itt,ngra, px, nuca;
       int nr,nc,nl,pnl,pnr,pnc;
       nr=nc=nl=pnl=pnr=pnc=0;
       float wcmf,wrmf,wlmf;                                      //size of each cell at the fine scale
       short debug;

       vector <float> wcmg;                                       //size of each block at the coarse scale
       vector <float> wrmg;
       vector <float> wlmg;
       vector<vector<int> > gra;
       vector <vector<short> > icon;
       vector <vector<vector<double> > > tmf;                     //input conductivity to be upscaled

       Read_para_k(ncmf,nrmf,nlmf, wcmf,wrmf,wlmf,ncpmfx,ncpmfy,ncpmfz, ncpmix,ncpmiy,ncpmiz,ncmg,nrmg,
                          nlmg, wcmg, wrmg,wlmg,ib,itt,ngra,px, gra,icon,tmf,debug);

//-----------check tmf-------------
       if (debug>4)
             Print(nlmf,nrmf,ncmf,ncpmfx,ncpmfy,ncpmfz,tmf);

    for (nuca=1;nuca<=ib;nuca++)
             caltra01(nuca,ib,nl,nr,nc,pnl,pnr,pnc,nlmg,nrmg,ncmg,ncpmfx,ncpmfy,ncpmfz,ncpmix,ncpmiy,
                     ncpmiz,wlmf,wrmf,wcmf,tmf,ngra,itt,wcmg,wrmg,wlmg,gra,icon,nlmf,debug,px);

}//end of main()
//----------------------------------------------------------------------------------
//                    Read parameters and conductivity
//----------------------------------------------------------------------------------
void Read_para_k(int &ncmf,int &nrmf,int &nlmf, float &wcmf,float &wrmf,float &wlmf,int &ncpmfx,
                 int &ncpmfy, int &ncpmfz, int &ncpmix,int &ncpmiy,int &ncpmiz,int &ncmg,int &nrmg,
                 int &nlmg, vector <float> & wcmg, vector<float> &wrmg,vector<float> &wlmg,int &ib,
                 int &itt,int &ngra, int &px, vector<vector<int> > &gra, vector <vector<short> > &icon,
                 vector <vector<vector<double> > > &tmf,short &debug)

{
       int i,j,k;
       float c,r,l;
       string comments;

       cout<<"Enter of parameter file:"<<" ";
       string para;
       cin>>para;

       cout<<"Enter the conductivity of the fine scale:"<<" ";
       string conduc;
       cin>>conduc;

       ifstream inFile1;
       inFile1.open(para.c_str());

       for(i=1;i<=3;i++)
             getline(inFile1,comments,'\n');
       inFile1>>ncmf>>nrmf>>nlmf;                                 getline(inFile1,comments,'\n');
       inFile1>>wcmf>>wrmf>>wlmf;                                 getline(inFile1,comments,'\n');
       inFile1>>ncpmfx>>ncpmfy>>ncpmfz;                           getline(inFile1,comments,'\n');
       inFile1>>ncpmix>>ncpmiy>>ncpmiz;                           getline(inFile1,comments,'\n');
       inFile1>>ncmg>>nrmg>>nlmg;                                 getline(inFile1,comments,'\n');
```

```
        for (i=0;i<ncmg;i++)
        {
                inFile1>>c;
                wcmg.push_back(c);
        }                                                             getline(inFile1,comments,'\n');
//---check the column width
        float wmg=0;
        for(i=0;i<ncmg;i++)
                wmg+=wcmg[i];
        if (wmg!=wcmf*ncmf)
                cout<<"Check the parameter file: wcmf, ncmf,wcmg, ncmg"<<endl;

        for (i=0;i<nrmg;i++)
        {
                inFile1>>r;
                wrmg.push_back(r);
        }                                                             getline(inFile1,comments,'\n');
//----check the row width
        wmg=0;
        for(i=0;i<nrmg;i++)
                wmg+=wrmg[i];
        if (wmg!=wrmf*nrmf)
                cout<<"Check the parameter file: wrmf, nrmf,wrmg, nrmg"<<endl;

        for (i=0;i<nlmg;i++)
        {
                inFile1>>l;
                wlmg.push_back(l);
        }
//----check the layer width
        wmg=0;
        for(i=0;i<nlmg;i++)
                wmg+=wlmg[i];
        if (wmg!=wlmf*nlmf)
                cout<<"Check the parameter file: wlmf, nlmf,wlmg, nlmg"<<endl;

        getline(inFile1,comments,'\n');

        inFile1>>ib;                                                  getline(inFile1,comments,'\n');
        inFile1>>itt;                                                 getline(inFile1,comments,'\n');

        if (itt==11)
        {
                inFile1>>px;
                getline(inFile1,comments,'\n');
        }
        else if (itt==31 || itt==32)
        {
                inFile1>>ngra;
                getline(inFile1,comments,'\n');
                int g;
                for (i=0;i<ngra;i++)
                {
                        vector <int> row0;
                        for (j=0;j<3;j++)
                        {
                                inFile1>>g;
                                row0.push_back(g);
                        }
                        gra.push_back(row0);
                }
                getline(inFile1,comments,'\n');

                int ic;
                for (i=0;i<ngra;i++)
                {
                        vector<short>row1;
                        for (j=0;j<6;j++)
                        {
                                inFile1>>ic;
                                row1.push_back(ic);
                        }
                        icon.push_back(row1);
                }
                getline(inFile1,comments,'\n');
        }
        inFile1>>debug;                                               getline(inFile1,comments,'\n');

        inFile1.close();
        inFile1.clear();
        inFile1.open(conduc.c_str());
        for(i=1;i<=3;i++)
                getline(inFile1,comments,'\n');
        double f;
        f=0;
        for (k=0;k<nlmf+2*ncpmfz;k++)
        {
                vector <vector<double> > tmf1;
                for (i=0;i<nrmf+2*ncpmfy;i++)
                {
                        vector<double> tmf2;
                        for(j=0;j<ncmf+2*ncpmfx;j++)
                        {
                                inFile1>>f;
                                tmf2.push_back(f);
                        }
                        tmf1.push_back(tmf2);
                }
                tmf.push_back(tmf1);
        }
        inFile1.close();
}
//------------------------------------------------------------
//               compute conductance cc,cr,cv(Cond)
//--------------------------------------------------------------
void Cond(int nl,int nr,int nc,int iflag,vector <vector<vector<double> > >tmi,
            vector <vector<vector<double> > > &cc, vector <vector<vector<double> > > &cr,
            vector <vector<vector<double> > > &cv,vector <double> &ccc, vector <double> &ccr,
            vector <double> &ccv, float dz, float dy, float dx,int nlmf)
{
        int i,j,k,pun0;
//----------------------------cr-----------------
        for(k=0;k<nl;k++)
                for(i=0;i<nr-1;i++)
                        for(j=0;j<nc;j++)
                        {
                                if (iflag==1)
                                        cr[k][i][j]=harm(tmi[k][i][j],tmi[k][i+1][j])*(dx*dz/dy);
                                else
                                        cr[k][i][j]=geom(tmi[k][i][j],tmi[k][i+1][j])*(dx*dz/dy);
                        }
        for (k=0;k<nl;k++)
                for(j=0;j<nc;j++)
                        cr[k][nr-1][j]=0;
//----------------------------cc-----------------
        for(k=0;k<nl;k++)
                for(i=0;i<nr;i++)
                        for(j=0;j<nc-1;j++)
                        {
                                if (iflag==1)
                                        cc[k][i][j]=harm(tmi[k][i][j],tmi[k][i][j+1])*(dy*dz/dx);
                                else
                                        cc[k][i][j]=geom(tmi[k][i][j],tmi[k][i][j+1])*(dy*dz/dx);
                        }
        for (k=0;k<nl;k++)
                for(i=0;i<nr;i++)
                        cc[k][i][nc-1]=0;
//----------------------------cv-------------
        for(k=0;k<nl-1;k++)
                for(i=0;i<nr;i++)
                        for(j=0;j<nc;j++)
                        {
                                if (iflag==1)
                                        cv[k][i][j]=harm(tmi[k][i][j],tmi[k+1][i][j])*(dy*dx/dz);
                                else
                                        cv[k][i][j]=geom(tmi[k][i][j],tmi[k+1][i][j])*(dy*dx/dz);
                        }
        for (i=0;i<nr;i++)
                for(j=0;j<nc;j++)
                        cv[nl-1][i][j]=0;
//-----------------------to ccc,ccr,ccv-------------
        for(k=0;k<nl;k++)
                for(i=0;i<nr;i++)
                        for (j=0;j<nc;j++)
                        {
                                pun0=k*nr*nc+i*nc+j;
                                ccc[pun0]=cc[k][i][j];
                                ccr[pun0]=cr[k][i][j];
                                ccv[pun0]=cv[k][i][j];
                        }
}
//----------------------------------------
//          harmonic mean
//-------------------------------------------
double harm(double x,double y)
{
        double harm;
        if (x==0 && y==0)
                harm=0;
    else
                harm=2*x*y/(x+y);
    return harm;
}
//--------------------------------------
//             geometric mean
//---------------------------------------
double geom(double x,double y)
{
        double geom;
        geom=sqrt(x*y);
        return geom;
}
//-------------------------------------------------
//   Calculate the geometric mean in Block
//-----------------------------------------------
void medgeo(vector <vector<vector<double> > >tmi,int nl,int nr,int nc,int ncpx,
                        int ncpy,int ncpz, vector <double> &tmb)
{
        int i,j,k;
        double st,nd;
        st=1;
        nd=(nl-2*ncpz)*(nr-2*ncpy)*(nc-2*ncpx);

        for(k=ncpz;k<nl-ncpz;k++)
                for (i=ncpy;i<nr-ncpy;i++)
                        for (j=ncpx;j<nc-ncpx;j++)
                                st=st*pow(tmi[k][i][j],1/nd);
        tmb[0]=st;
}
//----------------------------------------------------
//   Calculate the cuber root mean in Block (isotropic)
//----------------------------------------------------
void cuber(vector <vector<vector<double> > >tmi,int nl,int nr,int nc,int ncpx,
                   int ncpy,int ncpz, vector <double> &tmb)
{
        int i,j,k;
        double st,nd;
        st=0;
        nd=(nl-2*ncpz)*(nr-2*ncpy)*(nc-2*ncpx);

        for(k=ncpz;k<nl-ncpz;k++)
                for (i=ncpy;i<nr-ncpy;i++)
                        for (j=ncpx;j<nc-ncpx;j++)
                                st=st+pow(tmi[k][i][j],0.33333333);

        st=st/nd;
        tmb[0]=pow(st,3);
}
//---------------------------------------------------------------------------
//   Calculate the effective k in Block (anisotropic, Ababou[1990])
//Note: the coefficients in both "pow" functions need input by hand, which is
//calculated according to the formula mentioned in the paper.
//---------------------------------------------------------------------------
void analy(vector <vector<vector<double> > >tmi,int nl,int nr,int nc,int ncpx,
                   int ncpy,int ncpz, vector <double> &tmb)
{
        int i,j,k;
        double st,nd;
        st=0;
        nd=(nl-2*ncpz)*(nr-2*ncpy)*(nc-2*ncpx);

        for(k=ncpz;k<nl-ncpz;k++)
                for (i=ncpy;i<nr-ncpy;i++)
                        for (j=ncpx;j<nc-ncpx;j++)
                                st=st+pow(tmi[k][i][j], 0.904762);

        st=st/nd;
        tmb[0]=pow(st, 1.105263);
}
```

```
//-------------------------------------------
//              print tmf for debug
//-------------------------------------------
void Print(int nlmf,int nrmf,int ncmf,int ncpmfx,int ncpmfy,int ncpmfz,
                  vector <vector<vector<double> > > tmf)
{
        int i,j,k;
        ofstream outFile0;
        outFile0.open("check_tmf.dbg");
        for (k=0;k<nlmf+2*ncpmfz;k++)
        {
                outFile0<<"tmf"<<"  "<<"layer"<<k<<endl;
                for (i=0;i<nrmf+2*ncpmfy;i++)
                {
                        for(j=0;j<ncmf+2*ncpmfx;j++)
                                outFile0<<tmf[k][i][j]<<"  ";
                        outFile0<<endl;
                }
                outFile0<<endl;
        }
        outFile0.close();
}
//--------------------------------------------------------------
//              print conductivity in one block(tmi) for debug
//--------------------------------------------------------------
void Print(int kg,int ig,int jg, int nrmg, int ncmg, int nl, int nr, int nc,
                  vector <vector<vector<double> > > tmi)
{
        int i,j,k;
        ofstream outFile0;
        outFile0.open("check.dbg",ios::out|ios::app);
        outFile0<<"tmi"<<"  "<<"block"<<kg*nrmg*ncmg+ig*ncmg+jg<<endl;

        for(k=0;k<nl;k++)
        {
                for(i=0;i<nr;i++)
                {
                        for(j=0;j<nc;j++)
                                outFile0<<tmi[k][i][j]<<"  ";
                        outFile0<<endl;
                }
                outFile0<<endl;
        }
        outFile0.close();
}
//---------------------------------------------------------
//               print conductance (cc, cr, cv)
//---------------------------------------------------------
void Printc(int nl,int nr,int nc,int kg, int ig,int jg,int nrmg,int ncmg,
                       vector <vector<vector<double> > > cc)
{
        int i,j,k;
        ofstream outFile0;
        outFile0.open("check.dbg",ios::out|ios::app);
        outFile0<<"c"<<"  "<<"block"<<kg*nrmg*ncmg+ig*ncmg+jg<<endl;

        for(k=0;k<nl;k++)
        {
                for(i=0;i<nr;i++)
                {
                        for(j=0;j<nc;j++)
                                outFile0<<cc[k][i][j]<<"  ";

                        outFile0<<endl;
                }
                outFile0<<endl;
        }
        outFile0.close();
}
//------------------------------------------------------------------
//         print head calculated at the fine scale within one block
//------------------------------------------------------------------
void Printh(int kount,int nl,int nr,int nc,int kz, int ix,int jy,
                       vector <vector<vector<double> > > hh,int nrmg,int ncmg)
{
        int i,j,k,n;
        n=0;
        ofstream outFile0;
        outFile0.open("check.dbg",ios::out|ios::app);
        outFile0<<"head"<<"  "<<"block"<<kz*nrmg*ncmg+ix*ncmg+jy<<"  "<<"gra"<<kount<<endl;
        for (k=0;k<nl;k++)
                {
                        for (i=0;i<nr;i++)
                        {
                                for (j=0;j<nc;j++)
                                {
                                        n=j+i*nc+k*nr*nc;
                                        outFile0<<hh[k][i][j]<<"  ";
                                }
                                outFile0<<endl;
                        }
                        outFile0<<endl;
                }
        outFile0.close();
}
//---------------------------------------------------------------------
//         print flux calculated at the fine scale within one block
//---------------------------------------------------------------------
void Printflux(int kount,int nl,int nr,int nc,int kz, int ix,int jy,double qx,double qy,
                           double qz, double xjx,double yjy,double zjz,int nrmg,int ncmg)
{
        ofstream outFile0;
        outFile0.open("check.dbg",ios::out|ios::app);
        outFile0<<"flux"<<"  "<<"block"<<kz*nrmg*ncmg+ix*ncmg+jy<<"  "<<"gra"<<kount<<endl;

        outFile0<<qx<<"   "<<qy<<"   "<<qz<<"    "<<xjx<<"   "<<yjy<<"   "<<zjz<<endl;

        outFile0<<endl;
        outFile0.close();
}
//-----------------------------------------------------------------
//                      block or interblock (ib,caltra01)
//-----------------------------------------------------------------
void caltra01(int nuca,int ib,int nl,int nr, int nc,int pnl,int pnr,int pnc,int nlmg,int nrmg,
                int ncmg,int ncpmfx,int ncpmfy,int ncpmfz, int ncpmix,int ncpmiy,int ncpmiz, float wlmf,
                float wrmf, float wcmf, vector <vector<vector<double> > >tmf,int ngra,int itt,
                vector <float> wcmg,vector<float> wrmg, vector<float> wlmg, vector<vector<int> > gra,
                vector <vector<short> > icon,int nlmf, short debug,int px)
{
        if (ib==1)
                genblo01(nl,nr,nc,pnl,pnr,pnc,nlmg,nrmg,ncmg,ncpmfx,ncpmfy,ncpmfz,ncpmix,ncpmiy,ncpmiz,
                        wlmf,wrmf,wcmf,tmf,gra,wcmg,wrmg,wlmg,ngra,itt,icon,nlmf,debug,px);
        if(nuca==1 && (ib==3 || ib==2))
                genblo02(nl,nr,nc,pnl,pnr,pnc,nlmg,nrmg,ncmg,ncpmfx,ncpmfy,ncpmfz,ncpmix,ncpmiy,ncpmiz,
                        wlmf,wrmf,wcmf,tmf,gra,wcmg,wrmg,wlmg,ngra,itt,icon,nlmf,debug,px);
        if(nuca==2 && (ib==3 || ib==2))
                genblo03(nl,nr,nc,pnl,pnr,pnc,nlmg,nrmg,ncmg,ncpmfx,ncpmfy,ncpmfz,ncpmix,ncpmiy,ncpmiz,
                        wlmf,wrmf,wcmf,tmf,gra,wcmg,wrmg,wlmg,ngra,itt,icon,nlmf,debug,px);
        if(nuca==3 && ib==3 && nlmf>1)
                genblo04(nl,nr,nc,pnl,pnr,pnc,nlmg,nrmg,ncmg,ncpmfx,ncpmfy,ncpmfz,ncpmix,ncpmiy,ncpmiz,
                        wlmf,wrmf,wcmf,tmf,gra,wcmg,wrmg,wlmg,ngra,itt,icon,nlmf,debug,px);
}
//-----------------------------------------------------------------
//                simple or diagonal or full(itt, caltra02)
//------------------------------------------------------------------
void caltra02(int nl,int nr,int nc,int itt,vector<vector<int> > gra,vector <vector<vector<double> > >tmi,
                float dz,float dy,float dx,int ngra,vector <vector<short> > icon,int ncpmix,int ncpmiy,
                int ncpmiz, vector <double> &tmb,short debug,int ix,int jy,int kz,int nrmg,int ncmg,
                int pnl,int pnr, int pnc,int nlmf,int px,int nunk)
{
        if (itt==11)
        {
                if(px==0)
                        medgeo(tmi, nl, nr, nc, ncpmix,ncpmiy,ncpmiz,tmb);//geometric mean
                if(px==1)
                        cuber(tmi,nl, nr, nc, ncpmix,ncpmiy,ncpmiz,tmb);//power lawer
                if(px==2)
                        analy(tmi,nl, nr, nc,ncpmix, ncpmiy, ncpmiz,tmb);
        }
        else if(itt==31||itt==32)
                tengen01(nl,nr,nc,itt,gra,tmi,dz,dy,dx,ngra,icon,ncpmix,ncpmiy,ncpmiz,tmb,debug,ix,jy,kz,
                        nrmg,ncmg,pnl,pnr,pnc,nlmf,nunk);
}
//-------------------------------------------------------------------------
//              read tmf into tmi and run caltra02(genblo01 for ib==1)
//-------------------------------------------------------------------------
void genblo01(int &nl,int &nr, int &nc,int &pnl,int &pnr,int &pnc,int nlmg,int nrmg, int ncmg,
                int ncpmfx, int ncpmfy, int ncpmfz,int ncpmix, int ncpmiy,int ncpmiz,float wlmf,
                float wrmf,float wcmf, vector <vector<vector<double> > >tmf, vector<vector<int> > gra,
                vector <float> wcmg, vector<float> wrmg,vector<float> wlmg, int ngra,int itt,
                vector <vector<short> > icon, int nlmf,short debug,int px)
{
  int ix,jy, kz,i,j,k,ig,jg,kg,nlmi,nrmi,ncmi,nunk;
  ix=0;jy=0;kz=0;
  vector <double> tmb(6,0);

  ofstream outFile1;
  outFile1.open("upscaled k in block.dat");
  outFile1<<"block conductivity full tensor"<<endl;
  if(itt==11)
  {
        nunk=1;
        outFile1<<nunk<<endl;
        outFile1<<"k"<<endl;
  }
  else if(itt==31 || itt==32)
  {
        if(nlmf>1)
        {
                nunk=6;
                outFile1<<nunk<<endl;
                outFile1<<"kxx"<<endl;
                outFile1<<"kxy"<<endl;
                outFile1<<"kxz"<<endl;
                outFile1<<"kyy"<<endl;
                outFile1<<"kyz"<<endl;
                outFile1<<"kzz"<<endl;
        }
        else
        {
                nunk=3;
                outFile1<<nunk<<endl;
                outFile1<<"kxx"<<endl;
                outFile1<<"kxy"<<endl;
                outFile1<<"kyy"<<endl;
        }
  }

  for (kg=0;kg<nlmg;kg++)
  {
    nlmi=wlmg[kg]/wlmf;
    nl=nlmi+2*ncpmiz;
    pnl=nl/2;

    for(ig=0;ig<nrmg;ig++)
    {
      nrmi=wrmg[ig]/wrmf;
      nr=nrmi+2*ncpmiy;
      pnr=nr/2;

      for(jg=0;jg<ncmg;jg++)
      {
        cout<<"layer"<<kg<<"  "<<"row"<<ig<<"  "<<"column"<<jg<<endl;
        ncmi=wcmg[jg]/wcmf;
        nc=ncmi+2*ncpmix;
        pnc=nc/2;

        vector <vector<vector<double> > > tmi_temp(nl, vector< vector<double> > (nr, vector<double>(nc,0)));

        for(k=kz;k<kz+nl;k++)
          for(i=ix;i<ix+nr;i++)
            for(j=jy;j<jy+nc;j++)
              tmi_temp[k-kz][i-ix][j-jy]=tmf[k+ncpmfz-ncpmiz][i+ncpmfy-ncpmiy][j+ncpmfx-ncpmix];

        vector <vector<vector<double> > > tmi(nl, vector< vector<double> > (nr, vector<double>(nc,0)) );
        for(k=nl-1;k>=0;k--)
          for(i=nr-1;i>=0;i--)
            for(j=0;j<nc;j++)
              tmi[k][i][j]=tmi_temp[nl-k-1][nr-i-1][j];

//--------check tmi-----------
        if (debug>2)
          Print(kg,ig,jg, nrmg, ncmg, nl,  nr, nc, tmi);

        caltra02(nl,nr, nc,itt,gra,tmi,wlmf,wrmf,wcmf,ngra,icon,ncpmix,ncpmiy,ncpmiz,tmb,debug,
                        ig,jg,kg,nrmg,ncmg,pnl,pnr,pnc,nlmf,px,nunk);

//------output block k (tmb)-----------
        for (i=0;i<nunk;i++)
```

```
          outFile1<<tmb[i]<<"  ";

//----------check positive definite---------------
        if(nlmf>1 && itt!=11)
        {
          if (tmb[0]<=0 || tmb[3]<=0 || tmb[5]<=0|| tmb[0]*tmb[3]-tmb[1]*tmb[1]<=0
                || tmb[0]*tmb[3]*tmb[5]-tmb[0]*tmb[4]*tmb[4]-tmb[1]*tmb[1]*tmb[5]+tmb[1]*tmb[2]*tmb[4]
                +tmb[2]*tmb[1]*tmb[4]-tmb[2]*tmb[2]*tmb[3]<=0)
          {
                cout<<"No_positive definite!!";
                system("pause");
          }
        }
        else if (nlmf==1 && itt!=11)
        {
          if (tmb[0]<=0 || tmb[2]<=0 || tmb[0]*tmb[2]-tmb[1]*tmb[1]<=0)
          {
                cout<<"No_positive definite!!";
                system("pause");
          }
        }
        outFile1<<endl;
        jy=jy+ncmi;
      }
      ix=ix+nrmi;
      jy=0;
    }
    kz=kz+nlmi;
    ix=0;
    jy=0;
  }
  outFile1.close();
  outFile1.clear();
}
//-----------------------------------------------------------------------------------------
//              read tmf into tmi and run caltra02(genblo02 for ib==2,3 and column)
//-----------------------------------------------------------------------------------------
void genblo02(int &nl,int &nr, int &nc,int &pnl,int &pnr,int &pnc,int nlmg,int nrmg, int ncmg,
                int ncpmfx, int ncpmfy, int ncpmfz,int ncpmix, int ncpmiy,int ncpmiz,float wlmf,
                float wrmf,float wcmf,vector <vector<vector<double> > >tmf, vector<vector<int> > gra,
                vector <float> wcmg,vector<float> wrmg,vector<float> wlmg, int ngra,int itt,
                vector <vector<short> > icon,int nlmf,short debug,int px)
{
  int ix,jy, kz,ipi,jpj,kpk,i,j,k,ig,jg,kg,nunk;
  ix=0;jy=0;kz=0;ipi=0;jpj=0;kpk=0;
  int nlmi,nrmi,ncmi1,ncmi2,ncmi;
  vector <double> tmb(6,0);

  ofstream outFile1;
  outFile1.open("upk betw col.dat");
  outFile1<<"block conductivity full tensor between columns"<<endl;
  if(itt==11)
  {
        nunk=1;
        outFile1<<nunk<<endl;
        outFile1<<"k"<<endl;
  }
  else if(itt==31 || itt==32)
  {
        if(nlmf>1)
        {
                nunk=6;
                outFile1<<"3"<<endl;
                outFile1<<"kxx"<<endl;
                outFile1<<"kxy"<<endl;
                outFile1<<"kxz"<<endl;
        }
        else
        {
                nunk=3;
                outFile1<<"2"<<endl;
                outFile1<<"kxx"<<endl;
                outFile1<<"kxy"<<endl;

        }
  }

  for (kg=0;kg<nlmg;kg++)
  {
    nlmi=wlmg[kg]/wlmf;
    nl=nlmi+2*ncpmiz;
    pnl=nl/2;
    kz=kpk;

    for(ig=0;ig<nrmg;ig++)
    {
      nrmi=wrmg[ig]/wrmf;
      nr=nrmi+2*ncpmiy;
      pnr=nr/2;
      ix=ipi;

      for(jg=0;jg<ncmg-1;jg++)
      {
        cout<<"layer"<<kg<<"  "<<"row"<<ig<<"  "<<"column"<<jg<<endl;
        ncmi1=wcmg[jg]/wcmf;
        ncmi2=wcmg[jg+1]/wcmf;
        ncmi=ncmi1-ncmi1/2+ncmi2-ncmi2/2;
        nc=ncmi+2*ncpmix;
        pnc=ncmi1-ncmi1/2+ncpmix;
        jy=jpj+ncmi1/2;

        vector <vector<vector<double> > > tmi_temp(nl, vector< vector<double> > (nr, vector<double>(nc,0)));

        for(k=kz;k<kz+nl;k++)
          for(i=ix;i<ix+nr;i++)
            for(j=jy;j<jy+nc;j++)
                tmi_temp[k-kz][i-ix][j-jy]=tmf[k+ncpmfz-ncpmiz][i+ncpmfy-ncpmiy][j+ncpmfx-ncpmix];

        vector <vector<vector<double> > > tmi(nl, vector< vector<double> > (nr, vector<double>(nc,0)));
        for(k=nl-1;k>=0;k--)
          for(i=nr-1;i>=0;i--)
            for(j=0;j<nc;j++)
                tmi[k][i][j]=tmi_temp[nl-k-1][nr-i-1][j];

//--------check tmi-----------
        if (debug>2)
            Print(kg,ig,jg, nrmg, ncmg, nl,  nr, nc, tmi);

        caltra02(nl,nr, nc,itt,gra,tmi,wlmf,wrmf,wcmf,ngra,icon,ncpmix,ncpmiy,ncpmiz,tmb,debug,
                        ig,jg,kg,nrmg,ncmg,pnl,pnr,pnc,nlmf,px,nunk);

//-------output inter-column conductivity---------
        if(itt==11)
          outFile1<<tmb[0];
        else if(itt==31 || itt==32)
        {
          if(nlmf>1)
            outFile1<<tmb[0]<<"  "<<tmb[1]<<"  "<<tmb[2]<<"  ";
          else
            outFile1<<tmb[0]<<"  "<<tmb[1]<<"  ";
        }
//----------check positive definite---------------
        if(nlmf>1 && itt!=11)
        {
          if (tmb[0]<=0 || tmb[3]<=0 || tmb[5]<=0|| tmb[0]*tmb[3]-tmb[1]*tmb[1]<=0
                || tmb[0]*tmb[3]*tmb[5]-tmb[0]*tmb[4]*tmb[4]-tmb[1]*tmb[1]*tmb[5]+tmb[1]*tmb[2]*tmb[4]
                +tmb[2]*tmb[1]*tmb[4]-tmb[2]*tmb[2]*tmb[3]<=0)
          {
                cout<<"No_positive definite!!";
                system("pause");
          }
        }
        else if (nlmf==1 && itt!=11)
        {
          if(tmb[0]<=0 || tmb[2]<=0 || tmb[0]*tmb[2]-tmb[1]*tmb[1]<=0)
          {
                cout<<"No_positive definite!!";
                system("pause");
          }
        }
        outFile1<<endl;
        jpj=jpj+ncmi1;
      }
      ipi=ipi+nrmi;
      jpj=0;
    }
    kpk=kpk+nlmi;
    ipi=0;
    jpj=0;
  }
  outFile1.close();
  outFile1.clear();
}
//-----------------------------------------------------------------------------------------
//                read tmf into tmi and run caltra02(genblo03 for ib==2,3 and row)
//-----------------------------------------------------------------------------------------
void genblo03(int &nl,int &nr, int &nc,int &pnl,int &pnr,int &pnc,int nlmg,int nrmg, int ncmg,
                int ncpmfx, int ncpmfy, int ncpmfz,int ncpmix, int ncpmiy,int ncpmiz,float wlmf,
                float wrmf,float wcmf,vector <vector<vector<double> > >tmf, vector<vector<int> > gra,
                vector <float> wcmg,vector<float> wrmg,vector<float> wlmg, int ngra,int itt,
                vector <vector<short> > icon,int nlmf,short debug,int px)
{
  int ix,jy, kz,ipi,jpj,kpk,i,j,k,ig,jg,kg,nunk;
  ix=0;jy=0;kz=0;ipi=0;jpj=0;kpk=0;
  int nlmi,nrmi,nrmi1,nrmi2,ncmi;
  vector <double> tmb(6,0);

  ofstream outFile1;
  outFile1.open("upk betw row.dat");
  outFile1<<"block conductivity full tensor between rows"<<endl;

  if(itt==11)
  {
    nunk=1;
    outFile1<<nunk<<endl;
    outFile1<<"k"<<endl;
  }
  else if(itt==31 || itt==32)
  {
    if(nlmf>1)
    {
        nunk=6;
        outFile1<<"3"<<endl;
        outFile1<<"kyx"<<endl;
        outFile1<<"kyy"<<endl;
        outFile1<<"kyz"<<endl;
    }
    else
    {
        nunk=3;
        outFile1<<"2"<<endl;
        outFile1<<"kyx"<<endl;
        outFile1<<"kyy"<<endl;
    }
  }

  for (kg=0;kg<nlmg;kg++)
  {
    nlmi=wlmg[kg]/wlmf;
    nl=nlmi+2*ncpmiz;
    pnl=nl/2;
    kz=kpk;

    for(ig=0;ig<nrmg-1;ig++)
    {
      nrmi1=wrmg[ig]/wrmf;
      nrmi2=wrmg[ig+1]/wrmf;
      nrmi=nrmi1-nrmi1/2+nrmi2-nrmi2/2;
      nr=nrmi+2*ncpmiy;
      pnr=nrmi1-nrmi1/2+ncpmiy;
      ix=ipi+nrmi1/2;

      for(jg=0;jg<ncmg;jg++)
      {
        cout<<"layer"<<kg<<"  "<<"row"<<ig<<"  "<<"column"<<jg<<endl;
        ncmi=wcmg[jg]/wcmf;
        nc=ncmi+2*ncpmix;
        pnc=nc/2;
        jy=jpj;

        vector <vector<vector<double> > > tmi_temp(nl, vector< vector<double> > (nr, vector<double>(nc,0)));

        for(k=kz;k<kz+nl;k++)
          for(i=ix;i<ix+nr;i++)
            for(j=jy;j<jy+nc;j++)
              tmi_temp[k-kz][i-ix][j-jy]=tmf[k+ncpmfz-ncpmiz][i+ncpmfy-ncpmiy][j+ncpmfx-ncpmix];

        vector <vector<vector<double> > > tmi(nl, vector< vector<double> > (nr, vector<double>(nc,0)));
        for(k=nl-1;k>=0;k--)
          for(i=nr-1;i>=0;i--)
```

```
            for(j=0;j<nc;j++)
                tmi[k][i][j]=tmi_temp[nl-k-1][nr-i-1][j];

//--------check tmi-----------
        if (debug>2)
            Print(kg,ig,jg, nrmg, ncmg, nl,  nr, nc, tmi);

        caltra02(nl,nr, nc,itt,gra,tmi,wlmf,wrmf,wcmf,ngra,icon,ncpmix,ncpmiy,ncpmiz,tmb,debug,
                        ig,jg,kg, nrmg,ncmg,pnl,pnr,pnc,nlmf,px,nunk);

//--------output inter-row conductivity-------------------
        if(itt==11)
            outFile1<<tmb[0];
        else if(itt==31 || itt==32)
        {
          if(nlmf>1)
            outFile1<<tmb[1]<<"  "<<tmb[3]<<"  "<<tmb[4]<<"  ";
          else
            outFile1<<tmb[1]<<"  "<<tmb[2]<<"  ";
        }

//----------check positive definite---------------
        if(nlmf>1 && itt!=11)
        {
          if (tmb[0]<=0 || tmb[3]<=0 || tmb[5]<=0|| tmb[0]*tmb[3]-tmb[1]*tmb[1]<=0
                || tmb[0]*tmb[3]*tmb[5]-tmb[0]*tmb[4]*tmb[4]-tmb[1]*tmb[1]*tmb[5]+tmb[1]*tmb[2]*tmb[4]
                +tmb[2]*tmb[1]*tmb[4]-tmb[2]*tmb[2]*tmb[3]<=0)
            {
                cout<<"No_positive definite!!";
                system("pause");
            }
        }
        else if (nlmf==1 && itt!=11)
        {
          if(tmb[0]<=0 || tmb[2]<=0 || tmb[0]*tmb[2]-tmb[1]*tmb[1]<=0)
           {
                cout<<"No_positive definite!!";
                system("pause");
           }
        }
        outFile1<<endl;
        jpj=jpj+ncmi;
      }
      ipi=ipi+nrmi1;
      jpj=0;
    }
    kpk=kpk+nlmi;
    ipi=0;
    jpj=0;
  }
  outFile1.close();
  outFile1.clear();
}
//-----------------------------------------------------------------------------------------
//              read tmf into tmi and run caltra02(genblo04 for ib==3 and layer)
//-----------------------------------------------------------------------------------------
void genblo04(int &nl,int &nr, int &nc,int &pnl,int &pnr,int &pnc,int nlmg,int nrmg, int ncmg,
               int ncpmfx, int ncpmfy, int ncpmfz,int ncpmix, int ncpmiy,int ncpmiz,float wlmf,
               float wrmf,float wcmf,vector <vector<vector<double> > >tmf, vector<vector<int> > gra,
               vector <float> wcmg,vector<float> wrmg,vector<float> wlmg, int ngra,int itt,
               vector <vector<short> > icon,int nlmf,short debug,int px)
{
  int ix,jy, kz,ipi,jpj,kpk,i,j,k,ig,jg,kg,nunk;
  ix=0;jy=0;kz=0;ipi=0;jpj=0;kpk=0;
  int nlmi,nlmi1,nlmi2,nrmi,ncmi;
  vector <double> tmb(6,0);

  ofstream outFile1;
  outFile1.open("upk betw lay.dat");
  outFile1<<"block conductivity full tensor between layers"<<endl;
  if(itt==11)
  {
        nunk=1;
        outFile1<<nunk<<endl;
        outFile1<<"k"<<endl;
  }
  else if(itt==31 || itt==32)
  {
        nunk=6;
        outFile1<<"3"<<endl;
        outFile1<<"kzx"<<endl;
        outFile1<<"kzy"<<endl;
        outFile1<<"kzz"<<endl;
  }

  for (kg=0;kg<nlmg-1;kg++)
  {
    nlmi1=wlmg[kg]/wlmf;
    nlmi2=wlmg[kg+1]/wlmf;
    nlmi=nlmi1-nlmi1/2+nlmi2-nlmi2/2;
    nl=nlmi+2*ncpmiz;
    pnl=nlmi1-nlmi1/2+ncpmiz;
    kz=kpk+nlmi1/2;

    for(ig=0;ig<nrmg;ig++)
    {
      nrmi=wrmg[ig]/wrmf;
      nr=nrmi+2*ncpmiy;
      pnr=nr/2;
      ix=ipi;

      for(jg=0;jg<ncmg;jg++)
      {
        cout<<"layer"<<kg<<"  "<<"row"<<ig<<"  "<<"column"<<jg<<endl;
        ncmi=wcmg[jg]/wcmf;
        nc=ncmi+2*ncpmix;
        pnc=nc/2;
        jy=jpj;

        vector <vector<vector<double> > > tmi_temp(nl, vector< vector<double> > (nr, vector<double>(nc,0)));

        for(k=kz;k<kz+nl;k++)
          for(i=ix;i<ix+nr;i++)
            for(j=jy;j<jy+nc;j++)
                tmi_temp[k-kz][i-ix][j-jy]=tmf[k+ncpmfz-ncpmiz][i+ncpmfy-ncpmiy][j+ncpmfx-ncpmix];

        vector <vector<vector<double> > > tmi(nl, vector< vector<double> > (nr, vector<double>(nc,0)));
        for(k=nl-1;k>=0;k--)
          for(i=nr-1;i>=0;i--)
            for(j=0;j<nc;j++)
                tmi[k][i][j]=tmi_temp[nl-k-1][nr-i-1][j];

//--------check tmi-----------
        if (debug>2)
            Print(kg,ig,jg, nrmg, ncmg, nl,  nr, nc, tmi);

        caltra02(nl,nr, nc,itt,gra,tmi,wlmf,wrmf,wcmf,ngra,icon,ncpmix,ncpmiy,ncpmiz,tmb,debug,
                        ig,jg,kg, nrmg,ncmg,pnl,pnr,pnc,nlmf,px,nunk);

//--------output inter-layer conductivity---------
        if(itt==11)
                outFile1<<tmb[0];
        else if(itt==31 || itt==32)
                outFile1<<tmb[2]<<"  "<<tmb[4]<<"  "<<tmb[5]<<"  ";

//----------check positive definite---------------
        if(nlmf>1 && itt!=11)
        {
          if (tmb[0]<=0 || tmb[3]<=0 || tmb[5]<=0|| tmb[0]*tmb[3]-tmb[1]*tmb[1]<=0
                || tmb[0]*tmb[3]*tmb[5]-tmb[0]*tmb[4]*tmb[4]-tmb[1]*tmb[1]*tmb[5]+tmb[1]*tmb[2]*tmb[4]
                +tmb[2]*tmb[1]*tmb[4]-tmb[2]*tmb[2]*tmb[3]<=0)
            {
                cout<<"No_positive definite!!";
                system("pause");
            }
        }
        else if (nlmf==1 && itt!=11)
        {
          if(tmb[0]<=0 || tmb[2]<=0 || tmb[0]*tmb[2]-tmb[1]*tmb[1]<=0)
          {
                cout<<"No_positive definite!!";
                system("pause");
          }
        }
        outFile1<<endl;
        jpj=jpj+ncmi;
      }
      ipi=ipi+nrmi;
      jpj=0;
    }
    kpk=kpk+nlmi1;
    ipi=0;
    jpj=0;
  }
  outFile1.close();
}
//--------------------------------------------------------------------------------
//                                  one block (tengen01)
//---------------------------------------------------------------------------------
void tengen01(int nl,int nr,int nc,int itt,vector<vector<int> > gra,vector <vector<vector<double> > >tmi,
                float dz,float dy,float dx,int ngra,vector <vector<short> > icon,int ncpmix,int ncpmiy,
                int ncpmiz, vector <double> &tmb,short debug,int ix,int jy,int kz,int nrmg,int ncmg,
                int pnl,int pnr, int pnc, int nlmf,int nunk)
{
        int iflag,neq,kount;
        iflag=1;
        neq=0;
        int mne,mnunk;
        mne=50;mnunk=10;
        vector <double> rhs2(mne,0);
        vector <vector <double> >coeff(mne,vector <double>(mnunk,0));

        vector <vector<vector<double> > > cc(nl, vector< vector<double> > (nr, vector<double>(nc,0)));
        vector <vector<vector<double> > > cr(nl, vector< vector<double> > (nr, vector<double>(nc,0)));
        vector <vector<vector<double> > > cv(nl, vector< vector<double> > (nr, vector<double>(nc,0)));
        vector <double> ccc(nl*nr*nc,0);
        vector <double> ccr(nl*nr*nc,0);
        vector <double> ccv(nl*nr*nc,0);
//-------------calculate conductance--------
        Cond(nl,nr,nc,iflag,tmi,cc,cr,cv,ccc,ccr,ccv,dz,dy,dx,nlmf);
//------------check cc,cv,cr------------
        if(debug>3)
        {
                Printc(nl,nr,nc,kz, ix,jy,nrmg, ncmg,cc);
                Printc(nl,nr,nc,kz, ix,jy,nrmg, ncmg,cr);
                Printc(nl,nr,nc,kz, ix,jy,nrmg, ncmg,cv);
        }

        for(kount=0;kount<ngra;kount++)
        {
                double xjx,yjy,zjz;
                int i,j,k,pun1,pun2;

                xjx=gra[kount][0];
                yjy=gra[kount][1];
                zjz=gra[kount][2];
//-------------initial heads and boundary condition----------
                vector <double> hcof(nl*nr*nc,0);
                vector <double> rhs(nl*nr*nc,0);
                vector <double> hnew(nl*nr*nc,0);
                vector <short> ibound(nl*nr*nc,0);

                vector <vector<vector<double> > > hh(nl, vector< vector<double> > (nr, vector<double>(nc,0)));

                for(k=0;k<nl;k++)
                        for(i=0;i<nr;i++)
                                for(j=0;j<nc;j++)
                                {
                                        pun1=k*nr*nc+i*nc+j;
                                        hcof[pun1]=0;
                                        rhs[pun1]=0;
                                        hnew[pun1]=j*dx*xjx+i*dy*yjy+k*dz*zjz;
                                        ibound[pun1]=1;
                                }

                for(k=0;k<nl;k++)
                        for(i=0;i<nr;i++)
                        {
                                pun1=k*nr*nc+i*nc;
                                pun2=k*nr*nc+i*nc+nc-1;
                                ibound[pun1]=icon[kount][0];
                                ibound[pun2]=icon[kount][1];
                        }

                for(k=0;k<nl;k++)
                        for(j=0;j<nc;j++)
                        {
                                pun1=k*nr*nc+j;
                                pun2=k*nr*nc+(nr-1)*nc+j;
                                ibound[pun1]=icon[kount][2];
```

```
                                        ibound[pun2]=icon[kount][3];
                                }

                if(nlmf>1)
                {
                        for(i=0;i<nr;i++)
                                for(j=0;j<nc;j++)
                                {
                                        pun1=i*nc+j;
                                        pun2=(nl-1)*nr*nc+i*nc+j;
                                        ibound[pun1]=icon[kount][4];
                                        ibound[pun2]=icon[kount][5];
                                }
                }

//--------------solver PCG2---------------
                PCG2(nl, nr,nc,ibound, ccr, ccc,ccv, hcof, rhs, hnew, hh);//sequence of ccc and ccr

//--------check piezometric head------------
                if (debug>1)
                        Printh(kount,nl,nr,nc,kz,ix,jy,hh,nrmg,ncmg);

//--------calculate flux in one block or inter-block---------------
                double qx,qy,qz;
                qx=0;qy=0;qz=0;
                if(itt==31 && nlmf==1)
                        Flumed01(nr,nc,cc, cr, dx,dy,ncpmix,ncpmiy,qx,qy,xjx,yjy,hh);
                if(itt==32 && nlmf==1)
                        Flumed02(nr, nc, cc,cr,dx,dy,pnr,pnc,ncpmix,ncpmiy,qx,qy,xjx,yjy, hh);
                if(itt==31 && nlmf>1)
                        Flumed03(nl,nr,nc,cc,cr,cv,dx,dy,dz,qx,qy,qz,xjx,yjy,zjz,hh,ncpmix,ncpmiy,ncpmiz);
                if(itt==32 && nlmf>1)
                        Flumed04(nl,nr,nc,cc,cr,cv,dx,dy,dz,qx,qy,qz,xjx,yjy,zjz,hh,ncpmix,ncpmiy,ncpmiz,
                                pnl, pnr, pnc);

//--------check flux----------------------
                if (debug>1 && nlmf>1)
                        Printflux(kount,nl,nr,nc,kz, ix,jy,qx,qy,qz,xjx,yjy,zjz, nrmg, ncmg);

//--------coefficient matrix for overdeter-------------
                if(nlmf>1)
                {
                        neq+=3;
                        rhs2[neq-3]=qx;
                        rhs2[neq-2]=qy;
                        rhs2[neq-1]=qz;
                        coeff[neq-3][0]=-xjx;
                        coeff[neq-3][1]=-yjy;
                        coeff[neq-3][2]=-zjz;
                        coeff[neq-3][3]=0;
                        coeff[neq-3][4]=0;
                        coeff[neq-3][5]=0;

                        coeff[neq-2][0]=0;
                        coeff[neq-2][1]=-xjx;
                        coeff[neq-2][2]=0;
                        coeff[neq-2][3]=-yjy;
                        coeff[neq-2][4]=-zjz;
                        coeff[neq-2][5]=0;

                        coeff[neq-1][0]=0;
                        coeff[neq-1][1]=0;
                        coeff[neq-1][2]=-xjx;
                        coeff[neq-1][3]=0;
                        coeff[neq-1][4]=-yjy;
                        coeff[neq-1][5]=-zjz;
                }
                else
                {
                        neq+=2;
                        rhs2[neq-2]=qx;
                        rhs2[neq-1]=qy;
                        coeff[neq-2][0]=-xjx;
                        coeff[neq-2][1]=-yjy;
                        coeff[neq-2][2]=0;
                        coeff[neq-1][0]=0;
                        coeff[neq-1][1]=-xjx;
                        coeff[neq-1][2]=-yjy;
                }
```

```
        }
        Overdet(coeff,rhs2,tmb,nunk,neq,mne);
}
//-------------------------------------
//            solver PCG2 (partly borrowed from the "pcg2.f" in MODFLOW by USGS)
//--------------------------------------
void PCG2(int nl,int nr,int nc,vector <short> ibound,vector <double> ccc,vector <double> ccr,
            vector <double> ccv, vector <double> hcof,vector <double> rhs, vector <double> &hnew,
            vector <vector<vector<double> > > &hh)
{
        const short norm=0;
        const int mxiter=1;
        const int iter=999;
        const int npcond=1;
        const double hclose=0.000000001;
        const double rclose=0.000000001;
        const double relax=0.98;
        const double damp=1.0;

        int kiter,nrc,n,i, j,k,ii,jj,kk,nrn,nrl,ncn,ncl,nln,nll,
                ncf,ncd,nrb,nrh,nls,nlz,iiter,niter,icnvg,iicnvg,
                Nc,Nr,Nl,ic,ir,il,Nh,nodes;
        float bigh, bigr,cd1;
        double biggestpos,biggestneg,srnew,srold,dzero,done,del,
                        bhnew,hhnew,dhnew,fhnew,zhnew,shnew,hchgn,rchgn,
                        B,H,D,F,Z,S,E,sscr,sscc,sscv,vn,pn,pap,alpha,
                        rrhs,hhcof,vcc,vcr,vcv,cdcr,cdcc,cdcv,fcc,fcr,fcv,fv;
        vector <double> ss(nl*nr*nc,0);
        vector <double> p(nl*nr*nc,0);
        vector <double> v(nl*nr*nc,0);
        vector <double> cd(nl*nr*nc,0);
        vector <double> hpcg(nl*nr*nc,0);
        vector <double> res(nl*nr*nc,0);

        for (kiter=0;kiter<mxiter;kiter++)
        {
                bigh=1;
                biggestpos=3.4e+038;
                biggestneg=-biggestpos;

                nrc=nc*nr;
                nodes=nl*nr*nc;
                dzero=0;
                done=1;
                srnew=dzero;
                cd1=0;
                for (n=0; n<nodes;n++)
                {
                        ss[n]=0;
                        p[0]=0;
                        v[n]=0;
                        cd[n]=0;
                        if (mxiter>1)
                                hpcg[n]=hnew[n];
                }
                for (k=0;k<nl;k++)
                {
                        for (i=0;i<nr;i++)
                        {
                                for (j=0;j<nc;j++)
                                {
                                        n=j+i*nc+k*nrc;
                                        if(ibound[n]==0)
                                        {
                                                ccc[n]=0;
                                                ccr[n]=0;
                                                if(n<(nodes-nrc))
                                                        ccv[n]=0;
                                                if(n>0)
                                                        ccr[n-1]=0;
                                                if(n>(nc-1))
                                                        ccc[n-nc]=0;
                                                if(n<(nodes-nrc) && n>(nrc-1))
                                                        ccv[n-nrc]=0;
                                                hcof[n]=0;
                                                rhs[n]=0;
                                                continue;
                                        }
                                        nrn=n+nc;
                                        nrl=n-nc;
                                        ncn=n+1;
                                        ncl=n-1;
                                        nln=n+nrc;
                                        nll=n-nrc;
                                        ncf=n;
                                        ncd=n-1;
                                        nrb=n-nc;
                                        nrh=n;
                                        nls=n;
                                        nlz=n-nrc;
                                        B=dzero;
                                        bhnew=dzero;
                                        if (i!=0)
                                        {
                                                B=ccc[nrb];
                                                bhnew=B*(hnew[nrl]-hnew[n]);
                                        }
                                        H=dzero;
                                        hhnew=dzero;
                                        if(i!=(nr-1))
                                        {
                                                H=ccc[nrh];
                                                hhnew=H*(hnew[nrn]-hnew[n]);
                                        }
                                        D=dzero;
                                        dhnew=dzero;
                                        if(j!=0)
                                        {
                                                D=ccr[ncd];
                                                dhnew=D*(hnew[ncl]-hnew[n]);
                                        }
                                        F=dzero;
                                        fhnew=dzero;
                                        if(j!=(nc-1))
                                        {
                                                F=ccr[ncf];
                                                fhnew=F*(hnew[ncn]-hnew[n]);
                                        }
                                        Z=dzero;
                                        zhnew=dzero;
                                        if(k!=0)
                                        {
                                                Z=ccv[nlz];
                                                zhnew=Z*(hnew[nll]-hnew[n]);
                                        }
                                        S=dzero;
                                        shnew=dzero;
                                        if(k!=(nl-1))
                                        {
                                                S=ccv[nls];
                                                shnew=S*(hnew[nln]-hnew[n]);
                                        }
                                        if (i==nr-1)
                                                ccc[n]=0;
                                        if (j==nc-1)
                                                ccr[n]=0;
                                        if (B+H+D+F+Z+S==0)
                                        {
                                                ibound[n]=0;
                                                hcof[n]=0;
                                                rhs[n]=0;
                                                continue;
                                        }
                                        E=-Z-B-D-F-H-S;
                                        rrhs=rhs[n];
                                        hhcof=hnew[n]*hcof[n];
                                        res[n]=rrhs-zhnew-bhnew-dhnew-hhcof-fhnew-hhnew-shnew;
                                        if (ibound[n]<0)
                                                res[n]=0;
                                }
                        }
                }
                iiter=0;
                if(kiter==0) niter=0;
                icnvg=0;
                iicnvg=0;
                for (iiter=1;iiter<=iter;iiter++)
                {
                        if (icnvg==0||iicnvg==0)
                        {
```

```
bigh=0;
bigr=0;
del=0;
for (k=0;k<nl;k++)
{
  for (i=0;i<nr;i++)
  {
    for (j=0;j<nc;j++)
    {
        n=j+i*nc+k*nrc;
        if (ibound[n]<1) continue;
        H=dzero;
        vcc=dzero;
        ic=n-nc;
        if (i!=0)
        {
                H=ccc[ic];
                if (cd[ic]!=0)
                        vcc=H*v[ic]/cd[ic];
        }
        F=dzero;
        vcr=dzero;
        ir=n-1;
        if (j!=0)
        {
                F=ccr[ir];
                if (cd[ir]!=0)
                        vcr=F*v[ir]/cd[ir];
        }
        S=dzero;
        vcv=dzero;
        il=n-nrc;
        if(k!=0)
        {
                S=ccv[il];
                if(cd[il]!=0)
                        vcv=S*v[il]/cd[il];
        }
        v[n]=res[n]-vcr-vcc-vcv;
        if (iiter==1)
        {
                cdcr=dzero;
                cdcc=dzero;
                cdcv=dzero;
                fcc=dzero;
                fcr=dzero;
                fcv=dzero;
                if (ir>=0)
                {
                  if (cd[ir]!=0)
                     cdcr=pow(F,2)/cd[ir];
                }
                if (ic>=0)
                {
                  if (cd[ic]!=0)
                     cdcc=pow(H,2)/cd[ic];
                }
                if (il>=0)
                {
                  if (cd[il]!=0)
                     cdcv=pow(S,2)/cd[il];
                }
                if (npcond==1)
                {
                  if (ir>=0)
                  {
                        fv=ccv[ir];
                        if (k==(nl-1) && (j+i)>1)
                            fv=dzero;
                        if (cd[ir]!=0)
                            fcr=(F/cd[ir])*(ccc[ir]+fv);
                  }
                  if (ic>=0)
                  {
                        fv=ccv[ic];
                        if (k==nl-1 && i>0)
                            fv=dzero;
                        if (cd[ic]!=0)
                            fcc=(H/cd[ic])*(ccr[ic]+fv);
                  }
                  if (il>=0)
                  {
                        if(cd[il]!=0)
                            fcv=(S/cd[il])*(ccr[il]+ccc[il]);
                  }
                }
                if (norm ==0)
                {
                        B=dzero;
                        H=dzero;
                        D=dzero;
                        F=dzero;
                        Z=dzero;
                        S=dzero;
                        if (i!=0)
                                B=ccc[ic];
                        if (i!=(nr-1))
                                H=ccc[n];
                        if (j!=0)
                                D=ccr[ir];
                        if (j!=(nc-1))
                                F=ccr[n];
                        if (k!=0)
                                Z=ccv[il];
                        if (k!=(nl-1))
                                S=ccv[n];
                        hhcof=hcof[n]-Z-B-D-F-H-S;
                }
                cd[n]=(done+del)*hhcof-cdcr-cdcc-cdcv-relax*(fcr+fcc+fcv);
                if (cd1==0 && cd[n]!=0)
                        cd1=cd[n];
                if (cd[n]*cd1<=0)
                {
                        del=1.5*del+0.001;
                        if (del>0.5)
                        break;
                }
        }
    }
  }
}
for (kk=(nl-1);kk>=0;kk--)
{
        for (ii=(nr-1);ii>=0;ii--)
        {
                for(jj=(nc-1);jj>=0;jj--)
                {
                        n=jj+ii*nc+kk*nrc;
                        if (ibound[n]<1)
                                continue;
                        Nc=n+1;
                        Nr=n+nc;
                        Nl=n+nrc;
                        sscr=dzero;
                        sscc=dzero;
                        sscv=dzero;
                        if (jj != (nc-1))
                                sscr=ccr[n]*ss[Nc]/cd[n];
                        if (ii!=(nr-1))
                                sscc=ccc[n]*ss[Nr]/cd[n];
                        if (kk!=(nl-1))
                                sscv=ccv[n]*ss[Nl]/cd[n];
                        vn=v[n]/cd[n];
                        ss[n]=vn-sscr-sscc-sscv;
                }
        }
}
srold= srnew;
srnew=dzero;
for (n=0;n<nodes;n++)
{
        if (ibound[n]>0)
                srnew=srnew+ss[n]*res[n];
}
if (iiter==1)
{
        for (n=0;n<nodes;n++)
                p[n]=ss[n];
}
else
```

```
                                {
                                        for (n=0;n<nodes;n++)
                                                p[n]=ss[n]+(srnew/srold)*p[n];
                                }
                                pap=dzero;
                                for (k=0;k<nl;k++)
                                {
                                        for (i=0;i<nr;i++)
                                        {
                                                for (j=0;j<nc;j++)
                                                {
                                                        n=j+i*nc+k*nrc;
                                                        v[n]=0;
                                                        if (ibound[n]<1)
                                                                continue;
                                                        nrn=n+nc;
                                                        nrl=n-nc;
                                                        ncn=n+1;
                                                        ncl=n-1;
                                                        nln=n+nrc;
                                                        nll=n-nrc;
                                                        ncf=n;
                                                        ncd=ncl;
                                                        nrb=nrl;
                                                        nrh=n;
                                                        nls=n;
                                                        nlz=nll;
                                                        B=dzero;
                                                        if (i!=0)
                                                                B=ccc[nrb];
                                                        H=dzero;
                                                        if(i!=(nr-1))
                                                                H=ccc[nrh];
                                                        D=dzero;
                                                        if (j!=0)
                                                                D=ccr[ncd];
                                                        F=dzero;
                                                        if (j!=(nc-1))
                                                                F=ccr[ncf];
                                                        Z=dzero;
                                                        if (k!=0)
                                                                Z=ccv[nlz];
                                                        S=dzero;
                                                        if (k!=(nl-1))
                                                                S=ccv[nls];
                                                        if (norm==0)
                                                                pn=p[n];
                                                        bhnew=dzero;
                                                        hhnew=dzero;
                                                        dhnew=dzero;
                                                        fhnew=dzero;
                                                        zhnew=dzero;
                                                        shnew=dzero;
                                                        if (nrl>=0)
                                                                bhnew=B*(p[nrl]-pn);
                                                        if (nrn<nodes)
                                                                hhnew=H*(p[nrn]-pn);
                                                        if (ncl>=0)
                                                                dhnew=D*(p[ncl]-pn);
                                                        if (ncn<nodes)
                                                                fhnew=F*(p[ncn]-pn);
                                                        if (nll>=0)
                                                                zhnew=Z*(p[nll]-pn);
                                                        if (nln<nodes)
                                                                shnew=S*(p[nln]-pn);
                                                        pn=hcof[n]*p[n];
                                                        vn=zhnew+bhnew+dhnew+pn+fhnew+hhnew+shnew;
                                                        v[n]=vn;
                                                        pap=pap+p[n]*vn;
                                                }
                                        }
                                }
//------------calculate alpha-----------------------
                                alpha=1;
                                if (pap==0 && mxiter ==1)
                                {
                                        cout<<"conjugate gradient method failed"<<endl;
                                        cout<<"set mxiter greater than 1 and try again";
                                }
                                if (pap!=0)
```

```
                                        alpha=srnew/pap;
//-----------calculate new heads and residuals--------
                                for (k=0;k<nl;k++)
                                {
                                  for (i=0;i<nr;i++)
                                  {
                                    for (j=0;j<nc;j++)
                                    {
                                        n=j+i*nc+k*nrc;
                                        if (ibound[n]<1)
                                                continue;
                                        hchgn=alpha*p[n];
                                        if ( fabs(hchgn)> fabsf (bigh))
                                        {
                                                if (hchgn < biggestpos && hchgn>biggestneg)
                                                        bigh=hchgn;
                                                else
                                                {
                                                        if (hchgn>0)
                                                                bigh=0.9999*biggestpos;
                                                        else
                                                                bigh=0.9999*biggestneg;
                                                }
                                                Nh=n;
                                        }
                                        hnew[n]=hnew[n]+hchgn;
                                        rchgn=-alpha*v[n];
                                        res[n]=res[n]+rchgn;
                                        if(fabs(res[n])> fabs(bigr))
                                        {
                                                bigr=res[n];
                                                Nr=n;
                                        }
                                    }
                                  }
                                }
//---------unscale largest change in head and residual, and check the convergence criterion
                                bigh=bigh*damp;
                                bigr=bigr*damp;
                                if (mxiter==1)
                                {
                                        if (fabs(bigh)<=hclose && fabs(bigr)<=rclose)
                                                icnvg=1;
                                }
                                else
                                {
                                        if(iiter==1 && fabs(bigh)<=hclose &&  fabs(bigr)<= rclose)
                                                icnvg=1;
                                }
                                if (fabs(bigh)<=hclose && fabs(bigr)<= rclose)
                                        iicnvg=1;
//--------store the lagrest unscaled head change and residual and their locations
                                ii=niter-1;
                        }
                }
                if(mxiter>1)
                {
                        for (n=0;n<nodes;n++)
                        {
                                if (ibound[n]<=0)
                                        continue;
                                hnew[n]=(done-damp)*hpcg[n]+damp*hnew[n];
                        }
                }
        }
        for (k=0;k<nl;k++)
        {
                for (i=0;i<nr;i++)
                {
                        for (j=0;j<nc;j++)
                        {
                                n=j+i*nc+k*nrc;
                                hh[k][i][j]=hnew[n];
                        }
                }
        }
}
//---------------------------------------------------------//
//                      flumed01 (2D inblock)              //
//---------------------------------------------------------//
```

```
void Flumed01(int nr,int nc,vector <vector<vector<double> > > cc,vector <vector<vector<double> > > cr,
                float dx,float dy,int ncpx,int ncpy,double &qx,double &qy,double &xjx,double &yjy,
                vector <vector<vector<double> > > hh)
{
        int i,j,k;
        k=0;

        for(i=ncpy+1;i<nr-ncpy-1;i++)
                for(j=ncpx;j<nc-ncpx-1;j++)
                        qx=qx-cc[k][i][j]*(hh[k][i][j+1]-hh[k][i][j]);

        qx=qx/(nc-2*ncpx-1)/(nr-2*ncpy-2)/dy;

        for(i=ncpy;i<nr-ncpy-1;i++)
                for(j=ncpx+1;j<nc-ncpx-1;j++)
                        qy=qy-cr[k][i][j]*(hh[k][i+1][j]-hh[k][i][j]);

        qy=qy/(nr-2*ncpy-1)/(nc-2*ncpx-2)/dx;

        xjx=0;
        for(i=ncpy+1;i<nr-ncpy-1;i++)
                for(j=ncpx;j<nc-ncpx-1;j++)
                        xjx=xjx+hh[k][i][j+1]-hh[k][i][j];

        xjx=xjx/(nc-2*ncpx-1)/dx/(nr-2*ncpy-2);

        yjy=0;
        for(i=ncpy;i<nr-ncpy-1;i++)
                for(j=ncpx+1;j<nc-ncpx-1;j++)
                        yjy=yjy+hh[k][i+1][j]-hh[k][i][j];

        yjy=yjy/(nr-2*ncpy-1)/(nc-2*ncpx-2)/dy;

}
//--------------------------------------------------------//
//                      flumed02 (2D interface)
//---------------------------------------------------------//
void Flumed02(int nr,int nc,vector <vector<vector<double> > > cc,vector <vector<vector<double> > > cr,
                float dx,float dy, int pnr,int pnc,int ncpx,int ncpy,double &qx,double &qy,double &xjx,
                double &yjy, vector <vector<vector<double> > > hh)
{
        int i,j,k,ic;
        double p1x,p2x,d12,p1y,p2y;
        k=0;

        qx=0;
        for(i=ncpy;i<nr-ncpy;i++)
                qx=qx-cc[k][i][pnc-1]*(hh[k][i][pnc]-hh[k][i][pnc-1]);
        qx=qx/(nr-2*ncpy)/dy;

        qy=0;
        for(j=ncpx;j<nc-ncpx;j++)
                qy=qy-cr[k][pnr-1][j]*(hh[k][pnr][j]-hh[k][pnr-1][j]);
        qy=qy/(nc-2*ncpx)/dx;

        int ii1,ii2,jj1,jj2;
        p1x=0;
        ic=0;
        ii1=pnc-ncpx;
        if (ii1>=ncpx)
        ii1=ncpx;
        for(i=ncpy;i<nr-ncpy;i++)
                for(j=ncpx-ii1;j<pnc;j++)
                {
                        p1x=p1x+hh[k][i][j];
                        ic+=1;
                }
        p1x=p1x/ic;

        p2x=0;
        ic=0;
        ii2=nc-ncpx-pnc;
        if(ii2>=ncpx)
        ii2=ncpx;
        for(i=ncpy;i<nr-ncpy;i++)
                for(j=pnc;j<nc-ncpx+ii2;j++)
                {
                        p2x=p2x+hh[k][i][j];
                        ic+=1;
                }
        p2x=p2x/ic;

        d12=dx*(double(pnc-(ncpx-ii1))/2+double(nc-ncpx+ii2-pnc)/2);// modified August 7, 2009
        xjx=(p2x-p1x)/d12;

        p1y=0;
        ic=0;
        jj1=pnr-ncpy;
        if(jj1>=ncpy)
                jj1=ncpy;
        for(i=ncpy-jj1;i<pnr;i++)
                for(j=ncpx;j<nc-ncpx;j++)
                {
                        p1y=p1y+hh[k][i][j];
                        ic+=1;
                }
        p1y=p1y/ic;

        p2y=0;
        ic=0;
        jj2=nr-ncpy-pnr;
        if(jj2>=ncpy)
                jj2=ncpy;
        for(i=pnr;i<nr-ncpy+jj2;i++)
                for(j=ncpx;j<nc-ncpx;j++)
                {
                        p2y=p2y+hh[k][i][j];
                        ic+=1;
                }
        p2y=p2y/ic;

        d12=dy*(double(pnr-(ncpy-jj1))/2+double(nr-ncpy+jj2-pnr)/2);// modified August 7, 2009
        yjy=(p2y-p1y)/d12;

}
//--------------------------------------------------------
//              populate flux in-block (Flumed03)
//---------------------------------------------------------
void Flumed03(int nl,int nr,int nc,vector <vector<vector<double> > > cc,
                vector <vector<vector<double> > > cr,vector <vector<vector<double> > > cv,float dx,
                float dy, float dz,double &qx,double &qy, double &qz, double &xjx,double &yjy,double &zjz,
                vector <vector<vector<double> > > hh,int ncpx,int ncpy,int ncpz)
{
        int i,j,k;

        for(k=ncpz+1;k<nl-ncpz-1;k++)
                for(i=ncpy+1;i<nr-ncpy-1;i++)
                        for(j=ncpx;j<nc-ncpx-1;j++)
                                qx=qx-cc[k][i][j]*(hh[k][i][j+1]-hh[k][i][j]);

        qx=qx/(nl-2*ncpz-2)/(nr-2*ncpy-2)/(nc-2*ncpx-1)/dy/dz;

        for(k=ncpz+1;k<nl-ncpz-1;k++)
                for(i=ncpy;i<nr-ncpy-1;i++)
                        for(j=ncpx+1;j<nc-ncpx-1;j++)
                                qy=qy-cr[k][i][j]*(hh[k][i+1][j]-hh[k][i][j]);

        qy=qy/(nl-2*ncpz-2)/(nr-2*ncpy-1)/(nc-2*ncpx-2)/dx/dz;

        for(k=ncpz;k<nl-ncpz-1;k++)
                for(i=ncpy+1;i<nr-ncpy-1;i++)
                        for(j=ncpx+1;j<nc-ncpx-1;j++)
                                qz=qz-cv[k][i][j]*(hh[k+1][i][j]-hh[k][i][j]);

        qz=qz/(nl-2*ncpz-1)/(nr-2*ncpy-2)/(nc-2*ncpx-2)/dx/dy;

        xjx=0;
        for(k=ncpz+1;k<nl-ncpz-1;k++)
                for(i=ncpy+1;i<nr-ncpy-1;i++)
                        for(j=ncpx;j<nc-ncpx-1;j++)
                                xjx=xjx+hh[k][i][j+1]-hh[k][i][j];

        xjx=xjx/(nl-2*ncpz-2)/(nr-2*ncpy-2)/(nc-2*ncpx-1)/dx;

        yjy=0;
        for(k=ncpz+1;k<nl-ncpz-1;k++)
                for(i=ncpy;i<nr-ncpy-1;i++)
                        for(j=ncpx+1;j<nc-ncpx-1;j++)
                                yjy=yjy+hh[k][i+1][j]-hh[k][i][j];
        yjy=yjy/(nl-2*ncpz-2)/(nr-2*ncpy-1)/(nc-2*ncpx-2)/dy;

        zjz=0;
        for(k=ncpz;k<nl-ncpz-1;k++)
                for(i=ncpy+1;i<nr-ncpy-1;i++)
                        for(j=ncpx+1;j<nc-ncpx-1;j++)
                                zjz=zjz+hh[k+1][i][j]-hh[k][i][j];

        zjz=zjz/(nl-2*ncpz-1)/(nr-2*ncpy-2)/(nc-2*ncpx-2)/dz;

}
//-----------------------------------------------------
//              populate flux inter-block(Flumed04)
//------------------------------------------------------
void Flumed04(int nl,int nr,int nc,vector <vector<vector<double> > > cc,vector <vector<vector<double> > > cr,
                vector <vector<vector<double> > > cv,float dx,float dy, float dz,double &qx,double &qy,
                double &qz, double &xjx,double &yjy,double &zjz,vector <vector<vector<double> > > hh,
                int ncpx,int ncpy, int ncpz, int pnl,int pnr,int pnc)
{
        int i,j,k,ic,ii1,ii2,jj1,jj2,kk1,kk2;
        double p1x,p2x,p1y,p2y,p1z,p2z,d12;
        d12=0;

        for (k=ncpz;k<nl-ncpz;k++)
                for(i=ncpy;i<nr-ncpy;i++)
                        qx=qx-cc[k][i][pnc-1]*(hh[k][i][pnc]-hh[k][i][pnc-1]);

        qx=qx/(nl-2*ncpz)/(nr-2*ncpy)/dy/dz;

        for (k=ncpz;k<nl-ncpz;k++)
                for(j=ncpx;j<nc-ncpx;j++)
                        qy=qy-cr[k][pnr-1][j]*(hh[k][pnr][j]-hh[k][pnr-1][j]);

        qy=qy/(nl-2*ncpz)/(nc-2*ncpx)/dx/dz;

        for(i=ncpy;i<nr-ncpy;i++)
                for(j=ncpx;j<nc-ncpx;j++)
                        qz=qz-cv[pnl-1][i][j]*(hh[pnl][i][j]-hh[pnl-1][i][j]);
        qz=qz/(nr-2*ncpy)/(nc-2*ncpx)/dx/dy;

        ic=0;
        p1x=0;
        ii1=pnc-ncpx;
        if (ii1>=ncpx)
                ii1=ncpx;
        for(k=ncpz;k<nl-ncpz;k++)
                for(i=ncpy;i<nr-ncpy;i++)
                        for(j=ncpx-ii1;j<pnc;j++)
                        {
                                p1x=p1x+hh[k][i][j];
                                ic+=1;
                        }
        p1x=p1x/ic;

        ic=0;
        p2x=0;
        ii2=nc-ncpx-pnc;
        if(ii2>=ncpx)
                ii2=ncpx;
        for(k=ncpz;k<nl-ncpz;k++)
                for(i=ncpy;i<nr-ncpy;i++)
                        for(j=pnc;j<nc-ncpx+ii2;j++)
                        {
                                p2x=p2x+hh[k][i][j];
                                ic+=1;
                        }
        p2x=p2x/ic;

        d12=dx*(double(pnc-(ncpx-ii1))/2+double(nc-ncpx+ii2-pnc)/2);// modified August 7, 2009
        xjx=(p2x-p1x)/d12;

        ic=0;
        p1y=0;
        jj1=pnr-ncpy;
        if (jj1>=ncpy)
                jj1=ncpy;
        for(k=ncpz;k<nl-ncpz;k++)
                for(i=ncpy-jj1;i<pnr;i++)
                        for(j=ncpx;j<nc-ncpx;j++)
                        {
                                p1y=p1y+hh[k][i][j];
                                ic+=1;
                        }
        p1y=p1y/ic;

        ic=0;
        p2y=0;
        jj2=nr-ncpy-pnr;
        if(jj2>=ncpy)
                jj2=ncpy;
        for(k=ncpz;k<nl-ncpz;k++)
                for(i=pnr;i<nr-ncpy+jj2;i++)
                        for(j=ncpx;j<nc-ncpx;j++)
                        {
                                p2y=p2y+hh[k][i][j];
                                ic+=1;
                        }
        p2y=p2y/ic;

        d12=dy*(double(pnr-(ncpy-jj1))/2+double(nr-ncpy+jj2-pnr)/2);// modified August 7, 2009
        yjy=(p2y-p1y)/d12;

        ic=0;
        p1z=0;
        kk1=pnl-ncpz;
        if(kk1>=ncpz)
                kk1=ncpz;
        for(k=ncpz-kk1;k<pnl;k++)
                for(i=ncpy;i<nr-ncpy;i++)
                        for(j=ncpx;j<nc-ncpx;j++)
                        {
                                p1z=p1z+hh[k][i][j];
                                ic+=1;
                        }
        p1z=p1z/ic;

        ic=0;
        p2z=0;
        kk2=nl-ncpz-pnl;
        if(kk2>=ncpz)
                kk2=ncpz;
        for(k=pnl;k<nl-ncpz+kk2;k++)
                for(i=ncpy;i<nr-ncpy;i++)
                        for(j=ncpx;j<nc-ncpx;j++)
                        {
                                p2z=p2z+hh[k][i][j];
                                ic+=1;
                        }
        p2z=p2z/ic;

        d12=dz*(double(pnl-(ncpz-kk1))/2+double(nl-ncpz+kk2-pnl)/2);// modified August 7, 2009
        zjz=(p2z-p1z)/d12;
}
//-----------------------------------------------------------------
//            solver the overdetermination equations(overdet)
//-----------------------------------------------------------------
void Overdet(vector <vector <double> >coeff,vector <double> rhs2,vector <double> &tmb,int nunk,
                        int neq,int mne)
{
        int i,j,k,mu,maxnu;
        double sum;
        mu=10;maxnu=10;
        vector <double> indx (mu,0);
        vector <vector<double> > ata (maxnu,vector<double> (maxnu,0) );

        for (i=0;i<nunk;i++)
                {
                        for(j=0;j<nunk;j++)
                        {
                                ata[i][j]=0;
                                for(k=0;k<neq;k++)
                                        ata[i][j]=ata[i][j]+coeff[k][i]*coeff[k][j];
                        }
                }

                for (i=0;i<nunk;i++)
                {
                        tmb[i]=0;
                        for (j=0;j<neq;j++)
                                tmb[i]=tmb[i]+coeff[j][i]*rhs2[j];
                }

                ludcmp(ata,nunk,indx,sum);
                lubksb(ata,nunk,indx,tmb,sum);

                for (i=0;i<neq;i++)
                {
                        for (j=0;j<nunk;j++)
                                rhs2[i]=rhs2[i]-coeff[i][j]*tmb[j];
                }

}
//---------------------------------------------------------//
//                      ludcmp(from "Numerical Recipes")
//---------------------------------------------------------//
void ludcmp(vector <vector<double> > &ata,int n,vector <double> &indx, double &sum)
{
        #define nmax 100
        #define tiny 1E-20
        int i,j,k,d,imax;
        d=1;
        double aamax,dum;
        vector <double> vv (nmax,0);
        for (i=0;i<n;i++)
        {
                aamax=0;
                for (j=0;j<n;j++)
                {
                        if (fabs(ata[i][j])>aamax)
                                aamax=fabs(ata[i][j]);
                }

                if (aamax==0)
                        cout<<"matriz singular"<<endl;

                vv[i]=1/aamax;
        }

        for(j=0;j<n;j++)
        {
                if (j>0)
                {
                        for (i=0;i<j;i++)
                        {
                                sum=ata[i][j];
                                if(i>0)
                                {
                                        for(k=0;k<i;k++)
                                                sum=sum-ata[i][k]*ata[k][j];
                                        ata[i][j]=sum;
                                }
                        }
                }
                aamax=0;

                for (i=j;i<n;i++)
                {
                        sum=ata[i][j];
                        if(j>0)
                        {
                                for(k=0;k<j;k++)
                                        sum=sum-ata[i][k]*ata[k][j];
                                ata[i][j]=sum;
                        }
                        dum=fabs(sum)*vv[i];
                        if (dum>=aamax)
                        {
                                imax=i;
                                aamax=dum;
                        }

                }
                if(j!=imax)
                {
                        for(k=0;k<n;k++)
                        {
                                dum=ata[imax][k];
                                ata[imax][k]=ata[j][k];
                                ata[j][k]=dum;
                        }
                        d=-d;
                        vv[imax]=vv[j];
                }

                indx[j]=imax;

                if(j!=(n-1))
                {
                        if(ata[j][j]==0)
                                ata[j][j]=tiny;
                        dum=1/ata[j][j];
                        for(i=j+1;i<n;i++)
                                ata[i][j]=ata[i][j]*dum;
                }

        }

        if(ata[n-1][n-1]==0)
                ata[n-1][n-1]=tiny;

}
//----------------------------------------------------------//
//                      lubksb(from"Numerical Recipes")
//----------------------------------------------------------//
void lubksb(vector <vector<double> > &ata, int n,vector <double> &indx,vector <double> &tmb,
                        double &sum)
{
        int i,j,ii,ll;
        ii=0;

        for (i=0;i<n;i++)
        {
                ll=indx[i];
                sum=tmb[ll];
                tmb[ll]=tmb[i];
                if(ii!=0)
                {
                        for(j=ii-1;j<i;j++)
                                sum=sum-ata[i][j]*tmb[j];
                }
                else if(sum!=0)
                        ii=i+1;
                tmb[i]=sum;
        }

        for(i=(n-1);i>=0;i--)
        {
                sum=tmb[i];
                if(i<(n-1))
                {
                        for(j=i+1;j<n;j++)
                                sum=sum-ata[i][j]*tmb[j];
                }

                tmb[i]=sum/ata[i][i];
        }

}
```

B
Input and Output Files

B.1 Parameter File

```
**********************************************************************
                            Parameter file
**********************************************************************
(Line 1)  100 50  50                              ** ncmf, nrmf, nlmf
(Line 2)  1   1   1                               ** wcmf, wrmf, wlmf
(Line 3)  10  10  10                              ** ncpmfx, ncpmfy, ncpmfz
(Line 4)  5   5   5                               ** ncpmix, ncpmiy, ncpmiz
(Line 5)  10  15  5                               ** ncmg, nrmg, nlmg
(Line 6)  10 10 10 10 10 10 10 10 10 10           ** wcmg
(Line 7)  10 10 10 10 10 10 10 10 10 10
          10 10 10 10 10                          ** wrmg
(Line 8)  10 10 10 10 10                          ** wlmg
(Line 9)  3                                       ** ib
(Line 10) 32                                      ** itt
(Line 11) 8                                       ** ngra
(Line 12) 1   0   0
          0   1   0
          0   0   1
          1   1   0
          1   0   1
          0   1   1
```

```
            1    1    1
            1   -1    1                             ** gra
(Line 13)  -1   -1   -1   -1   -1   -1
           -1   -1   -1   -1   -1   -1
           -1   -1   -1   -1   -1   -1
           -1   -1   -1   -1   -1   -1
           -1   -1   -1   -1   -1   -1
           -1   -1   -1   -1   -1   -1
           -1   -1   -1   -1   -1   -1
           -1   -1   -1   -1   -1   -1              ** icon
(Line 14)  0                                        ** debug
```

Line 1: number of columns, rows and layers at the fine scale.
Line 2: size of each cell at the fine scale.
Line 3: number of skin cells surrounding the coarse model.
Line 4: number of skin cells used to upscale each block. Note the inner skins cannot exceed outer skins.
Line 5: number of columns, rows and layers at the coarse scale.
Line 6: (variable length) width of each block along the row direction ("ncmg" in total).
Line 7: (variable length) width of each block along the column direction ("nrmg" in total).
Line 8: (variable length) width of each block along the layer direction ("nlmg" in total). Fig.B.1 depicts the meaning of these parameters in 2D.
Line 9: block conductivity ("ib"= 1) or interblock conductivity ("ib"= 2 in 2D and "ib"= 3 in 3D).
Line 10: full tensor ("itt"= 31 at the center of block and "itt"= 32 at the interface) or simple averaging such as geometric mean and power low ("itt"= 11).
Line 11: number of directions used to solve locally the groundwater flow equation for each upscaled block.
Line 12: coordinates of the vector indicating the direction of the "ngra" gradients (i.e., "0 1 0" indicates a head gradient parallel to the y -axis, "0 0 1" indicates a head gradient parallel to the z-axis, or "1 1 1" indicates a head gradient parallel to the diagonal of a cube.
Line 13: boundary condition ("ngra" in total). "-1" means prescribed head boundary, it is the only option in the present implementation.
Line 14: a flag for debug: there are 5 levels for debugging, i.e., $0 \sim 4$, the higher the value the more information printed.

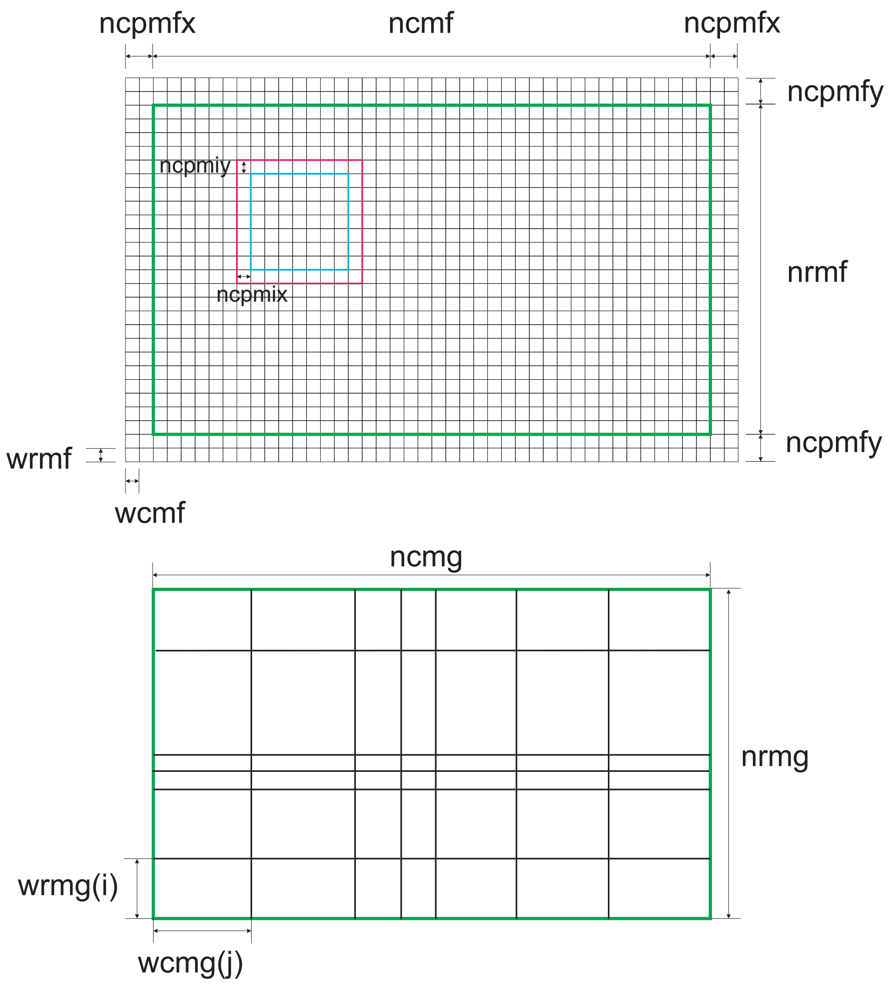

Figure B.1. Sketch explaining some of the input parameters in a 2D case

B.2 Conductivity Input File

The input data file contains the hydraulic conductivity at the fine scale. Recall that because the upscaling process requires the use of a skin, the size of this conductivity field should be larger than the size of the upscaled model by an amount which is indicated in the input parameter file. The program accepts conductivity at the fine scale following the GSLIB convention, i.e., the conductivity data are read in the sequence of layer by layer, row by row and column by column starting from the down left corner.

B.3 Output File

Number of output files depends on whether the block conductivity is located in the center or at the interface of the block. There will be one output file for the former and three for the latter case, in this case, interblock conductivities are printed separately according to the orientation of the interface, one file for interblock conductivities between columns, one between rows and and one between layers, for a 3D case. Note that there will be an upscaled tensor for each block if the user is interested in the conductivity at the block center; however, in the interblock case, the number of output upscaled conductivity corresponds to the number of interfaces rather than the number of blocks, e.g, the interblock conductivity between columns (aligned to x-axis) equals (No. of layers) $\times$ (No. of rows) $\times$ (No. of columns -1). Note that the upscaled conductivity is also printed in GSLIB format. For example, the output file “upk between col.dat” in 3D case, would start like this

```
Block conductivity full tensor between columns
3
kxx
kxy
kxz
0.55    0.01    -0.2
1.24    0.05    -0.11
...
```